Quantifying and Improving DNS Availability

By

Casey Deccio
B.S. (Brigham Young University) 2002
M.S. (Brigham Young University) 2004

DISSERTATION

Submitted in partial satisfaction of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

Computer Science

in the

OFFICE OF GRADUATE STUDIES

of the

UNIVERSITY OF CALIFORNIA

DAVIS

Approved:

Prasant Mohapatra, Chair

Krishna Kant

S. Felix Wu

Committee in Charge

2010

Abstract

Quantifying and Improving DNS Availability

by

Casey Deccio

The Domain Name System (DNS) is one of the components most critical to Internet functionality. Nearly all Internet applications rely on the DNS for name-to-address translation. The ubiquity of the DNS necessitates both the accuracy and availability of responses. In this dissertation we present a model of DNS name resolution from which the availability of a domain name can be quantified in the context of its deployment. Using this model, DNS administrators will better understand the complex processes required to resolve domain names and quantitatively improve the robustness of their DNS configurations, from a perspective of availability.

We begin our analysis by providing relevant background on the DNS. We summarize protocol details surrounding name resolution, protocol and implementation vulnerabilities, and security extensions (DNSSEC).

Next we formalize a model for identifying DNS dependencies, based on DNS specification and server implementation. Using this model we introduce metrics to quantify the diversity of the namespace affecting the name resolution of a domain name. We observe that out of the set of zones influencing resolution of a domain name an average of 92% were explicitly configured by DNS administrators. However, certain resolver caching behaviors increase the likelihood that a domain name is influenced by third parties.

We further our DNS dependency model to describe DNS availability, a measure of the resolvability of a domain name. We derive a model and metrics for measuring availability and identify weaknesses in deployments. We identify specific misconfigurations that degrade the availability of a domain name and quantify their impact. In our analysis of production DNS data we observe that 14% of domain names exhibit lower redundancy than that which administrators have explicitly configured. We also observe that 6.7% of domain names required queries to more than an optimal number of servers to obtain an answer.

Our final analysis pertains to misconfigurations affecting availability in DNSSEC deployments. Because DNSSEC deployment is still new to administrators, many deployments have suffered from server misconfiguration or maintenance neglect which ultimately render a domain name unresolvable, even if servers are responsive. We introduce metrics for improving availability, and we present methodology for increased name resolution robustness in the presence of DNSSEC misconfiguration. In our survey of production signed zones, we observe that 31% of the validation errors detected might be mitigated using the technique proposed in our research.

The models and metrics presented in this dissertation can assist DNS administrators in better understanding their DNS deployments and avoiding name resolution failure through proper design and maintenance of DNS.

To my wife, Talia, and children, Zion, Hyrum, and Ezra.

Contents

List of Figures

List of Tables

Acknowledgments

Many people have contributed, inspired, or otherwise supported me to make completion of this dissertation possible. The direction from my adviser, Professor Prasant Mohapatra, has been very instrumental in helping me to progress towards completion. Despite his many responsibilities as department chair, professor, and adviser to other students, he always found time to provide feedback and direction on my research from home and abroad. His optimism has helped me keep the end goal in sight.

Dr. Krishna Kant and Jeff Sedayao, both of Intel Corporation, have provided perspectives and feedback to improve the quality of the content, writing, and results of my research. Members of the Networks Research Laboratory at UC Davis provided useful feedback on both my research ideas and presentation skills.

Professor S. Felix Wu guided me during the initial stretch of my Ph.D. research and encouraged me to pursue a topic that was I was passionate about and that I felt would make a difference. Knowing of my interest in DNS, he ultimately introduced me to Professor Mohapatra and George Chen, who were actively pursuing this research. That introduction made all the difference in my progress.

Management and other colleagues at Sandia National Laboratories have been extremely supportive in sponsoring and encouraging my educational success. Pursuing my Ph.D. from a remote location was often difficult because of the lack of direct contact with other students and professors. Dr. Tamara Kolda provided local mentoring and direction at a time when I was struggling with discouragement and lack of progress. In our regular "research therapy" meetings she helped familiarize me with the research process in general, and allowed me to sound out my own ideas.

My family has been supportive and understanding throughout this entire process. When I struggled for months with direction, they were sympathetic. When I worked late to meet deadlines or traveled to present research, they were understanding. They also helped me keep my life in proper perspective with regular family time, activities, and outings. Thank you Talia, Zion, Hyrum, and Ezra!

Finally, I thank my Heavenly Father for allowing me struggle but giving the strength and momentum to accomplish the task that I set out to do.

Chapter 1

Introduction

The Domain Name System (DNS) is one of the systems most critical to the Internet. Nearly all Internet applications rely on the DNS for proper function. Although Internet hosts communicate using the Internet Protocol (IP) and are identified by IP addresses, the DNS abstracts IP addressing from users and clients, so they can identify Internet locations using domain names representative of human-friendly words, titles, and abbreviations.

The use of domain names is an integral part World-wide Web. End users recognize them as part of the Uniform Resource Locator (URL) or "Web address" corresponding to a Web site. The names often intuitively reflect the name and nature of the organization with which they are associated, such as *www.ucdavis.edu*, which is the domain name for the Web server of the University of California, Davis, an institution of higher education. URLs are the glue forming the links of the Web itself.

Other applications also rely heavily on the DNS as a lookup service. Email servers use the DNS to look up mail exchange (`MX`) records for a domain, such as *ucdavis.edu*, to determine the server(s) to which they should direct email destined for that domain. Clients use DNS service (`SRV`) records to discover the server(s) providing for a particular service on the network, such as Kerberos or Lightweight Directory Access Protocol (LDAP). Reverse lookups for host identification (i.e., mapping IP addresses to domain names) are also accomplished using the DNS.

The layer of abstraction that the DNS provides between domain names and IP addresses allows network versatility. Even if the IP address of a resource changes, its domain name does not need to change; it may be modified in the DNS to point to the IP address, or it may alias another domain name that resolves to the appropriate IP address. A simple load balancing technique can be implemented in the DNS by configuring a domain name to resolve to multiple IP addresses. More complex solutions, such as Global Server Load Balancing (GSLB), attempt to identify the location of a client and respond dynamically to a request with the address within closest proximity.

1.1 DNS availability and security

The ubiquity of the DNS necessitates both the accuracy and availability of responses. Name resolution failure may interrupt critical services, and tainted DNS responses may direct users to malicious servers with the intent to steal sensitive information. The DNS must be resilient to both scenarios. Empirical studies have shown that deployed DNS servers generally have high availability [1]. However, name resolution in the DNS is quite complex and cannot be measured solely on server availability. DNS name resolution includes dependencies that reach beyond the administrative control of the name being resolved [2, 3]. Misconfiguration of a domain name or any of its dependencies can result in reduced robustness for the name [4].

The DNS is inherently insecure, as it was not originally designed with security in mind. Resolvers have no mechanism for verifying the validity of a response, so an altered DNS response may go undetected. The DNS Security Extensions (DNSSEC) [5, 6, 7] have been introduced as a security retrofit. DNSSEC responses may be cryptographically verified using public keys and signatures embedded in the DNS. However, the protocol and administrative complexity added by DNSSEC is nontrivial. Administrators face the challenge of properly maintaining a DNSSEC deployment to achieve security at the risk of reducing DNS availability if misconfigurations arise or maintenance is otherwise neglected. Early experience has demonstrated this challenge for those designing, implementing, and deploying DNSSEC [8, 9].

1.2 Objective and outline

The objective of this dissertation is to create a model of DNS name resolution from which the availability of a domain name can be quantified in the context of its deployment. Using this model, DNS administrators will better understand the complex processes required to resolve domain names and quantitatively improve the robustness of their DNS configurations, from a perspective of availability. We break up our analysis into several parts to achieve our objective: DNS dependencies; DNS availability; and DNSSEC availability.

DNS dependencies Formalizing a DNS dependency model is an essential first step in our work [3]. Many forms of dependencies exist in the DNS, and these directly affect reliable name resolution. Our dependency model provides an understanding and metrics to both describe and quantify the diversity of namespace and organizations affecting name resolution. Administrators using this model can identify the domains and organizations influencing resolution of their domains, and quantify this influence.

DNS availability We further our DNS dependency model to describe domain name availability [10]. Considering all dependencies required for proper resolution of a domain name, we derive a model and metrics for measuring availability and identify weaknesses in deployments, so-called "availability bottlenecks". We identify specific instances of misconfiguration that degrade the availability of a domain name. This model provides administrators with a comprehensive picture of name resolution for their deployed domains, so they may adjust their configurations appropriately to improve robustness.

DNSSEC availability Our model of DNSSEC approaches availability from different standpoint. Given the novelty of DNSSEC deployment, many deployments have suffered from server misconfiguration or maintenance neglect which ultimately render a domain name unresolvable, even if servers are responsive. While mostly independent of the dependency view of availability, which focuses on server responsiveness, this analysis is equally important when validating responses in a DNSSEC deployment. We introduce metrics for improving availability, and we present methodology for increased name resolution robustness in the presence of DNSSEC misconfiguration.

We preface our analysis with relevant DNS protocol background in Chapter 2, discussing name resolution, vulnerabilities, and security. In Chapter 3 we present our DNS dependency model and introduce metrics for quantifying the influence of domain name dependencies. In Chapter 4 we formalize our DNS availability model on the basis of name and server dependencies and measure availability with associated metrics. Chapter 5 describes our methodology for measuring DNSSEC availability, based on the likelihood of validation failure due to misconfiguration. In Chapter 6, we summarize our inferences and describe future extensions to the analysis presented in this dissertation.

Chapter 2

DNS Background

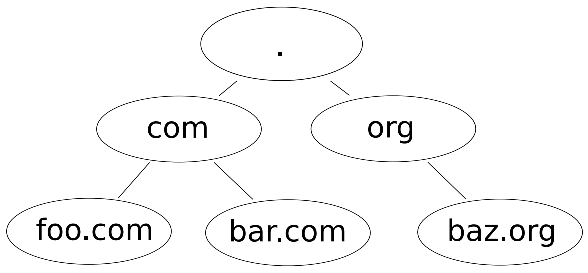

Figure 2.1: An example DNS zone hierarchy.

The Domain Name System (DNS)[11, 12] is a lookup service primarily known for mapping Internet names to addresses. It was designed as a robust and scalable database distributed among servers spread across the globe. In this chapter we provide an overview of the DNS, as it pertains to the research provided in this dissertation. We discuss name resolution in the DNS, and we provide a brief summary of vulnerabilities, attacks, and some proposed solutions. We then describe the DNS Security Extensions.

2.1 DNS fundamentals

The DNS namespace is organized hierarchically. Each *domain name* is comprised of dot-separated, alpha-numeric *labels*, which describe the ancestry of the name itself, from left to right. For example, the ancestry of *www.foo.com* is: *www.foo.com*, *foo.com*, *com*. All domain names are descendants, or *subdomains*, of the *root* domain (represented by a single dot with no labels: ".").

A *zone* is an autonomously managed piece of DNS namespace, typically administered by a single organization. The *origin* of a zone is the domain name at the top of the zone's namespace (e.g., *foo.com*). The administrator of a zone may *delegate* management of subdomain namespace, so the corresponding child zone may be administered by another individual or organization. For example, the *com* zone may delegate the administrative control of *foo.com*. An example zone hierarchy is shown in Fig. 2.1. For each zone a set of name servers are configured and advertised as *authoritative* to provide answers for non-delegated namespace in that zone. Servers may be authoritative for multiple zones.

$ORIGIN foo.com.				
	Name	TTL	Type	Value
1	foo.com.	3600	`NS`	ns1.foo.com.
2	foo.com.	3600	`NS`	ns2.foo.com.
3	foo.com.	3600	`NS`	ns.bar.com.
4	ns1.foo.com.	3600	`A`	192.0.2.1
5	ns2.foo.com.	3600	`A`	192.0.2.2
6	www.foo.com.	3600	`CNAME`	baz.foo.com.
7	baz.foo.com.	3600	`A`	192.0.2.3

$ORIGIN com.				
	Name	TTL	Type	Value
1	com.	3600	`NS`	ns1.com.
2	com.	3600	`NS`	ns2.com.
3	ns1.com.	3600	`A`	192.0.2.4
4	ns2.com.	3600	`A`	192.0.2.5
5	foo.com.	3600	`NS`	ns1.foo.com.
6	foo.com.	3600	`NS`	ns2.foo.com.
7	foo.com.	3600	`NS`	ns.bar.com.
8	ns1.foo.com.	3600	`A`	192.0.2.1
9	ns2.foo.com.	3600	`A`	192.0.2.2
10	ns.bar.com.	3600	`A`	192.0.2.6

Table 2.1: Example zone data for several fictitious zones.

2.1.1 Zone data

Zone data consists of *resource records* (RRs), each of which has (among other attributes) an owner name (e.g., *www.foo.com*), a time-to-live value (TTL) (e.g., 3600), a RR type (e.g., `A`), and record data specific to its type. For example, the record data for an RR of type `A` (address) is an Internet address (e.g., 192.0.2.3). Some RR types, such as the `CNAME` (canonical name) RR, use another domain name as record data—a *target* for further lookup (e.g., *baz.foo.com*). RRs of the same name and type comprise a *resource record set* (RRset). Table 2.1 shows zone data for fictitious zones.

One or more servers are designated as *authoritative* for a zone by including their domain names as the targets of RRs comprising a the `NS` (name server) RRset corre-

QUESTION			
www.foo.com.		`A`	

ANSWER			
www.foo.com.	3600	`CNAME`	baz.foo.com.
baz.foo.com.	3600	`A`	192.0.2.3

AUTHORITY			
foo.com.	3600	`NS`	ns1.foo.com.
foo.com.	3600	`NS`	ns2.foo.com.
foo.com.	3600	`NS`	ns.bar.com.

ADDITIONAL			
ns1.foo.com.	3600	`A`	192.0.2.1
ns2.foo.com.	3600	`A`	192.0.2.2

Figure 2.2: A DNS response for *www.foo.com* issued by a server authoritative for *foo.com*.

sponding to the zone origin. To properly handle delegation, these records must not only exist in the child zone, but also in its parent, as so-called *delegation records*. These RRs form the link between the parent and child, and proper function requires careful coordination between administrators of both zones. In Table 2.1 the delegation and authoritative `NS` RRs for *foo.com* are found on lines lines 5–7 of the *com* zone and lines 1–3 of the *foo.com* zone, respectively.

2.1.2 Name resolution

DNS name resolution typically involves three roles: a *stub resolver*, a *recursive resolver*, and an *authoritative server*. The resolver library of an operating system is a stub resolver. It is configured with one or more recursive resolvers to which it directs name lookups (e.g., in `/etc/resolv.conf` for UNIX systems). The recursive resolver performs lookups on behalf of requesting clients by querying authoritative servers. It may store answers in cache until their TTLs expire. Authoritative servers respond with answers to names corresponding to zones for which they are authoritative (e.g., *www.foo.com* for the *foo.com* zone). If an authoritative server receives a query for a domain name in namespace which has been delegated, then it responds with a referral to the servers authoritative for the child zone.

DNS messages are comprised of several sections, each built of RRs. The *question* section contains the RR describing the name being queried but does not include the corresponding record data. Answers to the question are returned in the *answer* section. Information about the servers authoritative for the zone of the name being queried are contained in the *authority* section. The *additional* section contains supplemental information that may be helpful or necessary for the resolver. The anatomy of a response that might be issued by a server authoritative for *foo.com* in response to a query for *www.foo.com* is shown in Fig. 2.2.

A resolver begins the name resolution process by issuing a query to one of the servers authoritative for the root zone. This root server responds by populating the authority section of the reply with the `NS` RRset for the delegated zone that is in the ancestry of the name in question. The resolver re-issues the query with the same question, this time directing it to one of the servers corresponding to the `NS` RRset provided in the authority section of the response. The resolver iteratively continues this process until it receives a response from an authoritative source containing either an answer or a response indicating that the requested RRset does not exist. A portion of the name resolution process for *baz.foo.com* is illustrated in Fig. 2.3, beginning with the query issued to server authoritative for the *com* zone.

2.1.3 `NS` target distribution

The simplest configuration for establishing and advertising authority for a zone is to use servers whose names are within the authoritative namespace of the zone. This typically implies that the zone administrators also maintain control of the authoritative servers. For example, the *ns1.foo.com* and *ns2.foo.com* servers from Table 2.1 are likely under the same administrative control as *foo.com.* If they alone were authoritative for *foo.com* then the administration of *foo.com* would be self-contained.

