A Simple Approach to DNS DoS Mitigation

Hitesh Ballani, Paul Francis
Cornell University, Ithaca, NY

ABSTRACT

We consider DoS attacks on DNS where 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 argue that a minor change in the caching behavior of DNS resolvers can significantly mitigate 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. This, in effect, implies that DNS resolvers store the part of the global DNS database that has been accessed by them but use it *only* when the relevant DNS servers are unavailable. While such a change to DNS resolvers also changes DNS semantics, we show 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 alleviating the impact of DoS attacks on DNS.

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 [16–19]. The frequency of such attacks on DNS can be attributed to a number of factors including, but not restricted to:

- Its pivotal role as a precursor to almost all Internet services implies that a common attack mechanism applies to a large number of services.
- Its connectionless and mostly unauthenticated mode of operation.
- The limited redundancy in nameservers and the resulting limited attack resilience [14]. Consequently, in many cases, it is easier to attack the DNS servers for a service than the actual application servers.
- Shared deployments whereby a single commercial DNS provider offers DNS services to a large number of customers are especially attractive targets due to the large attack impact [19].[1]

In response to such attacks, some of the DNS root-servers and top-level domain servers have been replicated through IP Anycast [7].

Lately, a number of research efforts have proposed new architectures for Internet's naming system. These architectures, among other things, aim to increase DNS robustness by ensuring system availability in the face of attacks. For instance, efforts arguing for a centralized DNS infrastructure [5] and a peer-to-peer based DNS infrastructure [4,13,15] represent the two extremes of this design space.

Alternatively, a complimentary tact to handle attacks on the infrastructure is to do away with the need for 100% availability. Specifically, in the case of DNS, this would entail ensuring that when the nameservers for a DNS zone are unavailable, most names in the zone can still be resolved and hence, most services in the zone are still accessible. For example, Kangasharju et. al. [9] achieve this by multicasting the global DNS database to specialized servers while Handley et. al. [6] propose a peer-to-peer design to do the same. However, we are not convinced of the need for a new dissemination mechanism to ensure DNS operation when nameservers are unavailable. In this paper we take a much more modest path 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

[1] Although it could be argued that shared deployments make attacks harder by amortizing the effort of a planned, robust server deployment.

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 have the stale records to rely upon and hence, DNS availability would be less critical that it is today. We show that the stale cache can be maintained on the disk and hence, our proposal boils down to using disk-space at the resolvers to negate the impact of DoS attacks. Further, our scheme has a number of practical advantages with regards to protection against such attacks; we discuss these in section 3.1.

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 3.2. 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.[2] 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[3]:

[2]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.

[3]This is a simplification of the algorithm used by resolvers but suffices for the purpose of exposition. See [10] for a more detailed version.

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.cs.cornell.edu*, the resolver determines if records regarding the authoritative nameservers for the zones *.cs.cornell.edu*, or *.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.

2.2 Proposed Resolver Modification

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.

Changing the caching behavior of DNS resolvers so that they shoulder more of the resolution burden, especially when nameservers are unavailable, is possible within 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.

This modification implies that when (and only when) the authoritative nameservers for a zone are unavailable, the resolver can resort to 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.cs.cornell.edu*, the resolver may query the authoritative nameservers of the zone *.cornell.edu* for the authoritative nameservers of the sub-zone *.cs.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 *.cs.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 *.cornell.edu* indicating that the sub-zone *.cs.cornell.edu* no longer exists will also lead to eviction of the existing nameserver record for *.cs.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.

2.3 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 *.cornell.edu*'s nameserver for the nameservers of the sub-zone *.cs.cornell.edu*, it can lookup its stale cache for information regarding the nameservers for *.cs.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 *.cornell.edu* and hence, the latency of the stale cache lookup is not critical. Consequently, the stale cache can be maintained on the resolver's disk.

