Measuring Availability in the Domain Name System

Casey Deccio
Sandia National Laboratories
ctdecci@sandia.gov

Jeff Sedayao and Krishna Kant
Intel Corporation
{jeff.sedayao,krishna.kant}@intel.com

Prasant Mohapatra
University of California, Davis
pmohapatra@ucdavis.edu

Abstract—**The domain name system (DNS) is critical to Internet functionality. The availability of a domain name refers to its ability to be resolved correctly. We develop a model for server dependencies that is used as a basis for measuring availability. We introduce the minimum number of servers queried (MSQ) and redundancy as availability metrics and show how common DNS misconfigurations impact the availability of domain names. We apply the availability model to domain names from production DNS and observe that 6.7% of names exhibit sub-optimal MSQ, and 14% experience false redundancy. The MSQ and redundancy values can be optimized by proper maintenance of delegation records for zones.**

I. Introduction

The Domain Name System (DNS) is a fundamental part of today's Internet. It is utilized by users and applications to map Internet names to addresses, which are required by machines for communication. Critical Internet systems rely on both responsiveness and accuracy of DNS for proper functionality.

Resolution of domain names often requires the resolution of other intermediate domain names. Such relationships form a graph of name dependencies which are collectively responsible for correct resolution of the dependent names. Subtle graph behaviors may go unnoticed in production DNS environments, but may increase potential for failure or attack. In many cases, this potential is caused by misconfiguration of downstream dependencies of a domain name, regardless of the security and robustness applied to the configuration of the domain name itself.

The availability of a domain name refers to its ability to be reliably resolved using DNS. In this paper we formalize the concept of domain name availability and develop a model for measuring availability. We show how common DNS misconfigurations apply to this model and discuss their impact quantitatively using metrics derived from the model. Using current DNS data we analyze the state of DNS in light of the availability model and discuss our observations and inferences.

The primary contributions of this paper are:

- A model formalizing name server dependencies in DNS.
- Metrics quantifying the availability of a domain name.
- A quantitative study of the impact of DNS misconfigurations on availability.

This research was supported in part by the National Science Foundation through the grant CNS-0716741.
Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

The metrics introduced in this paper characterize DNS availability in terms of the number of servers that must be queried for resolution and the level of server redundancy.

In Section II we discuss previous research related to that presented in this paper. Section III provides a description of the workings of DNS and the concept of dependencies within DNS. In Section IV we introduce metrics to measure the availability of domain names, as well as discussion on common DNS misconfigurations. Section V describes our methodology for data collection and contains an analysis of live DNS data. We conclude in Section VI.

II. Previous work

DNS name dependencies are analyzed in [1], in which the potential for a large number and variety of servers affecting name resolution is demonstrated. A formal model for DNS name dependencies is introduced in [2], in which dependencies are represented by edges between domain names in a directed graph. In this paper we extend the DNS name dependency model to name servers, and the resulting model is used as a basis for measuring availability.

Surveys of the state of DNS are presented in [3]–[5]. Various misconfigurations are analyzed, including lame delegation, diminished server redundancy, cyclic dependencies, and NS RR inconsistency. DNS availability and robustness have been studied in [6], [7]. These studies are largely based on empirical analysis, whereas in this paper we derive a theoretical availability model and methodology to systematically identify such misconfigurations and quantify their impact on availability.

The DNS Security Extensions (DNSSEC) [8]–[10] are the industry-accepted standard for cryptographically validating DNS queries. Proper application of DNSSEC foils attacks on DNS integrity. However, just like unsigned names, DNSSEC-signed names also rely on the availability of their dependencies. In fact signed zones actually have greater reliance on such dependencies, such as the "chain of trust" between parent and child zones. The model presented in this paper thus forms a useful analysis for both signed and unsigned domain names.

III. DNS and dependencies

In this section we describe the DNS protocol, as it pertains to this research, and discuss the idea of dependencies in DNS. We refer to the fictitious zone data in Table I for examples throughout the remainder of this paper.

The DNS namespace is hierarchical. Each domain name is comprised of a dot-separated list of *labels* describing its ancestry from left to right. For example, *foo.net* is a *subdomain*,

$ORIGIN foo.net.

	Name	Type	Value
1	@	NS	ns1
2	@	NS	ns2
3	@	NS	ns1.bar.com.
4	@	NS	ns3.bar.com.
5	ns1	A	192.0.2.1
6	ns2	A	192.0.2.2

$ORIGIN bar.com.