However, high availability of DNS name resolution requires network and geographic diversity of servers. This can be costly to acquire and maintain, especially for non-technical businesses or institutions that don't have the expertise or resources to accomplish this in-house. One solution for building robustness in an organization's

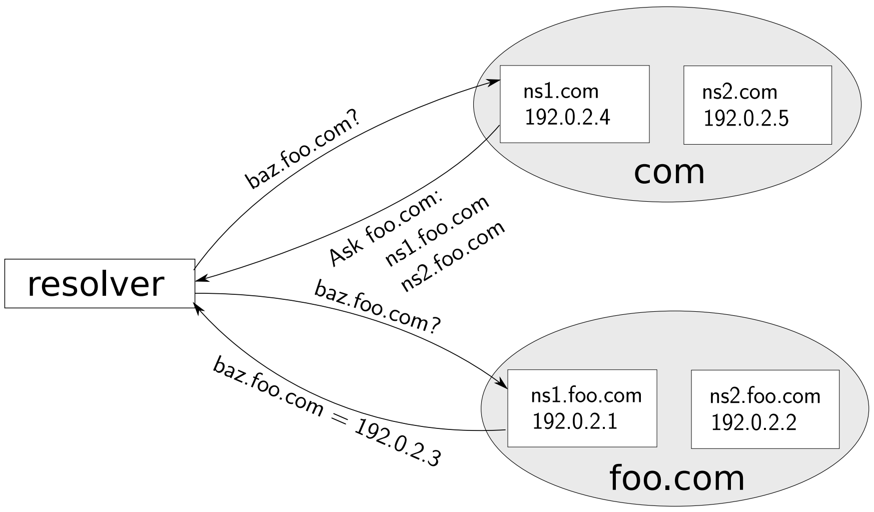

Figure 2.3: Part of the name resolution process for *baz.foo.com*. The resolver issues a query for *baz.foo.com* to *ns1.com*, which is authoritative for the *com* zone. The *ns1.com* server returns a referral to the servers authoritative for *foo.com*. The resolver re-issues the query for *baz.foo.com* to *ns1.foo.com* which returns an authoritative answer.

DNS infrastructure is to hire an external organization with the desired diversity to handle part or all of the DNS services. Alternatively an organization may add resilience at a small cost by partnering with other organizations with similar interests to host each others' zone data on their servers. In Table 2.1 the *ns.bar.com* server is authoritative for the *foo.com* zone. This sharing provides both load balancing and resilience in the case of network failure or other disaster, but it creates a dependency on an external entity.

2.1.4 Aliasing

The `CNAME` RR is defined to provide aliasing of one domain name to another. For example, *www.foo.com* is an alias for *baz.foo.com* in Table 2.1. If a resolver receives a response indicating that the name in question is an alias to another name, it must subsequently resolve the target of the alias, and so on until an address is returned. This functionality is mostly provided for ease of maintenance or for seamless transition of namespace.

It is suggested that `CNAME` RRs be used conservatively, that they not co-exist with any other RRs of the same name, and that they not be used as targets of `NS` RRs, `MX` (mail exchange) RRs, or other `CNAME` RRs [13]. However, these practices are not completely disallowed by existing implementations. The misuse of aliases results in added complexity, overhead, and potential vulnerability to name resolution.

2.2 DNS vulnerabilities

The fundamental nature of its functionality and its inherent insecurity have made the DNS the target of attack since its inception. Various vulnerabilities in protocol and implementation have facilitated exploits, which lead to compromise at higher levels [14, 15]. For example a malicious party might fabricate a response to a query for *www.foo.com* to redirect Web clients from its legitimate Internet address to a server set up to collect private information.

One of the largest targets in the DNS is the transport mechanism. The connectionless User Datagram Protocol (UDP) is used most often for DNS queries for several reasons. The size of queries and their responses is typically such that they can fit in a single datagram. The use of UDP also avoids the overhead required to establish a reliable Transmission Control Protocol (TCP) for a single query.

The issue that confronts the DNS is the effort required for a third-party to hijack DNS requests and inject the malicious response in a way which will be accepted by the requester. The connection-oriented TCP utilizes source and destination port, as well as sequence and acknowledgment number to maintain a connection between two parties. However, since UDP lacks the components utilized by TCP and only distinguishes using source and destination port, the problem space becomes much smaller for third-party injection of UDP packets. The destination port for DNS requests (and therefore the source port for DNS responses) is well-known (port 53), so the problem becomes guessing the unknown, 16-bit UDP port and an additional 16-bit identifier supplied in the header of the DNS packet itself.

Two prominent attacks which seek to narrow the problem space to correctly guess the bit sequence for the response that a client is expecting, are the birthday attack [16] and the Kaminsky attack [17]. In the birthday attack an attacker issues a large number of requests for the same name to a recursive resolver, resulting in an equal number of simultaneous requests open for that name on vulnerable resolver implementations. Each open request increases the chance for a successful spoof by an attacker, whose forged packet may match any of the open requests. Thus the number of attempts required by the attacker to achieve success is lessened significantly, according to the birthday paradox [18].

After a resolver receives a response from an authoritative server, the answer remains in its cache until it expires, which means that future queries to the resolver for the same name will not induce queries to the authoritative server. However, the Kaminsky attack skirts this limitation by making requests for non-existent names. For this approach there is no limit to queries that can be elicited by the resolver because non-existent names are in rich supply to the attacker. The objective is not to poison the RRs in the answer section, but rather the RRs in the authority and additional sections of the reply. The attacker, if successful, can redirect all queries for names in the compromised namespace to malicious servers, now recognized by the resolver as "authoritative" for the hijacked domains. The deficiency of sufficient UDP source-port randomization in resolver implementation reduces the problem space to guessing the 16 bits from the DNS query identifier through repeated queries.

Successful injection leads to *cache poisoning*, in which an illegitimate response is stored in a resolver's cache until expiration, which may be set arbitrarily long by the attacker. Until the response is expunged, the resolver will continue to use the false information.

2.2.1 Proposed solutions

Several solutions have been proposed for securing the DNS. With the DNS Security Extensions (DNSSEC) [5, 6, 7] answers may be cryptographically validated by resolvers. DNSSEC will be discussed in the next section.

DNSCurve [19] is another cryptographic solution for securing the DNS. With DNSCurve the communication channel between resolver and authoritative server is encrypted and authenticated using an elliptic-curve cryptographic system. The setup and maintenance for DNSCurve incurs very little overhead, and authoritative servers do not have any special requirements since the key is stored as the `NS` target of a delegation record for the domain. Authentication of DNS data with DNSCurve relies solely on the security of the channel between servers. Thus, a resolver may verify that it has received a DNS answer from an authoritative server that has not been modified in-transit. However, a third party, (e.g., a stub resolver) cannot in turn independently validate the response offline.

Several techniques have been proposed to specifically address the problem of lack of entropy, which facilitates third-party injection. Dagon, et al. [20] propose 0x20-bit encoding, which mixes the case of the owner name being queried. The authoritative server must respond with the same mixed case query name which it received from the resolver, which increases the problem space. WSEC DNS [21] introduces entropy by adding wildcard RRs into a zone, each of which aliases the legitimate record. Resolvers querying the authoritative servers for a name prepend a random string to the query name. The authoritative answer will respond with the correct answer because the random string will match the wildcard which aliases the real record. In addition to increasing entropy to impede would-be DNS hijackers, both of these solutions are quite backward compatible.

CoDoNS [22] is a peer-to-peer solution which proposes distributing the DNS namespace uniformly across a peer-to-peer network, as opposed to its current hierarchical structure. The hash of each domain name maps to a "home node" from which authoritative data can be queried. Name lookups are verified cryptographically using a certificate system. DoX [23] is a another peer-to-peer proposal, in which requests are sent to multiple, collaborating peers, and the answers returned are checked for consistency. Discrepancies raise a warning of cache poisoning.

2.3 Security extensions

The DNS Security Extensions (DNSSEC) [5, 6, 7] add authentication to the DNS. Public keys are included in the zone data for each zone using a `DNSKEY`-type RR with the same name as the zone. Each RRset in a zone is signed by the zone's private key, and each signature is included in the record data of a `RRSIG`-type RR, with the same name as the RRset it covers. `RRSIG` RRs also contain validity dates for the signature and the references to the `DNSKEY` RR needed to validate the signature. Any `RRSIG` RRs covering an RRset are included in the response to a DNSSEC query, so the resolver may perform validation using the appropriate `DNSKEY`.

A resolver may only authenticate an RRset with a `DNSKEY` RR that it has found to be authentic. The resolver is initially seeded with a *trust anchor*, which corresponds to a key that has signed the `DNSKEY` RRset in a zone (i.e., is *self-signing*). This trust anchor provides a *secure entry point* (SEP) into the zone. By verifying the self-signature of the trust anchor the resolver can authenticate other `DNSKEY` RRs in the `DNSKEY` RRset, which may then be used for validating DNSSEC data.

Rather than requiring a resolver to maintain a `DNSKEY` trust anchor for every signed zone, DNSSEC scales by establishing an authentication chain upwards through the namespace hierarchy, so resolvers may anchor with the `DNSKEY` of a common ancestor zone. The link between zones is accomplished by the introduction of `DS` (*delegation signer*) RRs in the parent zone. A `DS` RR maps to a `DNSKEY` in the child zone of the same name using a cryptographic digest of the `DNSKEY` stored as part of the record data for the `DS` RR. The `DS` RRset for a child zone is signed by the parent zone's key, so a resolver may validate the secure delegation between parent and child. A validating resolver can thus authenticate an RRset upwards along a *chain of trust* to a trust anchor. An *island of security* is a chain of trust comprised of one or more zones whose "top" zone is not securely linked to its parent.

A very common setup is for a zone to sign only its `DNSKEY` RRset with the SEP key (a *key signing key* or KSK) and sign zone data with a second key (a *zone signing key* or ZSK). This allows the ZSK to periodically be replaced without affecting parent zone or trust anchor dependencies associated with the SEP. The KSK maintains minimum exposure, having only a single `RRSIG`, so its replacement can be less frequent than the ZSK. In much of this dissertation, the implementation of a ZSK/KSK split is abstracted, except as necessary for discussion.

Figure 2.4 illustrates the chain of trust for an example DNS hierarchy. The *se-*

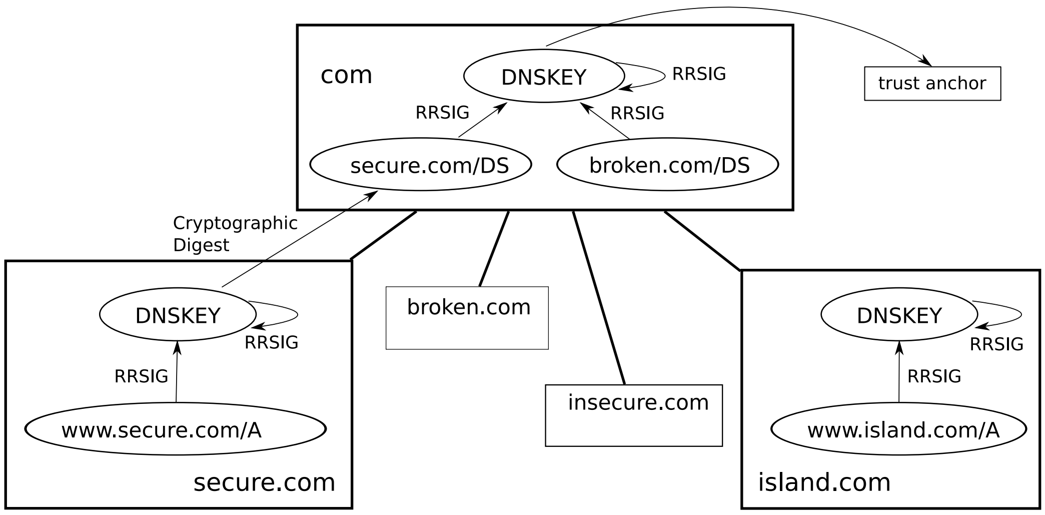

Figure 2.4: The DNSSEC authentication chains for several fictitious zones. `RRSIG`s are represented by upward arrows extending from the RRset they cover to the `DNSKEY` which can validate it. SEP `DNSKEY`s are mapped to their corresponding trust anchor or `DS` RR with an arrow. Self-signatures at each SEP `DNSKEY` are represented by a self-loop.

cure.com zone is linked to its parent, while *island.com* is an island of security. Neither *broken.com* nor *insecure.com* are signed.

The `NSEC` RR is used for *authenticated denial of existence.* When queried for an RRset that does not exist, an authoritative server sends the appropriate `NSEC` RRs to show where the RRset would be included if it did exist, along with their `RRSIG` RRs covering the `NSEC` RRs. This is important for a resolver to verify that a delegation is legitimately insecure, when there is a chain of trust to the signed parent zone. For example, a server authoritative for the *com* zone in Figure 2.4 should send the appropriate `NSEC` RR(s) in response to a querying resolver to prove that no `DS` RRs exist for the *insecure.com* and *island.com* zones.

The use of `NSEC3` for *hashed authenticated denial of existence* [24] was introduced to address the side effect of being able to traverse the `NSEC` chain of a zone to discover all RRsets comprising a zone. `NSEC3` provides the same functionality as `NSEC` RRs, except that the chain of owner names is replaced with a chain of cryptographic hashes of owner names, which impedes the potential to have third-parties discover all the zone data. The use of `NSEC3` is also more computationally demanding on both resolver and authoritative server. For the purposes of this dissertation, we use `NSEC` to represent authenticated denial of existence, unless noted otherwise.

2.3.1 Response validation

When a resolver is configured to validate DNS responses, there are three primary outcomes with regard to validation of an RRset using DNSSEC [7]. Our examples refer to Figure 2.4 and assume that a resolver is anchored with the *com* `DNSKEY`.

- *Secure*: An unbroken chain of trust can be established by the resolver between the RRset and a trust anchor. Example: an RRset in *secure.com* properly authenticated is deemed *secure.*

- *Insecure*: The resolver cannot verify authenticity of an RRset, but it has securely proven that there is no path wherein it might be validated. Example: if the `NSEC` RRs proving the non-existence of `DS` RRs for *insecure.com* and *island.com* are authenticated, proving the insecure delegation to those zones, then responses for those zones are *insecure.*

- *Bogus*: The resolver is unable to form a chain of trust between the RRset and a trust anchor and is unable to securely show that no such chain should exist. Example: an expired `RRSIG` covering an RRset in the *secure.com* yields a *bogus* response; likewise, any RRset in *broken.com* is *bogus* because there are no `DNSKEY`s, despite the existence of a `DS` RR.

In response to a query which a resolver deems bogus, the resolver returns a `SERVFAIL` status, which indicates general name resolution failure.

2.3.2 DNSSEC maintenance

A signed zone requires more careful maintenance than an unsigned zone. Since `RRSIG`s have a limited lifetime, a signed zone must be periodically re-signed to maintain validity. Also, periodic replacement of `DNSKEY`s is recommended, either because of compromise or prolonged exposure, which may make them the target of crypt-analysis attack. Such replacement of a `DNSKEY` is called a *key rollover*, and current best practices for rollovers are documented in RFC 4641 [25]. Non-SEP `DNSKEY`s (i.e., with no association with a trust anchor or a `DS` RR in the parent zone) can be rolled without involving third-parties and are thus self-contained. However, the rollover of a SEP `DNSKEY` requires involvement of the parent zone to handle the change in *DS* RRs, and validating resolvers must be engaged when their trust anchors are rolled.

It is common and good practice for multiple servers to act as authoritative for a zone for purposes of high availability. However, zone data must be consistent across all authoritative servers. This is especially true for a signed zone which contains time-sensitive `RRSIG`s. Servers must also be consistent in the level of DNSSEC functionality they support. For example, if a server lacks DNSSEC support altogether, then its responses will not contain the RRs required for DNSSEC, such as `RRSIG` or `NSEC`. If a zone is signed with `NSEC3`, authoritative servers must know how to return `NSEC3` RRs.

Chapter 3

DNS Dependency Model

While the concept of name resolution is relatively simple, the overall DNS is complex and its effects far-reaching. Name resolution for a domain is often dependent on servers well outside the control of the domain's owner and managed by third parties. A network of inter-organizational relationships overlays the DNS infrastructure, and configurations that create a dependency on peer organizations are in turn affected by the security and accuracy of namespaces linked through this network. An understanding of a domain's context in the entire system will allow architects and administrators to better design and configure their DNS services to maximize the reliability of DNS name resolution.