Given an estimated size of ≈65GB for the global DNS database [5] and the fact that resolvers maintain the stale cache on their disk, it is not far fetched to imagine that resolvers store responses for all queries that they have issued in their stale cache. In practice, we expect resolvers to assign some maximum storage space for the stale cache and utilize a popularity-based eviction algorithm (for example, LRU) when the space fills up. To come up with a back of the envelope estimate for the required storage space, we consider a week-long DNS trace collected at MIT's border router by Jung et. al. in 2001 [8]. The trace contained ≈350,000 distinct names. With an average of 100 bytes for the record(s) of each name, this would amount to 35MB of data. While DNS traffic is bound to have increased since the time this trace was collected, we can safely say that resolvers can store the responses to all queries made by them for the duration of a week with a small amount of storage space.

3 DISCUSSION

A more "clean-slate" approach to make the availability of specific nameservers less critical for the operation of Internet's naming system would be to replicate the entire DNS database at all resolvers and have authoritative nameservers only disseminate the updates for the records in their zones ([6,9] exemplify two possible approaches to achieve this). However, apart from the likelihood of the dissemination process itself being prone to attacks, any such 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) [13] and the long timeouts used by resolvers in case of errors [12]. Consequently, despite a number of proposals arguing to the contrary [4–6,9,13,15], 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.

3.1 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 stale cache. DNS's hierarchical structure entails that almost all modified resolvers would have information for zones higher up in the hierarchy, such the DNS root-zone and the top-level zones, stored in their stale cache. Hence, the proposal would significantly reduce the impact of DoS attacks on such zones.

As a matter of fact, popularity of DNS names follows a zipf-like distribution [8]. Consequently, stale responses for a large fraction of queries to be issued by a resolver in the near future should already be present in the resolver's stale cache. Thus, having the stale cache in place insures the resolver (and its clients) from DoS attacks against DNS nameservers since a large fraction of queries can, if needed, be answered based on the records in the stale cache.

Simplicity. The biggest argument in favor of this proposal 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.

3.2 Objections

DNS caching semantics and the possibility of obsolete 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 records served by the zone's authoritative nameservers to be completely expunged by resolvers within TTL seconds.[4] 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, its has the zone's obsolete records in its stale cache. We don't place any bound on the time for which the records can be kept in the stale cache. So, 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 obsolete records present in the stale cache.

The problematic scenario described above arises only when all the authoritative nameservers for a zone are unavailable. In such a scenario, existing resolvers would fail to resolve any queries pertaining to the zone or any of its sub-zones (assuming that the records for the sub-zones are not present in the resolver cache). For the modified resolvers, if the resolver has not accessed the zone since the zone's records were last updated, it would use obsolete information. While this is far from perfect, the small possibility of obsolete information being used seems like a small price to pay for the robustness offered by having a stale cache in place. Also, the possibility of a resolver using obsolete information for a zone is much less for zones that the resolver frequently accesses.

Further, resolvers may choose to apply the modified caching scheme to infrastructure records only. Infrastructure records, as defined by [11], 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 very infrequently [6,11] and hence, this would further reduce the possibility of resolvers using obsolete information while still providing a large robustness gain.

Attackers attempting to force the use of obsolete information. Apart from the possibility of obsolete data being used, there is also the possibility of attackers taking advantage of the stale cache maintained by resolvers to

[4]In practice, zone operators need to be more flexible due to a large number of misbehaving resolvers that disregard TTL values and use expired records even though the nameservers for a zone are available [22].

force the use of obsolete data. 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 this is certainly a possibility, we have not been able to imagine scenarios where this would be worse than not being able to access the zone at all (which is the case with the status quo).

Autonomy for zone operators. A related 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 *.cornell.edu* needs to kill off the sub-zone *.cs.cornell.edu*. Typically, this would involve zone *.cornell.edu*'s operator configuring the zone's authoritative nameservers to respond to any queries regarding *.cs.cornell.edu* with a NXDOMAIN, implying that no such domain exists. Consequently, a resolver trying to resolve a query like the *A* record for *www.cs.cornell.edu* by traversing down the DNS zone hierarchy would receive a NXDOMAIN response from one of the nameservers for *.cornell.edu* 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 nameservers for *.cornell.edu* are unavailable, the resolver would still return a NXDOMAIN response.