	Name	Type	Value
1	@	NS	ns1
2	@	NS	ns2
3	ns1	A	192.0.2.5
4	ns2	A	192.0.2.6
5	ns3	A	192.0.2.7

$ORIGIN net.

	Name	Type	Value
1	@	NS	ns1
2	@	NS	ns2
3	ns1	A	192.0.2.3
4	ns2	A	192.0.2.4
5	foo	NS	ns1.foo
6	foo	NS	ns2.foo
7	foo	NS	ns1.bar.com.
8	foo	NS	ns3.bar.com.
9	ns1.foo	A	192.0.2.1

$ORIGIN com.

	Name	Type	Value
1	@	NS	ns1
2	ns1	A	192.0.2.8
3	bar	NS	ns1.bar
4	bar	NS	ns2.bar
5	ns1.bar	A	192.0.2.5
6	ns2.bar	A	192.0.2.6

TABLE I
EXAMPLE ZONE DATA USING RFC 1035 NOTATION [11].

or descendant, of *net*. All names below the root domain (".") are *delegated* to other organizations for administration, which may in turn delegate a portion of their namespace. A *zone* is an autonomous portion of namespace typically administered by a single organization as a result of such delegations. For example, *foo.net* is delegated by *net*. For any zone, z, a set of servers are designated and configured as *authoritative*. The names of these servers are maintained in the z zone using NS (name server) resource records (RRs). The delegation from zone $Parent(z)$ to z is handled by maintaining corresponding NS RRs in the $Parent(z)$ zone as *delegation records*. The sets names of servers authoritative for z maintained in z and $Parent(z)$ are denoted NS_z and NS'_z, respectively. In our example, the authoritative and delegation NS RRs for *foo.net* are found on *foo.net* lines 1–4 and *net* lines 5–8, respectively.

For a resolver to query a server corresponding to name $u \in NS_z$, it must first resolve the address(es) corresponding to u. If u is a subdomain of z, administrators of $Parent(z)$ should include a corresponding address (A- or AAAA-type) RR in $Parent(z)$ as a *glue record* for u to "bootstrap" resolution [12]. Like delegation records, glue records are maintained in the parent zone independently from their authoritative values and must be kept in sync for proper functionality. A glue record for *ns1.foo.net* is shown on line 9 of the *net* zone. The glue record for *ns2.foo.net* is missing; this type of misconfiguration is addressed in Section IV-B.

In response to a query for a name in z, a name server authoritative for $Parent(z)$ responds with the set of delegation NS RRs for z to direct the resolver to the proper servers. The response also contains pertinent address records, such as glue records, data from zones for which it is authoritative, or data from its cache. However, it is recommended that the resolver only trust address records if their names are *in-bailiwick*—that is, if they are subdomains of $Parent(z)$. Otherwise, the resolver must look up the addresses itself [2], [13].

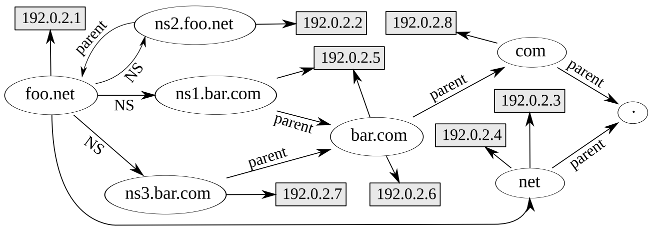

Fig. 1. The server dependency graph for *foo.net*, derived from the zone data in Table I. The gray, rectangular nodes represent name servers, and the oval nodes represent domain names.

In the remainder of this section we discuss name and server dependencies, which both contribute to the performance, security, and robustness of name resolution.

A. Name dependencies

Name dependencies in DNS exist both as part of its hierarchical structure and as a result of explicit configuration. For example, *foo.net* depends on *net* since a resolver must trust the delegation information provided by its parent. The name *ns1.bar.com* is also a dependency of *foo.net* because it is not in-bailiwick and therefore requires a resolver to determine its address before it can send it queries.

The collection of dependencies for the resolution of domain name d is modeled as a directed graph, $G_d = (V_d, A_d)$, with edges representing direct dependencies between names [2]. This name dependency graph is the foundation for name resolution, and server dependencies are derived from this model. The dependency graph for *foo.com* is shown in Fig. 1.

The research in [2] considers both *active* and *passive* influence. Active influence signifies a true dependence—one name must be resolved before another. Passive influence is the result of name servers giving preference to data received from authoritative sources over data from glue and other sources [13]. Since this paper is concerned with domain name availability, we consider only active influence in our model.