In this chapter we present a quantitative analysis of name dependencies in the DNS. We review relevant aspects of the DNS protocol, as defined by Internet standards. We also examine several DNS server implementations and compare their behaviors to the specifications. Using these analyses, we derive a model to probabilistically determine the namespace influencing resolution of a domain name and test it against real domain names. Based on our model, we define several metrics to assess the quality of name resolution for a domain name, based on the other names that affect its resolution. The behaviors of name server implementations, some of which are configurable, can affect these metrics. We analyze a large sample of recent DNS data in light of the presented model. The results show the impact of name dependencies in terms of the size of the namespace that influences a domain name, and in the percentage of queries that will reach namespace not explicitly configured by DNS administrators.

The following are the primary contributions of this chapter:

- A detailed study of DNS protocol specification and name server implementation, with respect to DNS name dependencies.
- A formal model for analysis of DNS name dependencies, based on specification and implementations.
- Metrics for quantifying the influence domain names have on other domain names.

Our analysis shows that 92% of influential namespace of domain names is explicitly configured by domain administrators. However, certain caching behaviors reduce that figure, and increase the probability that resolution of a domain name is influenced by names or organizations not explicitly configured by administrators. A diverse set of dependencies has the potential to affect the reliability of name resolution for a domain name. Based on our findings, we discuss best practices for design and administration of DNS services to contain influence of domain names.

We describe previous work in this area in Section 3.1. In Section 3.2 we introduce the concept of DNS name dependencies and examine technical details of name resolution. In Section 3.3 we formalize a graph model for analyzing DNS name dependencies and derive methods for quantifying influence. We describe methodologies employed for data collection, an evaluation of the graph model using real DNS data, and an analysis of the observed quality of name resolution in Section 3.4. We summarize our analysis in Section 3.5.

3.1 Previous work

Ramasubramanian, et al. [2] demonstrate the far-reaching effects of DNS dependencies by surveying the DNS namespace and tracing the dependencies of a large number of domain names. They identify the set of all name servers potentially involved in resolution of each name, which comprise the potential attack target for that name. Their results show that a domain name relies on 44 name servers on average and 6% of names depend on more than 200 servers. We perform further examination of name resolution behaviors to create a formal model of name dependencies in the DNS and quantify the significance of such dependencies.

3.2 Name dependencies in DNS

DNS specification and the behavior of DNS server implementations result in various dependencies for domain name resolution. Prior to establishing a formal dependency model we first define domain name dependency, after which we discuss DNS behaviors resulting in name dependencies.

Domain name u *depends on* domain name v if resolution of v may *influence* resolution of u. This dependence is transitive: if u depends on v and v depends on w, then u depends on w. We define the *trusted computing base* (TCB) for a domain name as the set of all domain names influencing it.

The raw size of the TCB for a domain name may be insufficient for analyzing the domain names influencing it. In some cases policy or preference may dictate whether or not it is acceptable for a domain name to influence another to any degree. For example, a government zone may prohibit zones operated by foreign governments from its TCB. However, a thorough analysis will show that not all names have equal influence. In this research we introduce *level of influence* $I_u(v)$ as a quantitative measure of v's influence on u. Level of influence is formally defined in Section 3.3.

Influence is categorized into two classes: *active* and *passive*. If domain name u is actively influenced by domain name v, then with some non-zero probability resolution of v will be *required for* resolution of name u. If domain name u is passively influenced by domain name v, then although v may not be required for resolution of u, resolution of v may *affect* resolution of u with some probability. The conditions for active and passive influence are described later in this section.

Three specific components in the DNS protocol lead to such domain name dependencies:

- *Parent zones*: Because name resolution is performed by following downward referrals in the name hierarchy, a name is always dependent on its parent zone.

- *`NS` targets*: The `NS`-type RR type uses names, rather than addresses, for specifying servers authoritative for a zone, so a resolver must resolve the names before it can query the authoritative servers.

- *Aliases*: If a name resolves to an alias, then to obtain an address, a resolver must subsequently resolve the alias target.

Some discussion of specific aspects of DNS behavior is required to properly create a well-formed dependency model. The role of glue and additional records in delegation, the selection of authoritative name servers, the trust ranking of data, and TTL dynamics are discussed in the remainder of this section. Table 3.1 is provided as a reference for this discussion, and Table 3.2 lists the notation referenced throughout this chapter. The behaviors of two popular DNS server implementations are also referenced: the Berkeley Internet Name Daemon version 9.5 (BIND) [26] and djbdns [27].

$ORIGIN soccer.com. (SoccerMania, Inc.)			
	Name	Type	Value
1	soccer.com.	`NS`	ball.soccer.com.
2	soccer.com.	`NS`	racket.tennis.com.
3	soccer.com.	`NS`	ns1.sports.net.
4	ball.soccer.com.	`A`	192.0.2.1
5	www.soccer.com.	`CNAME`	www.tennis.com.

$ORIGIN tennis.com. (Tennis Pro, Inc.)			
	Name	Type	Value
1	tennis.com.	`NS`	ns1.tennis.com.
2	tennis.com.	`NS`	ball.soccer.com.
3	tennis.com.	`NS`	ns1.sports.net.
4	ns1.tennis.com.	`A`	192.0.2.2
5	www.tennis.com.	`A`	192.0.2.3
6	racket.tennis.com.	`A`	192.0.2.4

$ORIGIN athletics.com. (Sports Central, Inc.)			
	Name	Type	Value
1	athletics.com.	`NS`	ns1.athletics.com.
2	ns1.athletics.com.	`A`	192.0.2.8

$ORIGIN com. (VeriSign, Inc.)

	Name	Type	Value
1	com.	`NS`	ns1.com.
2	ns1.com.	`A`	192.0.2.5
3	athletics.com.	`NS`	ns1.athletics.com.
4	soccer.com.	`NS`	ball.soccer.com.
5	soccer.com.	`NS`	racket.tennis.com.
6	soccer.com.	`NS`	ns1.sports.net.
7	tennis.com.	`NS`	ball.soccer.com.
8	tennis.com.	`NS`	ns1.tennis.com.
9	tennis.com.	`NS`	ns1.sports.net.
10	ball.soccer.com.	`A`	192.0.2.1
11	ns1.tennis.com.	`A`	192.0.2.2
12	ns1.athletics.com.	`A`	192.0.2.8

$ORIGIN sports.net. (Sports Central, Inc.)

	Name	Type	Value
1	sports.net.	`NS`	ns1.sports.net.
2	sports.net.	`NS`	ns1.athletics.com.
3	ns1.sports.net.	`A`	192.0.2.6

$ORIGIN net. (VeriSign, Inc.)

	Name	Type	Value
1	net.	`NS`	ns1.net.
2	ns1.net.	`A`	192.0.2.7
3	sports.net.	`NS`	ns1.sports.net.
4	sports.net.	`NS`	ns1.athletics.com.
5	ns1.sports.net.	`A`	192.0.2.6

Table 3.1: Example zone data for several fictitious zones.

3.2.1 Glue and additional records

When a query for a name in zone z reaches name server s, which is authoritative for $Parent(z)$, s responds with the `NS` RRset corresponding to the name servers authoritative for z as a referral. Let NS_z denote the `NS` target names from the RRset. Addresses of the `NS` targets in NS_z are required for the resolver to subsequently query the servers. If any `NS` targets are subdomains of z, then s must also include *glue records* for those targets in the additional section of the response to "bootstrap" the resolution process, so there isn't a cyclic dependency between a zone and its descendants [11]. The glue records are `A` RRs corresponding to the target names of the `NS` RRs for z but maintained in the $Parent(z)$ zone. The `NS` RRs and associated glue records for *tennis.com* are found on lines 7–11 of the *com* zone in Table 3.1.

If server s has pertinent non-glue `A` RRs available locally, it may send them in the additional section of its response to expedite the resolution process for the resolver. This could happen if s is also authoritative for the zones to which the targets belong or if s has an answer cached from an authoritative response [11]. However, any such RRs included in the response for which $Parent(z)$ is not a superdomain are considered *out-of-bailiwick* (i.e., outside its jurisdiction). Thus resolver implementations should independently obtain an authoritative answer for the out-of-bailiwick target names before querying such servers.

The resolver is responsible for resolving any names from NS_z which are out-of-bailiwick or not included in the additional section of a response from s. Such induced queries indicate active influence of the resolved names on z, since it is directly dependent on their resolution.

3.2.2 Name server selection

RFC 1035[12] describes the process by which servers are selected by a resolver for querying a zone z as part of the resolution process. The resolver begins with the list of all server names NS_z. The addresses known by the resolver for target names in

NS_z initially populate the set of corresponding addresses, and it initiates requests in parallel to acquire addresses for any others. The resolver also associates historical statistics, such as response time and success rate, to each address. The complete set of addresses corresponding to `NS` target names in NS_z is denoted NSA_z. A resolver will avoid using an address from NSA_z twice until all addresses have been tried at least once. After that, it prefers the server with the best performance record, thus fine-tuning the performance for lookups of z [12].

This behavior is not consistent across implementations. The djbdns name server selects a server from NSA_z uniformly at random. However, a resolver using BIND, which follows the performance-based selection guideline, will gravitate toward preferring a single server or set of servers in NSA_z. We make the assumption that requests for subdomains of z arrive from resolvers in diverse network and geographic locations, such that the preference to servers in NSA_z is distributed uniformly among such resolvers. This leads to an equal probability that any server in NSA_z receives a query for subdomains of z.

3.2.3 Trust ranking

RFC 2181[28] outlines a relative ranking of trustworthiness of data for name servers to consider as part of operation. Among the total ranking are the following (in decreasing order of trustworthiness):

- Data from a zone for which the server is authoritative, other than glue data
- The authoritative data included in the *answer* section of an authoritative reply
- The data in the *authority* section of an authoritative reply
- Glue from a zone for which the server is authoritative
- Data from *additional* section of a response

This trust ranking has effects on name dependencies with regard to both the resolver and the authoritative server. The authoritative set of `NS` target names for z,

NS_z, may differ from those configured in $Parent(z)$, denoted NS'_z. While a resolver must initially use the set NS'_z provided by a server authoritative for $Parent(z)$, once it receives an answer for a name in z from a server authoritative for z, it will use any target names in NS_z (provided in the authority section) in preference to those in NS'_z. This behavior is consistent with both BIND and djbdns. Server selection therefore depends not only on the `NS` targets in NS_z but also on the probability that the set of `NS` RRs for z has been cached by the resolver—either from the answer or authority section of an authoritative reply. This probability is denoted $P_{NS}(z)$.

If authoritative server $s \in NSA_{Parent(z)}$ has caching functionality enabled and has stored the `A` RR for an `NS` target $v \in NS_z$ from the answer section of an authoritative response, according to the RFC, it will trust this RR more than a glue in its own configuration. Let $P_s(v)$ denote the probability that s has in cache and provides such authoritative data for v. This behavior is configurable in BIND, but it is enabled by default.

If resolver c has cached the address for $v \in NS_z$, as the result of an answer from an authoritative source from a prior transaction, then c deems the cached data more trustworthy than any data received in the additional section of a response from an in-bailiwick authority. Thus, it will use the previously cached data in preference to data—whether from glue or s's cache—returned in the additional section by $s \in NSA_{Parent(z)}$. $P_c(v)$ denotes the probability that c has and uses such authoritative data for v in its cache. BIND adheres strictly to this, as it will direct queries to an address received by a more "trustworthy" source over a server returned in an additional section—unless the authoritative data is an alias (i.e., a `CNAME` RR). The djbdns name server treats the `A` RRs with equal precedence, but will always use an authoritative `CNAME` RR over an additional `A` RR of the same name.

Suppose $v \in NS_z$ is a subdomain of $Parent(z)$, $Parent(v) \neq z$, and $Parent(z)$ is properly configured with a glue record for v. If an authoritative answer for v has previously been resolved and cached by either $s \in NSA_{Parent(z)}$ or resolver c, then z is affected by v and its name dependencies. This behavior describes passive influence of v on z. The probability of passive influence, $P_{\{s,c\}}(v)$, is the combined probability of $P_s(v)$ and $P_c(v)$, the likelihood that either s or c has and uses a cached authoritative answer for v. Since the probabilities are independent of one another, $P_{\{s,c\}}(v)$ is calculated:

$$P_{\{s,c\}}(v) = P_s(v) \vee P_c(v) = 1 - (1 - P_s(v))(1 - P_c(v))$$

3.2.4 TTL dynamics

The TTL value of related RRs impacts influence. For example, suppose the TTL of the `NS` RRs for zone z is 3600 and the glue record for in-bailiwick `NS` target $v \in NS_z$ has a TTL of 1800, and suppose that a resolver caches both the `NS` RRs and the `A` RRs (from glue) at time $t = 0$. For lookups in z during $0 < t \leq 1800$ the resolver will have the addresses necessary to query authoritative server v. However, when $1800 < t \leq 3600$ the `NS` RRs for z are still in the resolver's cache, but now a lookup is required to query v, so z is now required by v, even though it is in-bailiwick and a glue exists. Not until $t > 3600$, when the `NS` RRs for z expire and it is required to re-query the $Parent(z)$ servers, does the resolver again receive glue records for v in the additional section of a response. However, as explained previously, if the glue records co-exist in cache with records from authoritative sources, the addresses from the glue records are ignored when selecting servers to query, in which case z is still subject to passive influence by v.

We note that after initially caching the set of `NS` RRs and any address records from the from the authority and additional sections, respectively, of an authoritative response, the BIND resolver does not replace these records with those from subsequent authoritative responses until they expire from the cache. The djbdns resolver, however, does update existing records. Thus if a djbdns resolver performs repeated lookups for domain names in z at a rate which prolongs expiration of cached `NS` and address records, passive influence is minimized. Conversely, depending on rate of lookup and TTL values for `NS` and address records for z and its authoritative servers, a BIND resolver may be more prone to experience passive influence for z.

While certain configurations may affect the values for $P_{\{s,c\}}(v)$, in this article we focus on the dependency behaviors at $t = 0$, and leave $P_{\{s,c\}}(v)$ as a variable.

Term	Definition
r	The root name "."
$I_u(v)$	The measure of name v's influence on name u
$I_u(D)$	The aggregate influence of names in set D on name u
$Parent(d)$	The nearest ancestor zone of name d
$Cname(d)$	The alias target of name d
NS_z, NS'_z	The set of `NS` target names authoritative for zone z, as configured in z and $Parent(z)$, respectively
NSA_z, NSA'_z	The set of addresses corresponding to the names in NS_z and NS'_z, respectively
NSA^y_z	The set of servers authoritative for zone z but not for zone y
$P_{NS}(z)$	The probability that the resolver has the set of `NS` RRs for z cached from an authoritative source
$P_{\{s,c\}}(v)$	The probability that either s or c has in cache and uses `NS` target name v from an authoritative source
$G_d = (V_d, A_d)$	Name dependency graph for name d
$G'_d = (V'_d, A'_d)$	Active influence dependency graph for name d
$P_q(z, v)$	The probability that `NS` target v is used to resolve z
$w(u, v)$	The weight of edge (u, v) in A_d
S_u	The set of addresses corresponding to name u
$U'_d \subseteq U_d \subseteq Z_d$	The sets of first-order, non-trivial, and all zones in V_d, respectively

Table 3.2: Notation used in this chapter.

3.3 DNS dependency model

Name dependencies are quantified using level of influence, which is the probability that one domain name will be utilized for resolving another. Let $I_u(v) \in [0, 1]$ denote v's level of influence on u—i.e., the probability that domain v will be used in the resolution process for u. Dependencies may be reciprocated (i.e., $I_u(v) > 0$ and $I_v(u) > 0$), though the level of influence in each direction may differ. In the remainder of this section, a model is defined for analysis and quantification of DNS name dependencies.