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 [21] 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 [12].

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.[5] 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.[6] Hence, any DNSSec policies expressed by the resolver operator for undetermined records naturally apply to the stale records.

4 RELATED WORK

A number of recent efforts [4–6,13,15] have proposed new architectures for the next generation Internet naming system that address DNS's performance and robustness problems. 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.

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

[6]Undetermined records correspond to records resulting from a non-DNSSec lookup [20].

Pappas et. al. [11] 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 does not completely offset the impact of DoS attacks (since a fraction of the records still expire at any given time) and makes it harder for operators to update their records.

Cohen and Kaplan [3] 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. As a contrast, we argue for the use of a zone's stale records only in case all nameservers for the zone are unavailable. CoDNS [12] 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-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 their disk-space to insure themselves (and their clients) against DoS attacks on DNS.

5 Future Work

This paper presents a very simple modification to the caching behavior of DNS resolvers. While the impact of changing a resolver on the resolver's ability to cope with DoS attacks on nameservers is pretty obvious, we would like to use real-world traces to quantify this impact. To this effect, we are in the process of obtaining traces collected at DNS resolvers and aim to determine the impact of different kind of attacks. For example, assuming an attack that takes down the DNS root-servers, how many queries for a typical resolver will fail, how many will use obsolete information (in case the nameservers for one of the TLDs changes during the attack), and how many will benefit from having the stale cache in place?

We also aim to implement this modification into common DNS resolvers (for example, BIND, DBJDNS, etc.) or even as an add-on to the CoDNS resolution service [12] running on PlanetLab [2]. 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 Paul Vixie at ISC for helpful discussions on why this proposal should "not" be incorporated in DNS resolvers; most of the objections discussed in section 3.2 arose during exchanges with him.

References

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

[2] Chun, B., Culler, D., Roscoe, T., Bavier, A., Peterson, L., Wawrzoniak, M., and Bowman, M. PlanetLab: An Overlay Testbed for Broad-Coverage Services. *ACM SIGCOMM Computer Communication Review 33*, 3 (July 2003).

[3] Cohen, E., and Kaplan, H. Proactive caching of dns records: Addressing a performance bottleneck. In *Proc. of Symposium on Applications and the Internet* (2001).

[4] Cox, R., Muthitacharoen, A., and Morris, R. T. Serving DNS using a Peer-to-Peer Lookup Service. In *Proc. of IPTPS* (2002).

[5] Deegan, T., Crowcroft, J., and Warfield, A. The Main Name System: An Exercise in Centralized Computing. *SIGCOMM Comput. Commun. Rev. 35*, 5 (2005).

[6] Handley, M., and Greenhalgh, A. The Case for Pushing DNS. In *Proc. of Hotnets-IV* (2005).

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

[8] Jung, J., Sit, E., Balakrishnan, H., and Morris, R. DNS performance and the effectiveness of caching. *IEEE/ACM Trans. Netw. 10*, 5 (2002).

[9] Kangasharju, J., and Ross, K. W. A Replicated Architecture for the Domain Name System. In *Proc. of INFOCOM* (2000).

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

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

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

[13] Ramasubramanian, V., and Sirer, E. G. The Design and Implementation of a Next Generation Name Service for the Internet. In *Proc of ACM SIGCOMM* (2004).

[14] Ramasubramanian, V., and Sirer, E. G. Perils of Transitive Trust in the Domain Name System. In *Proc. of ACM SIGCOMM IMC* (2005).

[15] Theimer, M., and Jones, M. B. Overlook: Scalable name service on an overlay network. In *Proc. of ICDCS* (2002).

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

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

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

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

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

[21] Internet Systems Consortium, Aug 2006. `http://www.isc.org/`.

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