B. Server dependencies

Just as a domain name may be dependent on other domain names, it also depends on name servers, identified by IP address. We model server dependencies by extending the active influence name dependency graph [2] for domain name d, $G'_d = (V'_d, A'_d)$, to include name servers and direct server dependencies. The resulting graph, $H_d = (W_d, B_d)$, has properties $B_d = A'_d \bigcup \{\text{direct server dependencies}\}$ and $W_d = V'_d \bigcup \{\text{name servers}\}$. We explain the methodology for adding server dependencies to the graph in the remainder of this section. Edges in H_d are not weighted, as weights have no significance in our study of availability.

A *direct dependency* between domain name u and server s, $u, s \in W_d$, exists in two cases. If s is the address corresponding to the glue record for an in-bailiwick name in NS_z, then we add edge (z, s) to B_d. Because a glue record for s is sent to the resolver, the resolver isn't dependent on the server's name, only on its address. The edge between *foo.net* and 192.0.2.1 is an example of such a dependency.

If u resolves to s, then we add edge (u, s) to B_d; the resolver must resolve u before it can query s. For example, *ns3.bar.com* depends on 192.0.2.7. Ultimately each zone in W_d is directly or indirectly dependent on the servers that answer authoritatively for each. Fig. 1 shows the server dependency graph derived from the data in Table I.

IV. Domain name availability

Using the server dependency graph H_d, we can begin to analyze the question of *availability* of d—whether or not the name can be resolved. The availability of a name depends on the contents of the resolver's cache. Specifically, we analyze two states of a resolver with regard to its cached knowledge about a particular zone, z: *knowledgeable* and *ignorant*.

If the resolver has cached both the names and addresses of servers authoritative for zone z, then we say that the resolver is knowledgeable about z. Since the resolver knows the collective addresses for the servers that are (reportedly) authoritative for z, its Boolean availability is based on at least one of the name servers authoritative for z responding authoritatively.

A resolver that is *ignorant* about zone z has no information about the names or addresses of servers authoritative for z in its cache. In order to become knowledgeable about zone z, it must learn NS_z and NSA_z through the standard name resolution process. Transitioning from an ignorant to a knowledgeable state involves learning indirect server dependencies in H_d using direct server dependencies (e.g., glue records) as "knowledge anchors". Caching allows a resolver to remain knowledgeable about a zone until the pertinent RRs expire in its cache. Since caching is temporary we consider only the more general view of availability, as seen through an ignorant resolver.

When evaluating availability for a domain name, each of its dependencies must be considered relative to one another. For example, a resolver only requires response from one of the servers authoritative for zone z. However, it relies on $Parent(z)$ regardless of which server or NS dependency is queried. Likewise for non-zone names, an alias dependency and any direct server dependencies are collectively mutually exclusive [13], but the parent of the name is required independent of the others. This concept is displayed for *foo.net* in Fig. 2, with symbolic *OR* nodes grouping mutually exclusive dependencies and *AND* nodes grouping all required dependencies. Because *ns2.foo.net* lacks a glue record, it depends on *foo.net*. This cyclic behavior is discussed in Section IV-B.

A. Minimum servers queried and redundancy

Having a formal model of server dependencies allows us to derive metrics to measure the availability of a domain name. It may be possible, using server availability as a basis, to recursively calculate a single normalized value which defines the availability of the domain name. However, such a metric would only serve a historical purpose and would not represent availability from the perspective of robustness. We introduce two metrics for analyzing the server dependency graph of d, H_d: *minimum number of servers queried (MSQ)* and *redundancy*. Both are defined with the assumption that the resolver is ignorant.

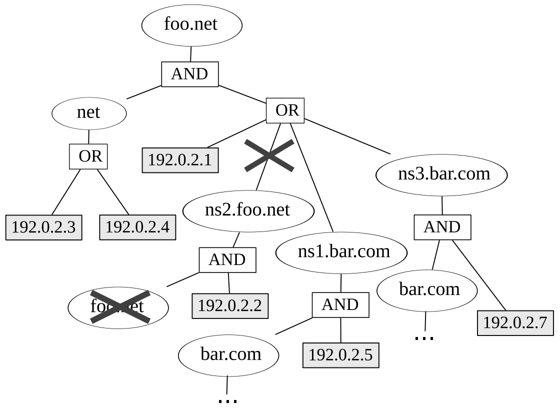

Fig. 2. A logical tree describing the availability of *foo.net*.