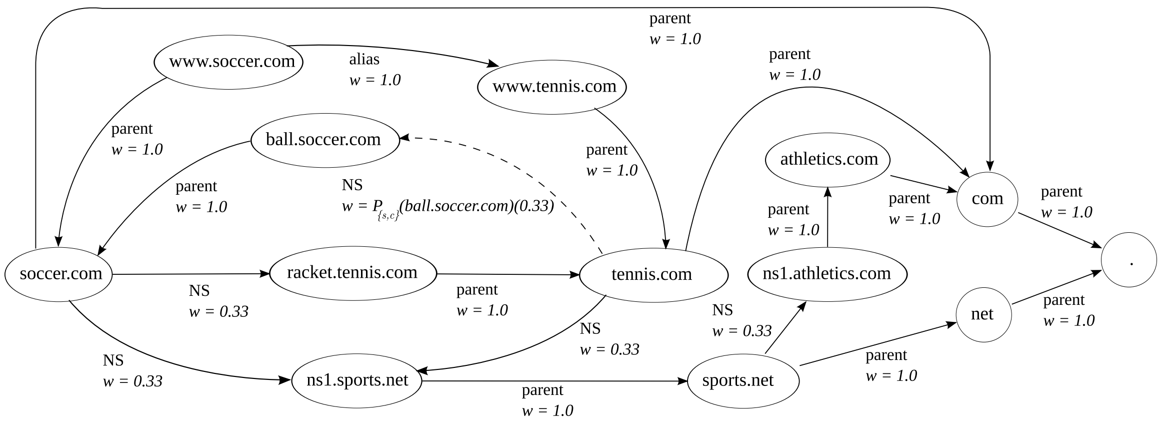

Figure 3.1: The dependency graph for *www.soccer.com*, derived from the zone data in Table 3.1. The solid lines represent active influence, and the dashed lines represent passive influence.

3.3.1 Name dependency graph

To derive the values for influence of domain name d a directed, connected graph, $G_d = (V_d, A_d)$, is used to model name dependencies. The graph G_d contains a single sink, r, which is the root zone. Each node in the graph $v \in V_d$ represents a domain name, and each edge, $(u, v) \in A_d$, signifies that u is directly dependent on v for proper resolution of itself and any descendant names. Each edge, $(u, v) \in A_d$, carries a weight, $w(u, v)$, indicative of the probability that it will be followed for resolving u. A name dependency graph for domain name *www.soccer.com* is shown in Fig. 3.1, built from the data in Table 3.1.

Edges are placed on the graph from each domain name $u, u \neq r$ to its parent $Parent(u)$ with $w(u, Parent(u)) = 1$; a domain name is always dependent on its parent. If resolution of domain name u yields a `CNAME` RR, then an edge is placed between u and its target name, $Cname(u)$, with $w(u, Cname(u)) = 1$; the resolution of an alias is always dependent on the resolution of its target. Such edges in Fig. 3.1 are those between *www.soccer.com* and its parent, *soccer.com*, and between *www.soccer.com* and its canonical name, *www.tennis.com*.

Placement of edges and weights corresponding to `NS` target dependencies is somewhat involved and draws from the discussion in Section 3.2. The considerations are

v subdomain of $Parent(z)$	Glue exists	$Parent(v) = z$	$w(z, v)$	Example (Table 3.1 and Fig. 3.1)
no			$P_q(z, v)$	*soccer.com* → *ns1.sports.net*
yes	no		$P_q(z, v)$	*soccer.com* → *racket.tennis.com*
yes	yes	no	$P_{\{s,c\}}(v)P_q(z, v)$	*tennis.com* → *ball.soccer.com*
yes	yes	yes	0	*soccer.com* → *ball.soccer.com*

Table 3.3: Rules for determining whether or not and with what weight $w(z, v)$ a edge is placed between a zone z and an `NS` target $v \in NSA_z$.

summarized in Table 3.3.

We first identify the proportion of queries distributed among each of the `NS` target names in NS_z, which we use as a base for calculating the weights of edges in A_d stemming from `NS` target dependencies. Since resolvers select from *addresses* rather than *names* of authoritative servers, the probability, $P_q(z, v)$, of querying any `NS` target $v \in NS_z$ for resolution of z will be some fraction of $|NSA_z|$ that reflects the proportion of server addresses attributed to v. Let S_v represent the set of addresses to which $v \in NS_z$ resolves. A naïve formula for determining query probability $P_q(z, v)$ is to simply calculate the fraction of total server addresses authoritative for z that correspond to v:

$$P_q(z, v) = \frac{|S_v|}{|NSA_z|}$$

The zone data for *foo.com* in Table 3.4 shows that an `NS` target name that resolves to multiple addresses, such as *ns1.foo.com*, has a higher probability of being queried for names in the zone than an `NS` target name that resolves to only a single address, such as *ns2.foo.com*.

It is possible that multiple `NS` target names in NS_z resolve to the same address, so a single address in S_v may also be attributed to other names in NS_z. A more complete approach to determining query probability therefore is to evenly divide the probabilistic weight attributed to a server address among all the names that resolve

Name	Type	Value	$P_q(z, v)$
foo.com.	`NS`	ns1.foo.com.	$\frac{2}{3} = 0.67$
foo.com.	`NS`	ns2.foo.com.	$\frac{1}{3} = 0.33$
ns1.foo.com.	`A`	192.0.2.9	
ns1.foo.com.	`A`	192.0.2.11	
ns2.foo.com.	`A`	192.0.2.10	
bar.com.	`NS`	ns1.bar.com.	$\frac{1+0.5}{2} = 0.75$
bar.com.	`NS`	ns2.bar.com.	$\frac{0.5}{2} = 0.25$
ns1.bar.com.	`A`	192.0.2.12	
ns1.bar.com.	`A`	192.0.2.13	
ns2.bar.com.	`A`	192.0.2.12	

Table 3.4: Example zone data to illustrate query distribution among `NS` target names of servers authoritative for a zone.

to that address:

$$P_q(z, v) = \frac{\sum_{s \in S_v} |\{u \in NS_z | s \in S_u\}|^{-1}}{|NSA_z|}$$

For example, in Table 3.4 both *ns1.bar.com* and *ns2.bar.com* resolve to 192.0.2.12, so the weight of that server is split evenly among both names. The result is that *ns1.bar.com* is queried with 0.75 probability for *bar.com* because it also resolves to 192.0.2.13, and *ns2.bar.com* is queried with only 0.25 probability.

When $NS_z \neq NS'_z$, the query probability of an edge to `NS` target v must also factor in the probability, $P_{NS}(z)$, that the resolver has cached the authoritative set of `NS` RRs for z, as well as v's membership in NS_z and NS'_z:

$$P_q(z, v) = P_{NS}(z) P(v \in NS_z) \frac{\sum_{s \in S_v} |\{u \in NS_z | s \in S_u\}|^{-1}}{|NSA_z|} + \\ (1 - P_{NS}(z)) P(v \in NS'_z) \frac{\sum_{s \in S_v} |\{u \in NS'_z | s \in S_u\}|^{-1}}{|NSA'_z|}$$

For simplicity we assume that $NS_z = NS'_z$ unless specified otherwise.

If `NS` target $v \in NS_z$ is not a subdomain of $Parent(z)$, edge (z, v) is added to G_d with $w(z, v) = P_q(z, v)$. Resolution of v is required for (i.e., actively influences) resolution of z. An example is *soccer.com*'s dependency on *ns1.sports.net*.

If target name $v \in NS_z$ is a subdomain of z, the $Parent(z)$ zone should include a glue record for v. If no glue record exists for v in the $Parent(z)$ zone, then resolution of v is required for (i.e., actively influences) resolution of z, and an edge (z, v) is added to G_d with $w(z, v) = P_q(z, v)$. Such is the case with *soccer.com*'s dependency on *racket.tennis.com*.

If v is in-bailiwick, and a glue record for v exists, then resolution of v is not required for resolving z because the resolver will use the address provided in glue from the $Parent(z)$ authoritative server. When $Parent(v) = z$, there is no edge (z, v) in G_d; all servers authoritative for z have the authoritative data for v, such as with *ball.soccer.com*'s relationship to *soccer.com*. However, when $Parent(v) \neq z$ an edge (z, v) is added with $w(z, v) = P_{\{s,c\}}(v)P_q(z, v)$; the name v passively influences z, dependent on the probability that either the resolver or the authoritative server has the address for v cached from an authoritative source. An example is *tennis.com*'s dependency on *ball.soccer.com*.

The active influence dependency graph, G'_d, of domain name d is the subgraph of G_d produced when $P_{\{s,c\}}(v) = 0, \forall v \in V_d$ and nodes with only zero-weight in-edges are removed from the graph. The active influence dependency graph for *www.soccer.com* would be created by eliminating the *ball.soccer.com* node in Fig. 3.1.

3.3.2 Level of influence

An analysis of the *dependency paths* in G_d is necessary to determine the level of influence of the domain names $v \in V_d$ on d. The dependency paths in G_d are modeled by performing a depth-first traversal of G_d, beginning with d. This depth-first traversal produces the exhaustive set of acyclic intermediate paths of name dependencies for resolving d. The level of influence is calculated by determining the probability that paths leading from d will reach v during resolution:

$$I_d(v) = P(d, \ldots, v)$$

To calculate $P(d, \ldots, v)$, the probabilities of encountering v in the dependency paths beginning with each of u's direct dependencies must first be recursively calculated and aggregated. The probability of encountering v in a path beginning with edge $(u, j) \in A_d$ is calculated by multiplying the probability, $w(u, j)$, of following edge (u, j) by the probability of encountering v in a path beginning with j:

$$P(u, j, \ldots, v) = \begin{cases} w(u, j) & \text{if } j = v \text{ (direct dep)} \\ 0 & \text{if } j = r \text{ (root)} \\ w(u, j)P(j, \ldots, v) & \text{otherwise} \end{cases}$$

For a given domain name $u \in V_d$, resolution of u often requires following multiple branches at an intermediate node, depending on the relationship between the dependency types. For `NS` target dependencies of u at most one address from NSA_u is followed (assuming no server failure). However, alias and parent dependencies exist independently of the `NS` target dependencies. For example, when resolving names in *tennis.com* using the zone data from Table 3.1, either *ns1.tennis.com*, *ball.soccer.com*, or *ns1.sports.net* will be selected, each with equal probability. However, its resolution remains entirely dependent on its parent, *com*, regardless of which server in $NSA_{tennis.com}$ is selected for query.

Aggregating the probability of encountering v in paths beginning with each of u's direct dependencies is as follows. First the probability of encountering v through any `NS`-type dependencies is determined by calculating the sum of encountering it in each of the `NS`-type dependency edges because the probabilities are dependent on one another:

$$P(u, [NS\ dep], \ldots, v) = \sum_{j \in NS_u} P(u, j, \ldots, v)$$

This probability is then combined independently with the probability of encountering v in paths beginning with any alias- or parent-type dependencies:

$$\begin{aligned} P(u, \ldots, v) = \ & 1 - \Big(1 - P(u, Parent(u), \ldots, v)\Big) \\ & \Big(1 - P(u, Cname(u), \ldots, v)\Big) \\ & \Big(1 - P(u, [NS\ dep], \ldots, v)\Big) \end{aligned}$$

Using these expressions, we calculate the level to which *sports.net* influences *soc-*

cer.com:

$$\begin{aligned}
&I_{www.soccer.com}(sports.net) = \\
&\qquad 1 - (1 - P(www.soccer.com, soccer.com, \ldots, sports.net)) \\
&\qquad (1 - P(www.soccer.com, www.tennis.com, \ldots, sports.net)) \\
&\quad \ldots \\
&\quad = 0.62 + 0.06 P_{\{s,c\}}(ball.soccer.com)
\end{aligned}$$

3.3.3 Graph properties

Finding the level of influence of a single name on d requires following all paths between d and r, which is computationally complex. However, often it may suffice to simply know the set of names influencing d, or other representative properties of G_d. This section describes some properties from which metrics can be derived for quantifying the TCB of d and measuring the extent to which its resolution is affected by third parties.

Influential zones

The set of influential zones $Z_d \subseteq V_d$ is a measure of the TCB of d. Although a single organization may maintain several zones in Z_d, it is generally representative of the diversity of organizations that influence resolution of d. In Fig. 3.1 $Z_{www.soccer.com} = \{soccer.com, tennis.com, sports.net, athletics.com, com, net, .\}$.

Non-trivial zones

Non-trivial zones are the result of explicitly configured inter-zone dependencies. Included in this set are the parent zones of any `NS` or alias targets in A_d: $U \subseteq Z_d$. A non-trivial zone *foo.bar.com* that influences d may contribute up to four zones to Z_d. However, if no in-edges resulting from alias- or `NS`-type dependencies exist for any of its ancestor zones (*bar.com*, *com*, and "."), then they exist in Z_d only because *foo.bar.com* is explicitly configured as a dependent zone and are thus trivial. Algorithm 1 identifies non-trivial zones by iterating the set of edges A_d and

Algorithm 1 `NonTrivialZones`(d)

```
Input: Domain name $d$
Output: Set of non-trivial zones in $V_d$
1: $D \leftarrow \{Parent(d)\}$
2: for all $(u, v) \in A_d$ do
3:     if $(u, v)$ is an NS target or alias dependency then
4:         $D \leftarrow D \bigcup \{Parent(v)\}$
5:     end if
6: end for
7: return $D$
```

adding the parent zones of `NS` and alias targets. In Fig. 3.1 $U_{www.soccer.com} = \{soccer.com, tennis.com, sports.net, athletics.com\}$.

First-order dependencies

A subset of non-trivial zones $U'_d \subseteq U_d$ are explicitly configured by d (or $Parent(d)$, if d is not a zone) and comprise *first-order* dependencies. U'_d also includes the non-trivial zones in the ancestry of each explicitly configured zone. Algorithm 2 finds all the alias (lines 5–7) and `NS` target (line 11) dependencies for a name d and then includes the parent zone for each target (line 15) and each non-trivial zone in its ancestry (lines 16–21). In Fig. 3.1 $U'_{www.soccer.com} = \{soccer.com, tennis.com, sports.net\}$.

Third-party influence

The computational complexity of calculating level of influence for all $u \in V_d$ renders it infeasible in a large dependency graph. A less computationally demanding metric is determining how much domain d is influenced by names outside of U'_d, i.e., $I_d(U_d - U'_d)$. We call this *third-party influence* (TPI). To do this, two helper algorithms are utilized: the `ControlledAlias` algorithm (Algorithm 3) analyzes a name to determine whether or not it aliases (directly or indirectly) another name outside of the set of U'_d. The `ThirdPartyInfluence1` algorithm (Algorithm 4) determines the probability that resolution of u will utilize a name outside the set of U'_d. The latter is computed by aggregating the probabilities that u will utilize a name outside of U'_d from aliasing (lines 3–5) or from `NS` target dependencies in its ancestry (lines 10–19).

Algorithm 5 describes the methodology for calculating the TPI of d. The TPI of d's alias, if any (line 6), is combined (line 18) with the TPI of its parent zones (line 11) and that of its collective `NS` target dependencies (lines 14–16).

3.4 Data collection and analysis

In this section we describe the methodology we employed for collecting data from the DNS infrastructure, and provide analysis of the data collected. With a subset of the DNS data we evaluate how well theoretical influence correlates with empirical analysis. Using results from the entire data set we analyze several different areas to assess quality of name resolution.

3.4.1 Data collection

We populated a database of name dependencies by crawling the namespace of known domain names. A set of over 3,000,000 hostnames was extracted from URLs indexed as part of the Open Directory Project (ODP) at DMOZ [29] dated April, 2009. These names were combined with over 100,000 names received as queries by the recursive servers at the International Conference for High-performance Computing, Networking, Storage and Analysis (SC08) [30]. The ODP/SC08 names were used to seed the domain name database.

Each name was investigated by first surveying each name in its ancestry which had not already been surveyed, beginning with the root. Surveying a domain name consisted of issuing queries to a recursive server to receive an authoritative answer for any matching `NS`, `MX` (mail exchange) and `CNAME` RRs. The relationships between the name and any corresponding targets returned were recorded and subsequently surveyed.

For each `NS` RR, we checked the consistency between parent and child zones by using some extra probing. For zone z we found the set of servers only authoritative for $Parent(z)$, $NSA^z_{Parent(z)} = NSA_{Parent(z)} - NSA_z$. For each server in $NSA^z_{Parent(z)}$ we issued a query for z until a response was received that did not have the authoritative answer (AA) flag set. Only if the AA flag was not set could we accurately obtain the set of `NS` target names (NS'_z) maintained by $Parent(z)$. If $NS_z \neq NS'_z$ an inconsistency is detected.

The TTL field of additional address records corresponding to targets of `NS` RRs in the authority section of server responses are used to identify the presence of glue records in the parent zone. When server s returns a non-authoritative response, a second query is issued to s after a two-second delay (both without the recursion-desired flag set). Since TTL is measured in seconds, the two-second delay between queries will result in a decreasing TTL for additional records sent from s's cache. If for an `NS` target there is no corresponding address record in the additional section, then it is indicative that the parent has not been configured with a glue record. If the TTL of the additional record differs between the two responses, then it is inferred that the record came from an authoritative response in s's cache. Since such a response would take precedence over any glue record, we optimistically give the zone the benefit of the doubt that it is configured with a glue record, if the `NS` target is in-bailiwick.

If the TTL value of an additional record does not vary between the two responses from s, it could indicate one of several things:

- the parent zone is configured with a glue record for the additional record;
- s is (also) authoritative for the zone to which the additional record belongs; or
- s is authoritative for an ancestor of the `NS` target and has been configured with a glue record for that `NS` target.

We assume optimistically in this case that if the `NS` target is in-bailiwick, a glue record is present.

If no non-authoritative answers are returned from querying the servers in the set $NSA^z_{Parent(z)}$, then we cannot determine inconsistencies between NS'_z and NS_z, and their corresponding glue records. However, in practice, if $NSA_{Parent(z)} \subseteq NSA_z$, then

Measurement	Values
ODP/SC08 hostnames	3,167,594
Total domain names collected	8,439,927
Total zones	2,996,460
`NS` target dependencies	6,855,379
`NS` targets requiring glue	3,723,203 (54%)
`NS` targets missing required glue	901 (0.024%)
Zones for which $NS_z \neq NS'_z$	587,865 (20%)

Table 3.5: A summary of results collected from surveying the DNS namespace, seeded with ODP/SC08 hostnames.

consistency is satisfied implicitly since all servers in $NSA_{Parent(z)}$ will send authoritative records from z over corresponding records from $Parent(z)$ [28]. For all zones in our analysis we let $P_{NS}(z) = 0.5$, so that `NS` target names in both NS_z and NS'_z were considered for server selection.

Our analysis did not follow dependencies of general top-level domains (gTLDs), such as *com* and *edu*. There were two reasons for this: all descendants of gTLDs share the same top-level ancestry and was therefore uninteresting from the top level up; and the names of many of the gTLD servers are in the *gtld-servers.net* zone, so as we increased the probability ($P_{\{s,c\}}(v)$) that `NS` target names were cached as part of our analysis, the third-party influence of names having non-*net* gTLDs approached 1, which skewed the results. Our analysis did, however, follow country-code top-level domains (e.g., *us*, *fr*). The results from the survey are summarized in Table 3.5.

3.4.2 Model validation

To validate the name dependency model presented in Section 3.3 a random sample of over 600 of the ODP hostnames was selected, and a corresponding active dependency graph, G'_d, was constructed for each name, d. For each name the level of influence of each other domain name in the graph was calculated with $P_{NS}(z) = 0$.

We deployed BIND [26] as a resolver on more than 100 PlanetLab nodes [31], attempting to create an environment diverse enough that queries for each name by the collective resolvers would be uniformly distributed amongst authoritative servers.

On each PlanetLab node a query was issued to the name daemon 100 times for each name, d. Before the initial query of each name, the server's cache was flushed, so the source of every name resolved during the process could be identified, rather than relying on existing cached data from unknown sources. All DNS traffic to and from the server was monitored. Any address queries issued by the server were induced because of active influence on d. For every answer received for a name u during the resolution of d, u was mapped to the name of the server from which the answer was received. When the final response was received, containing the address corresponding to d, the names formed a graph of dependency paths from d to r representing the path(s) followed to resolve d, a subgraph of G'_d.

After each iteration, the addresses for any names resolved by induced queries were flushed from the server's cache and explicitly re-queried, before beginning the next iteration. This is equivalent to speeding up the expiration of the cached names. Without this action, the server would always respond with the cached name from the previously acquired source, and the likelihood of exploring other potential paths to the root would be diminished. After the 100 iterations of querying d, the influence of each other name, u, on d is determined by the calculating the fraction of the iterations in which u was included in the experimental graph.

We compared the observed dependency graph with the theoretical active dependency graph for each sample ODP name. For each name analyzed we verified that the influential names was a subset of those in V_d. The probability density function (PDF) of the difference in influence of each is shown in Fig. 3.2. The large peak in the graph demonstrates that 55% of the observed influence was exactly in line with the influence predicted by the model.

3.4.3 Trusted computing base

The raw size of the TCB for hostnames collected in terms of influential zones and non-trivial zones is shown in Fig. 3.3 as a cumulative density function (CDF), and the statistics are shown in Table 3.6. Nearly all hostnames have a TCB smaller than 20 zones when $P_{\{s,c\}}(v) = 0$, and the average size of the TCB was 2.26 non-trivial

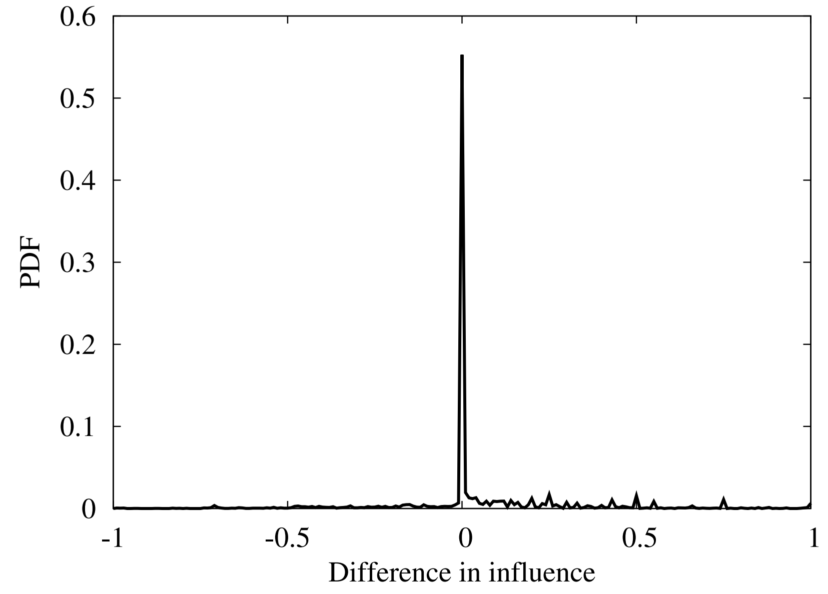

Figure 3.2: The distribution of differences between the theoretical and empirical level of influence for each sample ODP name. Positive values indicate that the model predicted more influence than was observed.

Metric	$P_{\{s,c\}}(v)$	Avg.	Max.
Influential zones	0	5.26	72
Influential zones	> 0	16.53	180
Non-trivial zones	0	2.26	45
Non-trivial zones	> 0	11.65	146
First-order zone ratio	0	0.92	1.0
First-order zone ratio	> 0	0.63	1.0
Third-party zone influence	0	0.08	1.0
Third-party zone influence	0.5	0.38	1.0
Third-party zone influence	1.0	0.55	1.0
Non-trivial organizations	0	1.67	37
Non-trivial organizations	> 0	8.41	113
Third-party organization influence	0	0.04	1.0
Third-party organization influence	0.5	0.34	1.0
Third-party organization influence	1.0	0.49	1.0

Table 3.6: TCB and influence statistics for the ODP/SC08 hostnames.

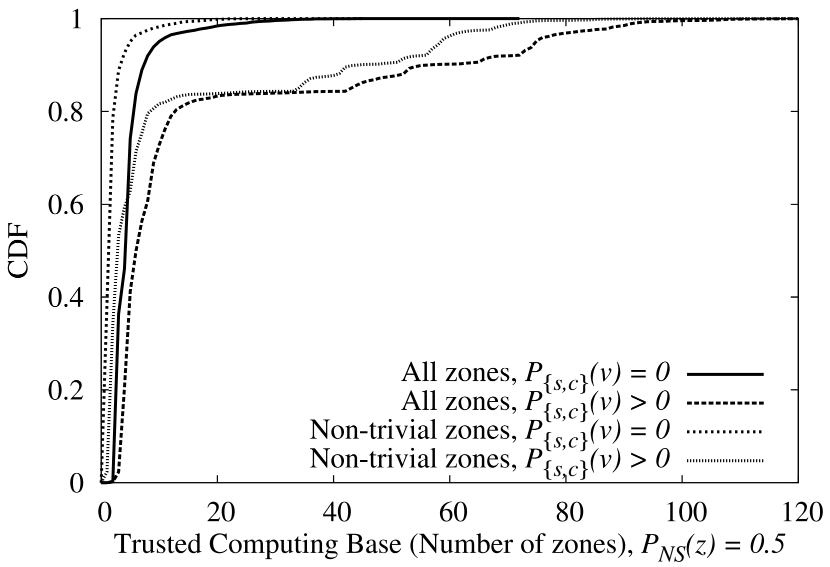

Figure 3.3: The CDF for the size of the TCB of ODP/SC08 hostnames. Included are the CDF for the number non-trivial and total zones in the TCB, for $P_{\{s,c\}}(v) = 0$ and $P_{\{s,c\}}(v) > 0$.

zones and 5.26 total zones—both of which are reasonably small. When $P_{\{s,c\}}(v) > 0$, the average size of the TCB increases several times to 11.65 non-trivial and 16.53 total zones. Only about 80% have fewer than 20 zones; most of the remaining 20% have between 30 and 90 non-trivial and total zones in their TCB. Caching and using `NS` target names from authoritative sources, rather than glue, can increase the size of the TCB of a domain by several times.

3.4.4 Controlled influence

The first-order ratio $\frac{U'_d}{U_d}$, shown in Fig. 3.4, is used to determine the percentage of non-trivial zones that are expressly configured by the administrators of d. Values closer to 1 indicate that the administrators are largely in control of the zones comprising the TCB. The average first-order ratio was 0.92 for $P_{\{s,c\}}(v) = 0$ and 0.63 for $P_{\{s,c\}}(v) > 0$, indicating that control of the TCB is lost as caching of `NS` target names is introduced. When $P_{\{s,c\}}(v) > 0$, third-party zones comprise more than half of the the non-trivial zones in the TCB of roughly 40% of the hostnames surveyed.

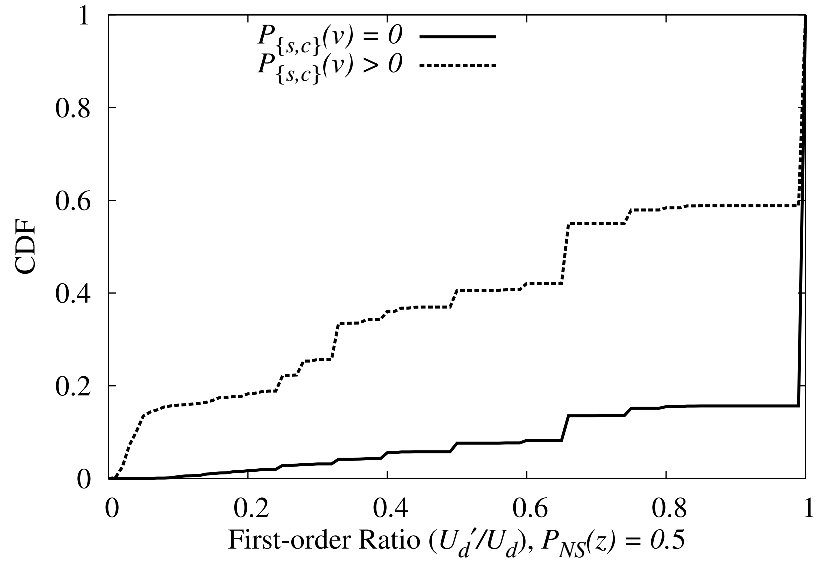

Figure 3.4: The CDF for the first-order ratio of the ODP/SC08 hostnames.

Fig. 3.5 shows the third-party influence of the ODP/SC08 hostnames. When $P_{\{s,c\}}(v) = 0$, 85% of the hostnames are not influenced at all by third parties. At $P_{\{S,C\}}(v) = 0.5$ only 60% of the hostnames are influenced less than 50% by third parties. When $P_{\{S,C\}}(v) = 1$ nearly half of the hostnames are influenced almost certainly by third parties. Again the behavior of caching preference of `NS` target names from authoritative sources at the resolver and authoritative servers greatly affects third-party influence of domain names.

3.4.5 Alias chains

One behavior affecting the third-party influence of domain names is the practice of chaining aliases. As an example, *www.example.com* is configured as an alias for *www.example.net* which, in turn, is an alias for *www.example.org*. The practice is common among content distribution networks for offloading the burden of addressing from the administrators of the original name (e.g., *www.example.com*). In our survey, we found that 33,873 (1%) of the ODP/SC08 names were affected by alias chains. Using that subset of affected names, we graphed the third-party influence

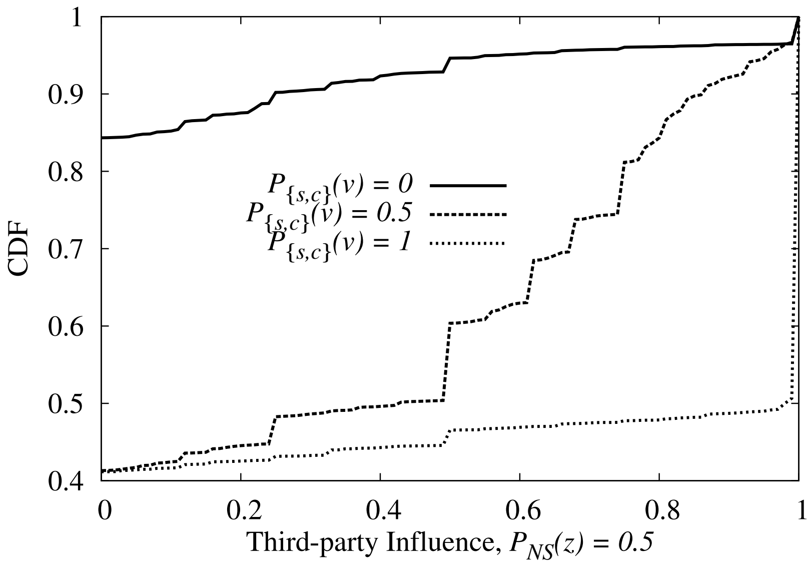

Figure 3.5: The CDF for the third-party influence of the ODP/SC08 hostnames, grouped by zone.

for analysis in Figure 3.6 using $P_{\{S,C\}}(v) = 0.5$. Only 10% of the names dependent on alias chains exhibit no third-party influence, compared to 42% of the entire set of ODP/SC08 names. However, when we removed the effects of the alias chains in our analysis, nearly half of the names avoided third-party influence completely, even with $P_{\{S,C\}}(v) > 0$.

3.4.6 Organizational dependencies

Throughout Sections 3.3 and 3.4 we have used zones as the unit of measurement for describing properties of the DNS name dependency graph. In reality the grouping of names or servers for analysis is arbitrary, and the methodology employed depends on certain assumptions and the desired results. Using zones as a unit of measurement may seem reasonable because of the assumption that security and reliability across the servers whose names are in a common zone are consistent. A similar assumption may hold for use of administering organizations as a unit of measurement. Analysis of domain name influence may yield different results when zones are grouped by organi-

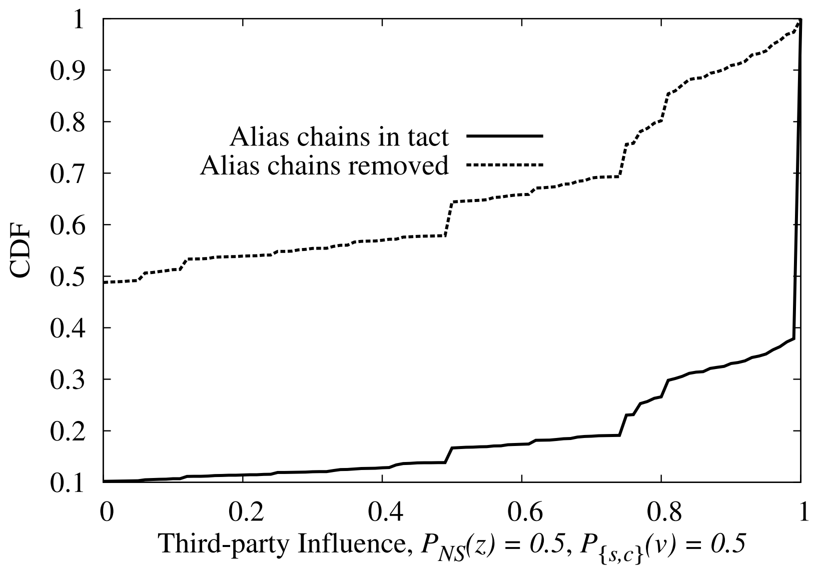

Figure 3.6: The CDF for the third-party influence of ODP/SC08 hostnames affected by alias chains, grouped by zone.

zation. For example, *sports.com* and *athletics.net* (Table 3.1) are both run by Sports Central, Inc. Although the *athletics.net* zone is a third party to *www.soccer.com*, when analyzed by organization it is grouped with *sports.com* under Sports Central, Inc., a first-order dependency.