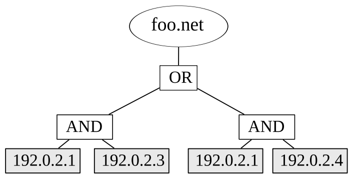

Fig. 3. The logical tree describing the MSQ of *foo.net*.

The MSQ for a domain name refers to the minimum number of servers which a resolver must query to resolve the name. Domain names with larger MSQs may result in additional resolution overhead for an ignorant resolver. However, caching minimizes overhead of subsequent lookups.

The MSQ for domain name d is returned by calling `FindMSQ`$(d, \emptyset)$ (Algorithm 1). In a logical sense, `FindMSQ` recursively performs a conversion of the Boolean availability expression for d, such as that shown in Fig. 2, into disjunctive normal form (DNF). Each resulting conjunction corresponds to a complete set of servers that may be queried to resolve d. Only the set of conjunctions having minimum size are returned from each call. Fig. 3 portrays the logical structure resulting from recursively reducing *foo.net* by calling `FindMSQ`(*foo.net*, $\emptyset$).

The MSQ is simply the size of any one of the sets returned by `FindMSQ`. The sets of servers comprising the MSQ returned for *foo.net* are: $\{192.0.2.1, 192.0.2.3\}$ and $\{192.0.2.1, 192.0.2.4\}$. Root servers are excluded from our example for simplicity, so we increase the MSQ by one to account for it. Thus, the minimum number of servers needed to resolve *foo.net* is 3.

The MSQ for a domain name is *optimal* if it is less than or equal to the number of zones in its ancestry, including the root zone. This accounts for a resolver querying a single server authoritative for each ancestor zone. Any queries required beyond this number constitutes a *sub-optimal* MSQ. For example, the MSQ of *foo.net* is optimal. Zones that completely outsource their DNS services to out-of-bailiwick servers (e.g., *foo.net* $\rightarrow$ *ns1.bar.com*) are among those that are prone to

Algorithm 1 `FindMSQ`(u, J)

```
Input: Domain name $u \in W_d$
Input: Set of names visited $J \subseteq W_d$
Output: Set of all sets of servers comprising MSQ for $u$
1: if $u \in J$ then /* cycle */
2:     return $\emptyset$
3: else if $u$ is a name server then /* knowledge anchor */
4:     return $\{\{u\}\}$
5: else if $MSQ(u)$ is already known then
6:     return $MSQ(u)$
7: $J \leftarrow J \bigcup \{u\}$ /* Add $u$ to history */
8: $MSQ_{Parent} \leftarrow$ FindMSQ$(Parent(u), J)$
9: if $u$ is a zone then
10:     /* Direct server and NS dependencies of $u$ */
11:     $D \leftarrow \{\forall v \in NSA'_u \bigcup NS'_u | \exists (u, v) \in B_d\}$
12: else
13:     /* Direct server and alias dependencies of $u$ */
14:     $D \leftarrow \{\forall v \in S_u \bigcup \{Cname(u)\} | \exists (u, v) \in B_d\}$
15: /* Find min. MSQ among mutually exclusive deps */
16: $MSQ_{Other} \leftarrow \emptyset$
17: $msq \leftarrow \infty$
18: for all $v \in D$ do
19:     $MSQ'_{Other} \leftarrow$ FindMSQ$(v, J)$
20:     $msq' \leftarrow \min_{s \in MSQ'_{Other}} |s|$
21:     if $msq' < msq$ then
22:         $MSQ_{Other} \leftarrow MSQ'_{Other}$
23:         $msq \leftarrow msq'$
24:     else if $msq = msq'$ then
25:         $MSQ_{Other} \leftarrow MSQ_{Other} \bigcup MSQ'_{Other}$
26: /* Find smallest union of $MSQ_{Parent}$, $MSQ_{Other}$ */
27: $msq \leftarrow \infty$
28: $MSQ \leftarrow \emptyset$
29: for all $MSQ'_P \in MSQ_{Parent}$, $MSQ'_O \in MSQ_{Other}$ do
30:     $MSQ' \leftarrow MSQ'_P \bigcup MSQ'_O$
31:     if $|MSQ'| < msq$ then
32:         $MSQ \leftarrow \{MSQ'\}$
33:         $msq \leftarrow |MSQ'|$
34:     else if $|MSQ'| = msq$ then
35:         $MSQ \leftarrow MSQ \bigcup \{MSQ'\}$
36: $MSQ(u) \leftarrow MSQ$ /* Store the value for future use */
37: return $MSQ$
```

have suboptimal MSQs because at least one additional lookup is required for the server names.