Every zone is configured with the email address of a responsible person (`RNAME`) in its `SOA` (start-of-authority) RR. To further our analysis, we used the domain suffix from the email as a method for grouping different zones into a single organization, under the assumption that the email suffix would reflect the organization responsible for maintenance. The results are shown in Figure 3.7. Using this organizational grouping, we found that the number of domain names with no third-party influence increased from 85% to over 90% when $P_{\{S,C\}}(v) = 0$ and from 42% to 48% when $P_{\{S,C\}}(v) > 0$.

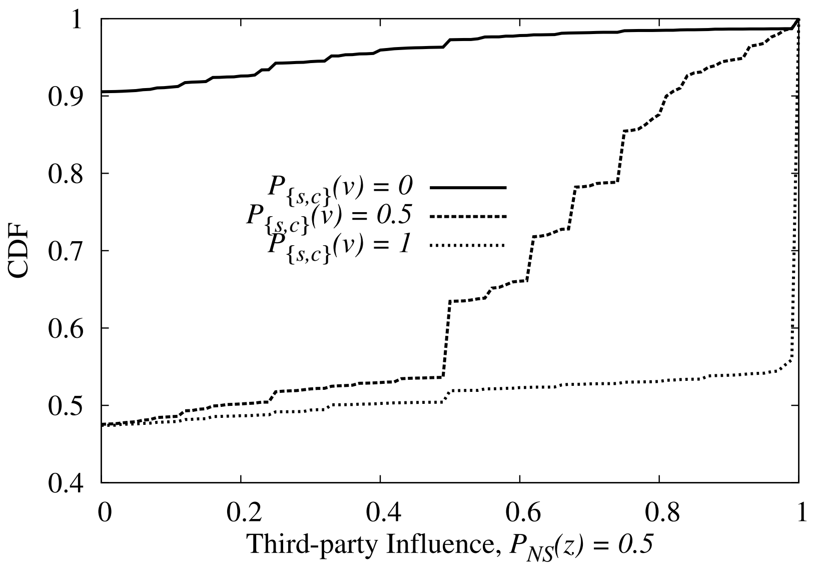

Figure 3.7: The CDF for the third-party influence of ODP/SC08 hostnames, grouped by organization.

3.5 Summary

In this chapter we have presented a graph model for analysis of name dependencies in the DNS, which was based on specification and behavior of deployed DNS servers. We defined the trusted computing base (TCB) of a domain name in terms of zones and organizations. We also derived metrics for assessing the dependency model of a domain name. Among these were the level of influence of influential domain names, and third-party influence—the probability that resolution of a domain name will utilize namespace outside the explicit configuration of domain administrators.

We observed that the TCB of domain names, when measured by influential zones and organizations, is much smaller than previously thought. On average 92% of the non-trivial zones in the TCB of a domain name were explicitly configured by the domain administrators. However, the practice of resolver and authoritative server using the address records for corresponding to `NS` targets from cache, rather than from additional records in a response or from glue, can increase the size of the TCB and the influence of third-party namespace significantly.

To maximize the reliability of name resolution from the perspective of both resolver and authoritative server, administrators and designers of DNS services should be aware of their server configurations as well as the names and organization comprising the TCBs of their domain names. Administrators should review the role of name servers in their environment to minimize the influence of third parties. The practice of chaining domain name aliases increases the potential for third-party influence and should be avoided, as suggested in RFC 1912 [13]. Additionally, we recommend that the roles of authoritative server and caching server be kept distinct or that authoritative servers be configured to not include information from their caches, as this creates additional dependence. As one data point, during our survey of the production DNS space we queried 6,033 distinct servers authoritative for a parent zone, in an attempt to detect glue records and discrepancies in the delegation records for a child zone. We found that 1,444 (24%) of these servers returned additional records whose TTL decreased with subsequent requests, indicating that they had come from the cache of the authoritative server.

A better understanding of DNS dependencies and an application of that understanding will put more control into the hands of DNS administrators and mitigate the risks associated with large and diverse TCBs and high third-party influence.

Algorithm 2 `FirstOrderDeps`(d)

Input: Domain name d

Output: Set of first-order dependencies in V_d

```
1: $N \leftarrow$ NonTrivialZones$(d)$
2: /* $M$ is the set of explicitly configured names for $d$ */
3: $M \leftarrow \{d\}$
4: if $d$ is not a zone then
5:     if $d$ is an alias then
6:         $M \leftarrow M \bigcup \{Cname(d)\}$
7:     end if
8:     $d \leftarrow Parent(d)$
9: end if
10: /* Add NS target edges for zone $d$ to $M$ */
11: $M \leftarrow M \bigcup \{u \in V_d | \exists (d, u) \in A_d$, an NS target dependency$\}$
12: $D \leftarrow \{d\}$
13: /* Add non-trivial zones in $M$'s ancestry to $D$ */
14: for all $u \in M$ do
15:     $v \leftarrow Parent(u)$
16:     while $v \neq r$ do
17:         if $v \in N$ then
18:             $D \leftarrow D \bigcup \{v\}$
19:         end if
20:         $v \leftarrow Parent(v)$
21:     end while
22: end for
23: return $D$
```

Algorithm 3 `ControlledAlias`(u, D)

```
Input: Domain name $u$
Input: Set of first-order dependencies $D$
Output: False if $u$ directly or indirectly aliases a name outside explicit dependency;
    True otherwise
 1: $H \leftarrow \{u\}$
 2: while $u$ is an alias do
 3:     if $Parent(Cname(u)) \notin D$ then
 4:         return False
 5:     else if $Cname(u) \in H$ then /* Loop detected */
 6:         return True
 7:     end if
 8:     $H \leftarrow H \bigcup \{u\}$
 9:     $u \leftarrow Cname(u)$
10: end while
11: return True
```

Algorithm 4 `ThirdPartyInfluence1`(u, D)

```
Input: Domain name $u$
Input: Set of first-order dependencies $D$
Output: Influence on $u$ by names outside of $D$
1: if $u$ is not a zone then
2:     /* $u$ aliases a name outside of $D$ */
3:     if ControlledAlias$(u, D) = False$ then
4:         return 1.0
5:     end if
6:     $u \leftarrow Parent(u)$
7: end if
8: $P \leftarrow 0$
9: /* Aggregate influence outside $D$ for $u$'s ancestors */
10: while $u \neq r$ do
11:     $P_u \leftarrow 0$
12:     for all $v \in V_d | \exists (u, v) \in A_d$, an NS target dependency do
13:         if $Parent(v) \notin D$ or ControlledAlias$(v, D) = False$ then
14:             $P_u \leftarrow P_u + w(u, v)$
15:         end if
16:     end for
17:     $P \leftarrow 1 - (1 - P)(1 - P_u)$
18:     $u \leftarrow Parent(u)$
19: end while
20: return $P$
```

Algorithm 5 `ThirdPartyInfluence`(d)

```
Input: Domain name $d$
Output: TPI of $d$
1: $D \leftarrow$ FirstOrderDeps$(d)$
2: $P_A \leftarrow 0$
3: if $d$ is not a zone then
4:     /* If $d$ is an alias, calculate the TPI of $Cname(d)$ */
5:     if $d$ is an alias then
6:         $P_A \leftarrow$ ThirdPartyInfluence1$(Cname(d), D)$
7:     end if
8:     $d \leftarrow Parent(d)$
9: end if
10: /* Calculate the TPI of $Parent(d)$ */
11: $P_P \leftarrow$ ThirdPartyInfluence1$(Parent(d), D)$
12: /* Calculate the TPI of each NS target of zone $d$ */
13: $P_{NS} \leftarrow 0$
14: for all $u \in V_d | \exists(d, u) \in A_d$, NS target dep. do
15:     $P_{NS} \leftarrow P_{NS} + w(d, u)$ThirdPartyInfluence1$(u, D)$
16: end for
17: /* Aggregate the TPI of all name dependencies */
18: return $1 - (1 - P_P)(1 - P_A)(1 - P_{NS})$
```

Chapter 4

DNS Availability Analysis

The dependency model presented in Chapter 4 creates a foundation for understanding prerequisites for name resolution. The name dependency graph for a domain name describes the other names which are collectively responsible for its correct resolution. Improper configuration of downstream name dependencies may result in increased potential for name resolution failure, regardless of the robustness and security applied to the configuration of the domain name itself.

The availability of a domain name refers to its ability to be reliably resolved using the DNS. In this chapter 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 the DNS in light of the availability model and discuss our observations and inferences.

The primary contributions of this chapter are:

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

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

In Section 4.1 we discuss previous research related to that presented in this chapter. Section 4.2 extends the concept of DNS dependencies from Chapter 3 to include server dependencies. In Section 4.3 we introduce metrics to measure the availability of domain names, as well as discussion on common DNS misconfigurations. Section 4.4 describes our methodology for data collection and contains an analysis of live DNS data. We summarize our findings in Section 4.5.

4.1 Previous work

The state of the DNS is presented in several surveys of production DNS deployments [4, 32, 33]. Various misconfigurations are analyzed, including lame delegation, diminished server redundancy, cyclic dependencies, and inconsistency of `NS` RRsets between parent and child zones. In this chapter we derive a theoretical availability model and methodology to systematically identify such misconfigurations and quantify their impact on availability.

DNS availability and robustness have been analyzed in other studies [34, 1]. In these empirical studies DNS availability was measured from a perspective of responsiveness of resolvers and authoritative servers, and diversity of DNS performance from different client perspectives. In our analysis, we apply our theoretical model to a deployment of a domain name and its dependencies to measure its availability.

4.2 DNS server dependencies

A natural extension to the DNS name dependency model presented in Chapter 3 is the addition of server dependencies. Such extension allows us to examine dependencies which contribute to both the performance, security, and robustness of name resolution, not simply namespace diversity. We refer to the fictitious zone data in Table 4.1 for examples throughout the remainder of this chapter. We also use notation from Table 3.2 for our analysis.

The research in Chapter 3 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 [28]. Since this chapter is concerned with domain name availability, we consider only active influence when applying it to server dependencies.

Just as a domain name may be dependent on other domain names, it also depends on name servers, identified by Internet address. We model server dependencies by extending the active influence name dependency graph for domain name d, $G'_d =$

$ORIGIN foo.net.			
	Name	Type	Value
1	foo.net.	NS	ns1.foo.net.
2	foo.net.	NS	ns2 .foo.net.
3	foo.net.	NS	ns1.bar.com.
4	foo.net.	NS	ns3.bar.com.
5	ns1.foo.net.	A	192.0.2.1
6	ns2.foo.net.	A	192.0.2.2

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

$ORIGIN bar.com.			
	Name	Type	Value
1	bar.com.	NS	ns1.bar.com.
2	bar.com.	NS	ns2.bar.com.
3	ns1.bar.com.	A	192.0.2.5
4	ns2.bar.com.	A	192.0.2.6
5	ns3.bar.com.	A	192.0.2.7

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

Table 4.1: Example zone data for several fictitious zones.

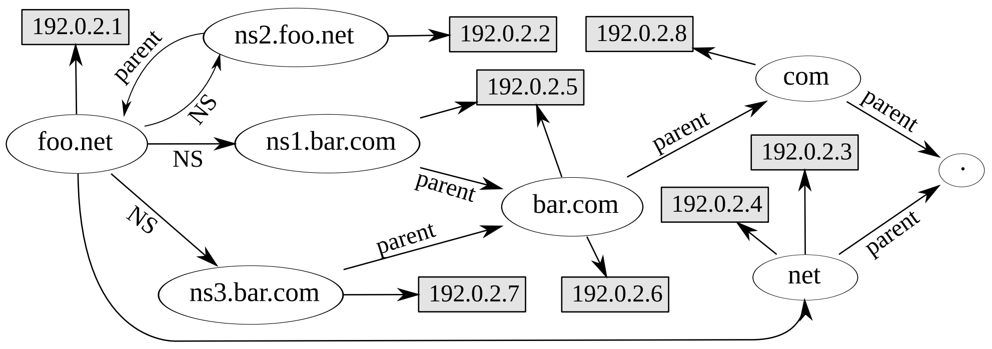

Figure 4.1: The server dependency graph for *foo.net*, derived from the zone data in Table 4.1. The gray, rectangular nodes represent name servers, and the oval nodes represent domain names.

(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 $W_d = V'_d \bigcup \{\text{name servers}\}$. 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. 4.1 shows the server dependency graph derived from the data in Table 4.1.

4.3 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 [28], but the parent of the name is required independent of the others. This concept is displayed for *foo.net* in Fig. 4.2, 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 4.3.2.

4.3.1 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

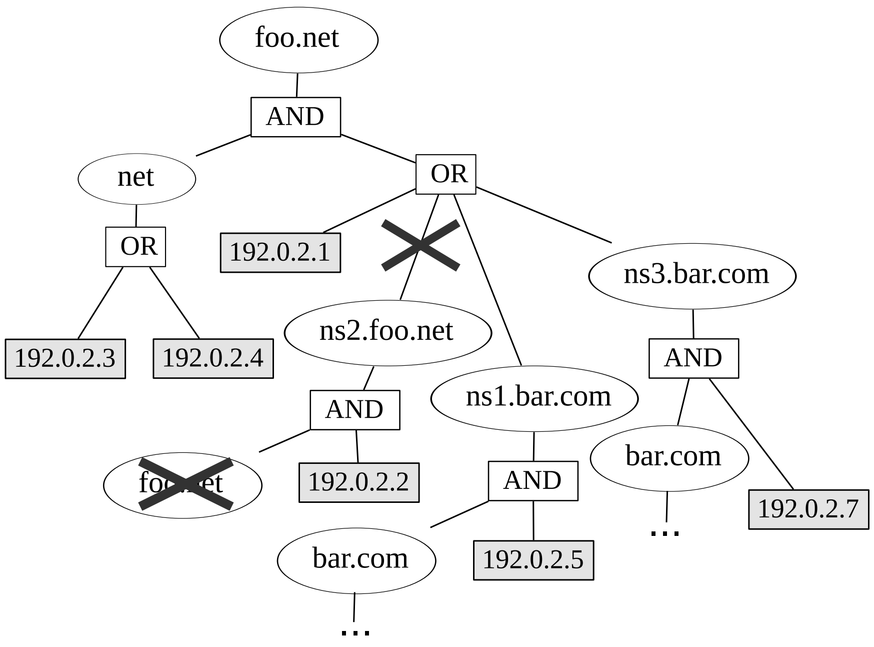

Figure 4.2: A logical tree describing the availability of *foo.net*.

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.

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`$(d, \emptyset)$ (Algorithms 6 and 7). In a logical sense, `FindMSQ` recursively performs a conversion of the Boolean availability expression for d, such as that shown in Fig. 4.2, 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. 4.3 portrays the logical structure resulting from recursively reducing *foo.net* by calling `FindMSQ`$(foo.net, \emptyset)$.

Algorithm 6 `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: end if
 8: $J \leftarrow J \bigcup \{u\}$ /* Add $u$ to history */
 9: $MSQ_{Parent} \leftarrow$ FindMSQ$(Parent(u), J)$
10: if $u$ is a zone then
11:     /* Direct server and NS dependencies of $u$ */
12:     $D \leftarrow \{\forall v \in NSA'_u \bigcup NS'_u | \exists (u, v) \in B_d\}$
13: else
14:     /* Direct server and alias dependencies of $u$ */
15:     $D \leftarrow \{\forall v \in S_u \bigcup \{Cname(u)\} | \exists (u, v) \in B_d\}$
16: end if
17: /* Continued in Algorithm 7... */
```

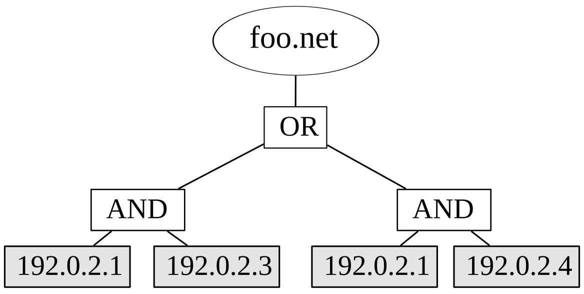

Figure 4.3: A logical tree describing the MSQ of *foo.net*.

Algorithm 7 `FindMSQ`(u, J) (continued from Algorithm 6)