The redundancy is the size of the smallest set of redundant servers at any point in a required resolution path and might be considered the "availability bottleneck" of a domain name. If all servers comprising the redundancy of a domain name were to fail, then the name would be rendered unavailable. The methodology for determining the redundancy of a domain name is very similar to that for determining the MSQ. The difference is that rather than reducing to DNF, the logical expression is reduced to conjunctive normal form (CNF), returning a set of disjunctions. We have not included the actual redundancy algorithm in this paper to conserve space. The sets of servers comprising the redundancy of *foo.net* are: $\{192.0.2.1, 192.0.2.8\}$ and $\{192.0.2.3, 192.0.2.4\}$. That is say that if both 192.0.2.1 and 192.0.2.8 are unavailable or both 192.0.2.3 and 192.0.2.4 are unavailable, then *foo.net* is rendered unavailable.

As a point of reference, we compare the redundancy of domain name d to its configured redundancy, which is the size of the set of server names, NS_z, authoritative for its nearest ancestor zone, z. If the redundancy for d is less than $|NS_z|$, then the true redundancy is less than the redundancy configured by the administrator. We categorize such a case as *false redundancy*. False redundancy may exist when multiple names resolve to the same address or not all NS RRs for z are included in $Parent(z)$, so $|NSA'_z| < |NS_z|$. It could also result from a narrower bottleneck in downstream dependencies. The redundancy for *foo.net* is a false redundancy, as there are four server names in the set of NS RRs for *foo.net*, but the size of the redundancy set is 2.

B. DNS misconfigurations

DNS misconfigurations may lessen availability of a domain name. We discuss in this section several DNS misconfigurations and their relationship to domain name availability.

Lame delegation occurs when a server is included in the NS RRs as authoritative for a zone, but does not actually contain authoritative data for the zone. It can be caused by either incorrect NS RRs for a zone or a misconfiguration on the lame server itself. Lame delegation impacts the availability of a domain name. If a server s is lame for zone z, then edge (z, s) is effectually excluded from H_d, which potentially increases MSQ and decreases redundancy of d. Our survey of DNS showed 2.5% of authoritative servers as non-responsive and another 1.2% that either issued an error response or returned non-authoritative data.

Cyclic dependencies, discussed in [3], are identified by a cycle in the server dependency graph, H_d, and affects the availability of domain name d for resolvers which are ignorant of d. A cyclic name dependency can be caused by a missing glue record, such as that for *ns2.foo.net*, or it may be two names that otherwise require each other for proper resolution. Fig. 2 demonstrates the effect of cyclic dependencies when measuring the availability of a domain name. A name which is a dependency of itself is effectively "unavailable". For example, *foo.net* (the node below *ns2.foo.net*) cannot be relied on for resolving *foo.net* (the root node) because they represent the same name. This in turn makes *ns2.foo.net* unavailable. Cyclic dependencies potentially decrease the redundancy of a domain name for an ignorant resolver. We observed that 0.095% of the zones we examined exhibited self-dependence, 76% of which was caused by missing glue records. Glue records required for delegation records were missing 0.024% of the time.

Since delegation records for z are maintained in $Parent(z)$ independently from the authoritative NS RRs maintained in z, it is not uncommon for the two sets to be out of sync (i.e.,

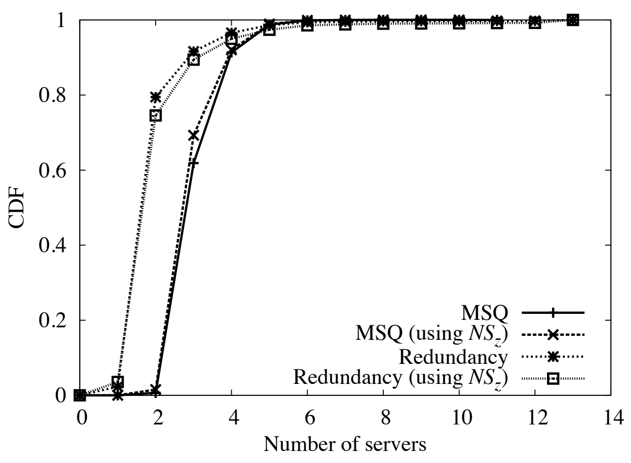

Fig. 4. The availability of a zone, in terms of MSQ and redundancy.

$NS_z \neq NS'_z$). Our survey showed that child/parent NS RR inconsistency exists in 20% of zones. Assuming that the set of authoritative server names for a zone is exactly correct, then extra delegation records lead to lame delegation, and missing delegation records potentially reduce redundancy and increase MSQ. Our survey showed that 6% of authoritative server names do not exist as delegation records, and 5% of delegation records do not exist in the authoritative zone data.

V. Data collection and availability analysis

For an availability analysis we extracted over 3 million hostnames from an April, 2009 index of URLs provided by the Open Directory (ODP) [1]. We added to these hostnames over 100,000 names issued as queries to recursive name servers at the 2008 Supercomputing conference (SC08) [2]. We built a graph of name and server dependencies for each of the ODP/SC08 hostnames by recursively following all dependencies of each name. The graph data included nearly 3 million zones.

In our survey we were unable to detect certain DNS configurations which affect availability. Requests sent to an *anycast* address are routed to one of multiple DNS servers, depending on source address and availability. Load balancers bear a single unicast address but distribute requests to multiple back-end servers. A *multi-homed* server responds to requests on multiple addresses. In our analysis we treat each IP address as a single server.

We assessed the availability of each of the ODP/SC08 names from our survey. Fig. 4 plots the MSQ and redundancy for the ODP/SC08 names as a cumulative distribution function (CDF). The average MSQ was 3.48, and 62% of names had an MSQ of 3 or less. However, the average MSQ decreased to 3.38 when the set of authoritative server names was used instead of the delegation records, and 69% of names had an MSQ of 3 or less. The names had an average redundancy of 2.34, and 79% of the names had a redundancy of less than 3. Only 3% of the names had a redundancy greater than 3. When the set of authoritative server names was used instead of the delegation records, the redundancy of 5% of the names increased to 3 or more. The differences in MSQ and redundancy when using the set of authoritative server names show the necessity of proper maintenance of delegation records in the parent zone.

We observed that 6.7% of ODP/SC08 names had sub-optimal MSQ values, and 14% exhibited false redundancy. We attribute the fraction of sub-optimal MSQ values to zones that outsource their DNS service to servers whose names are in out-of-bailiwick zones. The large percentage of names experiencing false redundancy demonstrates the impact that downstream dependencies can have on domain name availability.

[1]http://www.dmoz.org/

[2]http://sc08.supercomputing.org/

VI. Conclusion

In this paper, we formalize a model for name server dependencies in DNS, and using that model we derive metrics for quantifying the availability of domain names. We have observed that 14% of domain names experience lower redundancy than that with which they've been configured, and the minimum number of servers required to query (MSQ) for resolution was sub-optimal in 6.7% of domain names.

High availability and robustness were built into the DNS protocol, but proper design and configuration by DNS administrators are required for these behaviors to be displayed. Common misconfigurations can negatively affect domain name availability and consequently cause potential disruption of critical services. DNS administrators should be aware of the workings of DNS and the dependencies of administered domain names. Such knowledge will allow them to handle these important issues, and in turn, enhance the functionality and accuracy of DNS services.

References

[1] V. Ramasubramanian and E. G. Sirer, "Perils of transitive trust in the domain name system," in *IMC '05, Proceedings*, 2005, pp. 379–384.

[2] C. Deccio, C.-C. Chen, J. Sedayao, K. Kant, and P. Mohapatra, "Quality of name resolution in the domain name system," in *ICNP '09, Proceedings*, 2009.

[3] V. Pappas, Z. Xu, S. Lu, D. Massey, A. Terzis, and L. Zhang, "Impact of configuration errors on DNS robustness," in *SIGCOMM '04, Proceedings*, 2004, pp. 319–330.

[4] A. J. Kalafut, C. A. Shue, and M. Gupta, "Understanding implications of DNS zone provisioning," in *IMC '08, Proceedings*, 2008, pp. 211–216.

[5] "The measurement factory." [Online]. Available: http://dns.measurement-factory.com/surveys/

[6] R. Liston, S. Srinivasan, and E. Zegura, "Diversity in DNS performance measures," in *SIGCOMM '02, Proceedings*, 2002, pp. 19–31.

[7] 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 *IMC '04, Proceedings*, 2004, pp. 1–14.

[8] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose, "RFC 4033," 2005.

[9] ——, "RFC 4034," 2005.

[10] ——, "RFC 4035," 2005.

[11] P. Mockapetris, "RFC 1035," 1987.

[12] ——, "RFC 1034," 1987.

[13] R. Elz and R. Bush, "RFC 2181," 1997.