```
18: /* Find minimum MSQ among mutually exclusive dependencies */
19: $MSQ_{Other} \leftarrow \emptyset$
20: $msq \leftarrow \infty$
21: for all $v \in D$ do
22:     $MSQ'_{Other} \leftarrow$ FindMSQ$(v, J)$
23:     $msq' \leftarrow \min_{s \in MSQ'_{Other}} |s|$
24:     if $msq' < msq$ then
25:         $MSQ_{Other} \leftarrow MSQ'_{Other}$
26:         $msq \leftarrow msq'$
27:     else if $msq = msq'$ then
28:         $MSQ_{Other} \leftarrow MSQ_{Other} \bigcup MSQ'_{Other}$
29:     end if
30: end for
31: /* Find smallest union of $MSQ_{Parent}$, $MSQ_{Other}$ */
32: $msq \leftarrow \infty$
33: $MSQ \leftarrow \emptyset$
34: for all $MSQ'_P \in MSQ_{Parent}$, $MSQ'_O \in MSQ_{Other}$ do
35:     $MSQ' \leftarrow MSQ'_P \bigcup MSQ'_O$
36:     if $|MSQ'| < msq$ then
37:         $MSQ \leftarrow \{MSQ'\}$
38:         $msq \leftarrow |MSQ'|$
39:     else if $|MSQ'| = msq$ then
40:         $MSQ \leftarrow MSQ \bigcup \{MSQ'\}$
41:     end if
42: end for
43: $MSQ(u) \leftarrow MSQ$ /* Store the value for future use */
44: return $MSQ$
```

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 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 work. 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.

4.3.2 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 the 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 [4], 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. 4.2 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., $NS_z \neq NS'_z$). 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.

4.4 Data collection and availability analysis

We built a graph of name and server dependencies for each ODP/SC08 hostname (from Chapter 3) by recursively following all dependencies of each name. The results of our analysis are contained in this section.

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. 4.4 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 val-

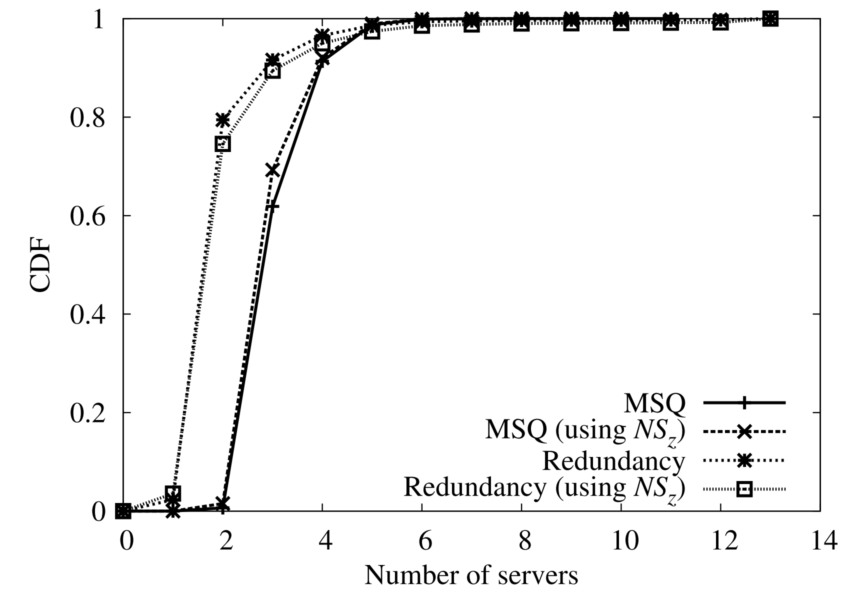

Figure 4.4: The availability of production DNS names, in terms of MSQ and redundancy.

ues 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.

4.5 Summary

In this chapter, we have extended our name dependency model from Chapter 3 to formalize a model for server dependencies in the DNS. Using that model we have derived metrics for quantifying the availability of domain names. We 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 the 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.

Chapter 5

DNSSEC Availability Analysis

The availability of a domain name as analyzed in Chapter 4 can be affected by improper configuration of any zones or servers on which it is dependent. DNSSEC adds new components that further impact the availability of a domain name.

As early adopters have begun signing zones and enabling validation, experience has shown that DNSSEC requires significantly more administration than standard DNS. The complexities of DNSSEC are fertile ground for misconfigurations which inhibit proper name resolution, and the availability of dependent names is much more likely to be affected by misconfiguration. For example, in June, 2010, signatures accompanying DNS records in the *arpa* zone expired[1]. This zone is the authority for all reverse name resolution—mapping Internet addresses to domain names. When the signatures expired, validating resolvers were unable to successfully perform reverse lookups until the zone was re-signed.

In this chapter we present a model to quantify the impact of DNSSEC-related misconfigurations, and we introduce several metrics to quantify the complexity of a DNS architecture, with respect to its hierarchical ancestry and administrative diversity. We propose a system for soft anchoring to minimize the impact of misconfigurations on name resolution. Using production DNSSEC data we show the pervasiveness of DNSSEC-related misconfigurations and show how our soft anchoring approach helps maintain availability of otherwise unreachable zones. We list the following as the major contributions of this chapter:

- An analysis of misconfigurations related to DNSSEC deployment and a survey of their pervasiveness in production DNS.
- A model to quantify the potential for resolution failure of zones due to DNSSEC misconfiguration.
- Metrics to measure complexity of a DNS setup which impact the failure potential of zones for which DNSSEC is deployed.
- A mechanism for soft anchoring to increase robustness in spite of DNSSEC misconfiguration.

[1]http://www.dnssec-deployment.org/pipermail/dnssec-deployment/2010-June/004009.html

In Section 5.1 we present our model for analyzing DNSSEC availability and introduce metrics for quantifying the complexity of DNS configurations which increase the potential for failure. Section 5.2 describes our proposal for increasing robustness through soft anchoring. Section 5.3 discusses our analysis of production DNS data in light of our model. Related work is summarized in Section 5.4, and we summarize our findings in Section 5.5.

5.1 Modeling DNSSEC availability

While DNSSEC has the obvious advantage of allowing a resolver to cryptographically verify the answer given for a domain name query, it adds complexity to the requirements for name resolution, and increases the potential for failure. Any server or zone misconfiguration in the line of trust between anchor and query name widens the target of error. In this section we present a model for availability of domain names in a DNSSEC deployment. Our model focuses solely on the issue of availability due to improper DNSSEC configuration, and does not consider incorrect responses which are the result of stale zone data (except as it relates to DNSSEC), cache poisoning, or other malicious injection.

5.1.1 Failure potential

Chapter 2 lists the possible results of a DNSSEC validation from the perspective of a resolver as *secure*, *insecure*, and *bogus*. When a resolver ultimately determines that the answer that it has resolved is bogus, it issues a failure message to the requester. However, the mechanism for determining that the answer is bogus is specific to implementation. Some resolver implementations, after encountering the bogus from a single authoritative server, continue trying each of the remaining authoritative servers until they get a valid response—or exhaust all possible avenues. Such is the case with Internet Systems Consortium's (ISC) Berkeley Internet Name Daemon (BIND) version 9.6 [26].

However, even when the validating resolver is so diligent, failure due to misconfiguration is still possible in the following circumstances: 1) The zone data is invalid on all of the zone's responsive authoritative servers; or 2) the resolver is behind one or more *proxy* resolvers to which it is configured to forward its requests. In the former case the resolver will exhaust all possibilities for success after querying each of the available servers. In the latter, the validating resolver is at the whim of the proxy resolvers, regardless of whether or not the upstream resolvers are configured for validation or have practiced validation diligence to obtain appropriate DNSSEC responses. For these reasons, we consider the more general case in which an invalid response received from any misconfigured server results in a validation failure.

We group the causes for validation failures into three classes:

- *Zone*: Missing, expired, or otherwise invalid `RRSIG`s covering zone data; *or* missing `DNSKEY` RRs required to verify `RRSIG`s.
- *Delegation*: Bogus delegations caused by lack of appropriate `DNSKEY`s in a child zone corresponding to `DS` RRs in the parent zone; *or* insufficient `NSEC` RRs to prove an insecure delegation to a resolver.
- *Anchor*: Stale trust anchors in a resolver, which no longer match appropriate `DNSKEY`s in the corresponding zone.

We quantify *failure potential* by determining the probability that the resolver queries a misconfigured server or misconfigured zone during resolution of a domain name, which would consequently result in a validation failure.

We establish some notation for our analysis. Let $z(i)$ denote the zone i generations above zone z, such that $z(0) = z$, $z(1) = Parent(z)$, and $z(m)$ is the root zone. Let NS_z and NSA_z denote the sets of names and addresses of servers authoritative for zone z, respectively. Figure 5.1 illustrates a DNS setup with a stub resolver configured with n recursive resolvers $r_1, r_2, \ldots, r_n$, resolving names in zone z.

We identify the set of servers authoritative for z which are both resolvable and responsive, denoted $NSA'_z \subseteq NSA_z$. If server $u \in NS_z$ is a subdomain of z's parent zone and z's parent zone provides an address (*glue record*) for u, then a resolver can use the glue address [12, 28]. This address is included in NSA'_z, provided the server

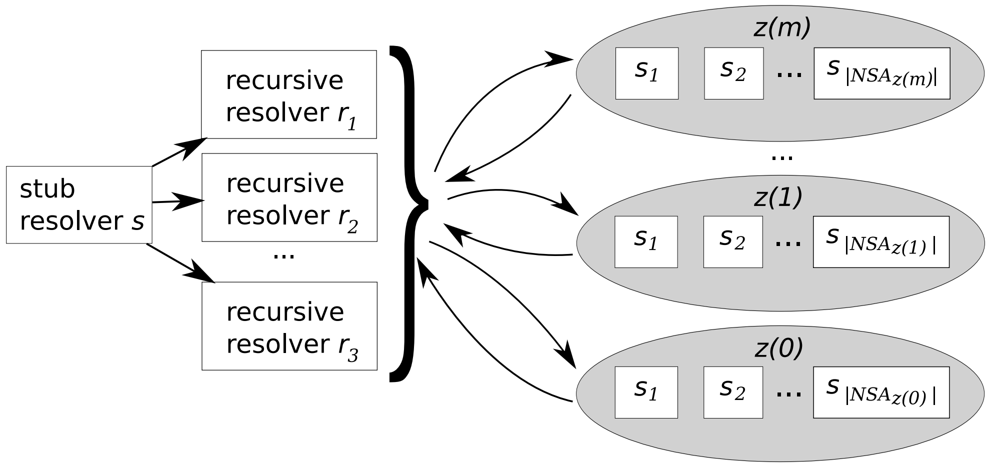

Figure 5.1: A typical DNS setup in which a client (stub resolver) is configured to use n recursive resolvers which resolve a name in zone z with an ancestry of size m.

is responsive. Otherwise, a resolver must resolve and validate u, the result of which, if successful, is included in NSA'_z. Non-responsive servers are excluded from NSA'_z because although a resolver may query the server, upon timeout the resolver will reissue the query to another authoritative server [12], in contrast to the scenario in which a server responds with incomplete or bogus data.

Let $B(z) \subseteq NSA'_z$ denote the set of servers authoritative for zone z which serve bogus or incomplete data for zone z, resulting in a *zone*-class misconfiguration. We assume that any authoritative server has an equal chance of being selected for query by the resolver. Under such circumstances the probability, $P'_f(z)$, that the resolver queries a server whose response results in a bogus validation is:

$$P'_f(z) = \begin{cases} 1.0 & \text{if } NSA'_z = \emptyset \\ \frac{|B(z)|}{|NSA'_z|} & \text{otherwise} \end{cases} \tag{5.1}$$

Because validation must follow the authentication chain from the zone in question to a trusted anchor, we include the servers authoritative for zones in z's ancestry, $z(0)$ through $z(m)$, as illustrated in Figure 5.1. Let $z(a), a \in [0, m]$ denote the zone for which the resolver is configured with a trust anchor. While it is possible that a resolver is configured with multiple trust anchors within the same hierarchy, we assume for the purposes of this chapter that for a given zone hierarchy at most one

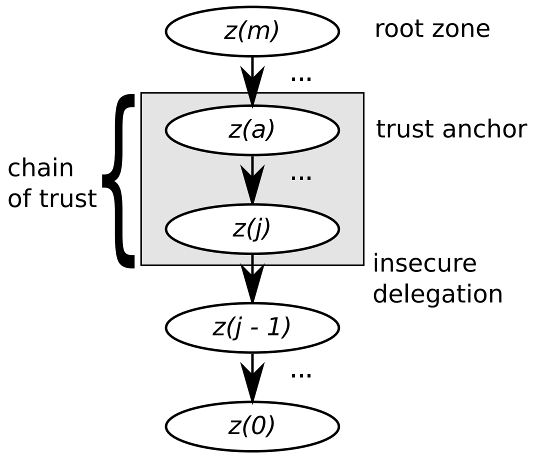

Figure 5.2: An illustration of the delegation model for zone z with ancestry $z(0)$ to $z(m)$ and chain of trust extending from $z(j)$ to $z(a)$, anchored at zone $z(a)$.

trust anchor exists.

We also consider the case of insecure delegation. Let $z(j), j \in (0, a]$ denote a zone such that 1) the delegations between $z(a)$ and $z(j)$ are linked with a chain of trust and 2) the delegation from zone $z(j)$ to zone $z(j-1)$ is insecure. Figure 5.2 illustrates the delegation model with this notation.

When extending the failure potential model for zone z to include its entire ancestry we must now consider *zone*-class DNSSEC problems in zones $z(i), i \in [j, a]$, *delegation*-class problems in zone $z(i)$ affecting delegation to $z(i-1)$, $i \in [j, a]$, and *anchor*-class problems in zone $z(a)$. We denote the sets of servers serving such bogus data for zone $z(i)$ as $B_z(z(i))$, $B_d(z(i))$, and $B_a(z(i))$, respectively. We combine the sets to form the comprehensive set referenced in Equation 5.1:

$$B(z(i)) = B_z(z(i)) \cup B_d(z(i)) \cup B_a(z(i)) \tag{5.2}$$

Note that for other zones, outside the chain of trust, non-anchor-class DNSSEC errors are innocuous, regardless of whether or not $z(i)$ is signed, such that $B_z(z(i)) = \emptyset$ and $B_d(z(i)) = \emptyset$. We now combine the probabilities of querying a misconfigured server in the preceding zones as independent events to define the potential for zone z to fail validation by a resolver:

$$P_f^r(z) = 1 - \prod_{i=0}^{m} \Big(1 - P_f'(z(i))\Big) \tag{5.3}$$

If responses across authoritative servers are consistent for all zones $z(i), i \in [j, a]$ then $P_f^r(z)$ will always be 0 or 1—all resolvers from the same vantage point will either fail together or succeed together. However, when authoritative servers exhibit inconsistent DNSSEC results, the failure potential is a fraction. In such cases, the probability of bogus response is the same for any resolvers having the same vantage point, but failure potential is reduced exponentially with each attempt by a distinct recursive resolver since they validate independently. Common behavior for a stub resolver receiving a `SERVFAIL` message is failover to its next configured recursive resolver. We therefore model the failure potential for a zone as experienced by a stub resolver configured with n recursive resolvers (as in Figure 5.1) as follows:

$$P_f^s(z) = \left(1 - \prod_{i=0}^{m} \left(1 - P_f'(z(i))\right)\right)^n \tag{5.4}$$

With this availability model we have only considered a common DNS configuration exemplified in Figure 5.1. Our figures for failure potential are calculated with the assumption of a single validating resolver for a stub resolver (i.e., $n = 1$). The model may be modified to suit alternate configurations. Also, we do not consider the effects of TTL values and caching on misconfiguration, which may affect the duration for which a problem persists.

5.1.2 Quantifying complexity

The disregard for DNSSEC maintenance can result in increased failure potential. One important necessity is careful coordination both hierarchically (i.e., between parent and child zones) or laterally, between organizations hosting each others' data. The hierarchical relationship is the most crucial, since the chain of trust extends vertically, and a break in the chain results in general failure to the namespace below. However, this coordination is less demanding because it generally involves only two entities. Problems caused by lateral coordination may be less severe, in particular if a resolver exercises validation diligence. However, the complexities still result in increased failure potential.

We present two metrics to quantify the complexity of a DNS zone. The metrics themselves are calculated independent of DNSSEC deployment, but higher metric values may increase the failure potential for signed zones because they indicate more areas where problems may occur. The first metric is the *hierarchical reduction potential* (HRP), which quantifies how much the ancestry of a zone might be reasonably consolidated to reduce hierarchical complexity. The second is *administrative complexity* which describes the diversity of a zone, with respect to organizations administering its authoritative servers.

The *depth* of a zone is measured by its distance from the root zone. For example, zone z has ancestry $z(0), z(1), \ldots, z(m)$ comprised of $m + 1$ zones and has a depth of m. Each ancestral zone $z(i)$ contributes to the failure potential for zone z, as it is an additional requirement of DNSSEC correctness that must be consistent. While delegation is necessary in many cases, there are some cases in which collapsing a delegated zone is both reasonable and possible. For example, if *example.com* and *sub.example.com* are two zones administered by the same organization, the zone data for *sub.example.com* might trivially be migrated to the *example.com* zone and the delegation to *sub.example.com* removed. Records previously a part of the *sub.example.com* zone would be added to the *example.com*: e.g., *www* in *sub.example.com* becomes *www.sub* in *example.com.* This consolidation reduces the number of zones ancestral to *sub.example.com* by 0.25 from 4 to 3.

We express the HRP of zone z, having $m + 1$ ancestral zones, as the fraction of layers that could be reduced if the number of zones is consolidated to $m' + 1$:

$$HRP_z = \frac{m - m'}{m + 1} \tag{5.5}$$

A greater HRP value indicates that failure potential might be reduced by minimizing hierarchical complexity.

Administrative complexity is the side effect of having multiple organizations host authoritative data for a zone. While third-party hosting is often advantageous to gain geographic and network diversity for high availability, the coordination increases the potential for failure. Unilateral changes to server address, firewalls, or software version or configuration could ultimately result in a lapse in synchronization between zone data hosted by servers from two organizations. The number distinct organizations serving authoritative data for a zone provides one means of measuring this diversity. However, to characterize the distribution we propose a metric which determines the probability that given n random selections with replacement from the servers authoritative for z, the servers selected are not administered by the same organization $o \in O_z$:

$$AC_n(z) = 1 - \sum_{o \in O_z} \left(\frac{|NSA_z^o|}{|NSA_z|} \right)^n \tag{5.6}$$

where O_z denotes the set of organizations administering servers which are hosting zone data for z and $NSA_z^o \subseteq NSA_z$ denotes the subset of servers in NSA_z administered by organization $o \in O_z$.

For example, assuming the servers *ns.example.com* and *ns.example.net* are operated by two separate organizations and are the only authoritative servers for *example.com*, the administrative complexity of *example.com* with $n = 2$ is:

$$AC_2(\mathit{example.com}) = 1 - \left(\left(\frac{1}{2} \right)^2 + \left(\frac{1}{2} \right)^2 \right) = 0.5$$

Accurate calculation of the HRP and administrative complexity for a zone requires some knowledge of the administrative nature of the zone not necessarily available by observation. We have made some inferences based on observation to estimate these metrics. We use the email suffix of the *responsible name* (`RNAME`) field from the *start of authority* (`SOA`) RR of a zone as a value for comparison. For HRP we assume that if the `RNAME` email suffix between parent and child zone matches, they are administered by the same organization, and that reducing the size of the zone's ancestry by 1 might be possible. For example, the *it.ohio-state.edu* and *ohio-state.edu* zones share an `RNAME` value of *hostmaster@osu.edu* and might from an outside perspective be considered for collapse into a single zone. We notably excluded the possibility of collapsing delegations from the root zone or TLDs, even if they shared common `RNAME` fields, which is the case with *com* and the root zone. Such consolidation would simply be unrealistic and infeasible.

For administrative complexity, we also used the `RNAME` email suffix to identify an organization responsible for administering an authoritative server. For each server authoritative for z, we identified the zone to which its server name $u \in NS_z$ belongs.

For example, servers from the *uoregon.edu* and *lsu.edu* zones are authoritative for *lsu.edu*. Each zone has a distinct `RNAME` suffix, so we classify these as two organizations. There may be some false positives (a single organization managing multiple zones with distinct `RNAME` values), but overall the comparison should be representative.

5.2 Soft anchoring

The metrics presented in Section 5.1.2 quantified factors that may increase the failure potential of a zone and its servers and implicitly suggested means for mitigation, e.g., by collapsing zones with common administration to reduce the size of a zone's ancestry. We now present an alternative method to mitigate failure potential for zones by shifting the focus from the authoritative servers to validating resolvers. We propose the use of additional trust anchors by the resolver for each zone in an authentication chain to decrease the reliance of each ancestor "link" in the chain. In this section we provide an explanation of the reasoning behind our proposal, as well as its implementation.

5.2.1 Extending the DNSSEC trust model

The authenticity of `DNSKEY`s in zones $z(j)$ through $z(a-1)$ is established by following the authentication chain to the trust anchor $z(a)$. The `DNSKEY` RRset for each zone is cached by validating resolvers until its TTL expires or its covering `RRSIG` expires—whichever occurs first [7]. Upon expiration it must be re-authenticated to the trust anchor at zone $z(a)$. However, `DNSKEY` values often remain static well beyond their expiration. Current recommended rollover practices suggest rollovers be performed on granularity of months or years, depending on algorithm, key size, and key function (KSK or ZSK) [25]. We therefore propose that validating resolvers install additional trust anchors at descendant zones, as `DNSKEY`s are authenticated through the established chain of trust.

If the resolver is configured with a trust anchor at each zone $z(i), i \in [j, a]$, then validation of each is self-contained. By installing additional anchors from authenticated `DNSKEY`s, the burden carried by ancestor zones to properly validate is eased, and each signed zone must only ensure the correctness of its own data. The result is that validation of zone z is unaffected by: zone- and anchor-class misconfigurations from zones $z(i), i \in (j, a]$; and delegation-class misconfigurations affecting delegations from zone $z(i)$ to $z(i-1)$, for $i \in (j, a]$.

5.2.2 Soft anchor management

To implement our trust anchor extension, we introduce the notion of *soft* anchors, in addition to traditional trust anchors, which we refer to as *hard* anchors. Hard and soft anchors are both used by resolvers for validation and are formally defined by the following rules of transitivity:

- A trust anchor installed on a resolver by a DNS administrator and verified out-of-band is a *hard anchor*.
- A trust anchor authenticated within the same `DNSKEY` RRset as a hard anchor is also a *hard anchor*.
- A trust anchor authenticated by establishing an authentication chain from a hard or soft anchor to a SEP in a descendant zone is a *soft anchor*.
- A trust anchor authenticated within the same `DNSKEY` RRset as a soft anchor is also a *soft anchor*.

Resolvers must maintain the source of each trust anchor in its repository as either a hard or soft anchor.

Hard anchors are maintained following procedures detailed in RFC 5011 [35]. Resolvers periodically poll the anchored zone for updates. A resolver adds a new trust anchor when a new `DNSKEY` RR with the SEP bit set is detected in the `DNSKEY` RRset of the anchored zone, after it has existed for longer than a specified *hold-down* time. Resolvers learn of `DNSKEY` revocation when they see a self-signed `DNSKEY` with the revoke bit set.

Our proposed procedures for managing soft anchors extend RFC 5011 for anchor addition and removal. A resolver periodically polls to re-authenticate the chain leading to the soft anchor. Soft anchors are added by a resolver when:

- the resolver detects a self-signed `DNSKEY` RR with the SEP bit set and which corresponds to an authenticated `DS` RR in the parent zone; or
- the resolver detects a new `DNSKEY` with the SEP bit set within a `DNSKEY` RRset already authenticated by an existing a soft anchor, and the `DNSKEY` persists for a hold-down period.

A hold-down period is only required in the second case because the existence of a `DS` RR in the first case suggests a legitimate addition by the zone administrator. The hold-down period specified by RFC 5011 is the greater of 30 days or the TTL of the `DNSKEY` RRset. However, soft-anchored zones don't have reason to implement RFC 5011 because they are linked via a chain of trust to their parent zone and don't anticipate being anchored by resolvers. For such zones with `DNSKEY` TTL less than 30 days (quite likely), a new key might be introduced and used to sign the `DNSKEY` RRset exclusively as a SEP before a hold-down period is complete. The resolver would be unable to add it as a soft anchor in that case (unless its `DS` RR was also detected). This would undermine the continuity needed to protect against a bad KSK rollover, in which the new SEP `DNSKEY` is rolled successfully, but the corresponding `DS` RRs are not. Without a new soft anchor, a resolver cannot legitimately validate the `DNSKEY` RRset until the remainder of the 30 days has passed. To remedy this we suggest a hold-down period matching the lesser of the `DS` TTL and `DNSKEY` TTL, which is sufficient for a KSK rollover following procedures documented in RFC 4641 [25].

Soft anchors are removed from a resolver's repository when:

- a `DNSKEY` RR is revoked, following RFC 5011;
- the `DS` RR previously in existence for a soft anchor has been removed; or
- the chain of trust from hard to soft anchor has been securely unlinked, such that there is no longer a path to authenticate the soft anchor.

The first two items indicate explicit revocation either by RFC 5011 standards or by removing the link from its parent. The third item recognizes the intent of the administrator of a zone or its ancestor to prevent a path for validation; soft anchors were not necessarily intended by the zone administrators to be maintained by resolvers as trust anchors.

Validation is first attempted with the soft anchor in the closest ancestor zone. When validation with soft anchors at each ancestor zone have failed, then the hard anchor is used for validation.

Although the soft anchoring approach will decrease dependence on the DNSSEC correctness of ancestor zones in the chain of trust, it is only a mitigation technique. Most notably if a problem exists at zone $z(i), i \in [0, j]$, then the availability of zone z will still be affected because there is no further soft anchoring below $z(j)$. Also, a valid chain of trust must extend to $z(j)$ to bootstrap soft anchoring. A resolver attempting to validate z for the first time will experience failure if the chain of trust is broken.

5.3 Data collection and analysis

We performed a study of production DNS data in light of our availability model. Our seed data came from two sources: the ODP/SC08 hostnames (from Chapter 3) and names submitted via the Web interface of the DNSViz analysis tool [36].

An important part of our study is selecting a representative subset of production signed zones for analysis. With this objective, our data set includes fewer signed zones than other analyses [37, 38] because we only crawled dependencies and didn't traverse `NSEC` RRs of signed zones. Names in our data set were either indexed by ODP, queried by clients at SC08, submitted by interested parties, or a were a dependency of such a name, so we justify our data set as a representative subset of production zones.

Although 2,170 zones from our data were signed, we needed to further distinguish production signed zones from non-production. We accomplished this by first excluding zones whose names contained “test” or ”bogus” or that were subdomains of known DNSSEC test namespaces (e.g., *dnsops.gov*, of the Secure Naming Infrastructure Pilot [39]). We further filtered zones by including only islands of security that had some public intent to be validated by resolvers. Because the root zone was not signed during the majority of our analysis, we identified the subset of islands whose top-most domain had registered with ISC's *DNSSEC Look-aside Validation* (DLV) service [40]. DLV [41] was introduced to allow an arbitrary zone to be securely linked to a zone other than its hierarchical parent. The result is that a resolver can avoid the maintenance of a large repository of trust anchors in favor of a single trust anchor for the DLV service. Other DLV services exist [37, 38], but are populated with `DNSKEY`s discovered through DNSSEC polling, which means that users may not have explicitly opted in for production validation. We therefore consider only zones registered with ISC's DLV service as production signed zones.

Based on our qualifications for production signed zones, we considered 1,456 zones production. We polled the production signed zones every four hours over a period spanning approximately six weeks. Some zones were only present for part of the polling period because they were added after our polling period began or because they were at some point unlinked from their parent zone, resulting in a non-production status. We included all production signed zones for which we obtained at least 10 samples.

Our analysis examined the authentication chain from each zone's `SOA` RR to the DLV trust anchor. Some of the zones were registered with the DLV at multiple points in their ancestry. For example, *isc.org* and *org* are both registered with the DLV. In such cases we optimistically selected the path with the most valid results. The results from our analysis are summarized in Table 5.1.

5.3.1 DNSSEC misconfigurations

We identified errors encountered during our polling period and classified the cause of such errors as either zone- or delegation-class misconfigurations (we exclusively used the KSK from ISC's DLV as a trust anchor, so there were no anchor-class misconfigurations in our analysis). Each misconfiguration may account for multiple errors in our data if affected subdomains are also included in the analysis. For example, a

Total zones	3,041,212
Production signed zones	1,456
Zone-class errors resulting in failure potential 1.0	134
Delegation-class errors resulting in failure potential 1.0	60
Errors resulting from misconfigured ancestor zones	61 (31%)

Table 5.1: Statistics for the DNS data collected for our analysis.

single misconfiguration in the *org* zone would result in errors for *org* as well as any affected subdomains also in our data. To account for any errors in our polling which might skew our data, we required errors resulting in complete validation failure to be detected in two consecutive samples. Over the polling period we detected 134 zone-class misconfigurations and 60 delegation-class misconfigurations that caused certain failure (i.e., had a failure potential of 1.0). Of these misconfigurations, 31% were caused by misconfigurations in a zone's hierarchy, and would have still been resolvable had a soft anchoring system been in use by resolvers, such as that proposed in Section 5.2 of this chapter.

For each zone we averaged the failure potential calculated at each poll during our analysis period. The resulting average for each production signed zone is displayed as a cumulative density function (CDF) in Figure 5.3. For 88% of the production signed zones, the failure potential averaged 0 or nearly 0. However, 4% of the production signed zones averaged a failure potential of over 0.4. Surprisingly, about 2% of the production signed zones were completely unresolvable due to misconfigurations that persisted through every poll we made to them.

Our analysis of failure potential considers only production signed zones whose ancestry is linked to the ISC DLV; this doesn't account for unsigned zones that may be affected by misconfigurations of signed zones at higher levels. For example, the expiration of `RRSIG`s in the *arpa* zone occurred during our analysis period and affected the authenticity of its `NSEC` RRs for insecure delegations, such as *in-addr.arpa*, which resulted in bogus responses for *in-addr.arpa*. However, such were not included in our analysis. We also analyzed insecure delegation by signed zones by testing for proper use of `NSEC` RRs by authoritative servers. We identified 36 signed zones for which one

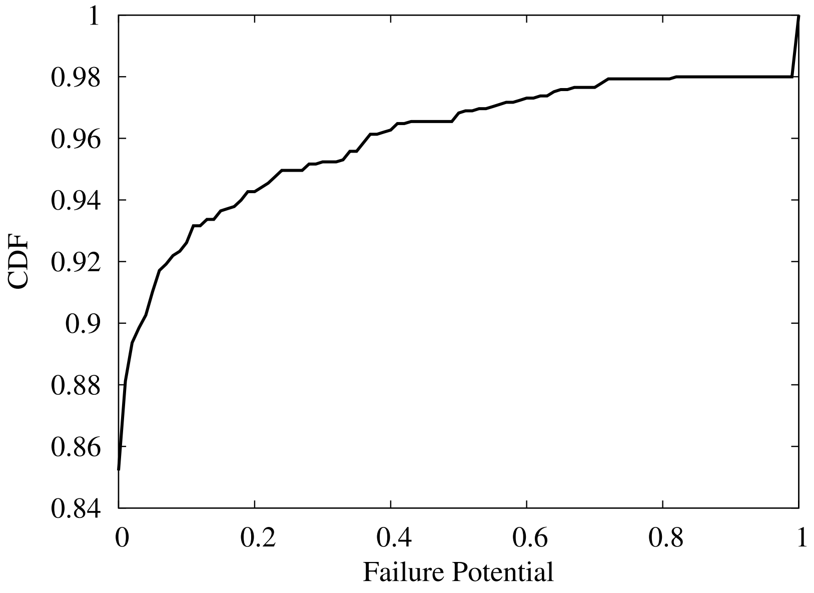

Figure 5.3: The CDF describing the average failure potential for production signed zones during our analysis period.

or more servers failed to return NSEC responses for authenticated denial of existence.

In an analysis prior to our polling period we discovered problems with insecure delegation from the *pt* country-code TLD for Portugal, which was signed and registered with DLV. The problem was that the *pt* zone was signed with NSEC3, yet two of the seven servers authoritative for *pt* were running a DNS server implementation that didn't support NSEC3. This scenario resulted in a failure potential of 0.29 for any subdomain of *pt* for which the immediate delegation from *pt* was insecure. We would anticipate this domain having a significant number of insecure delegations that were affected. The zone administrators have since fixed this issue.

5.3.2 Complexity analysis

We calculated the hierarchical reduction potential (HRP) for the complete set of zones and our subset of production signed zones. We used the RNAME suffix as a factor for comparison, as discussed in Section 5.1.2. The CDF is shown in Figure 5.4. The stair-step appearance is due to common values produced by typical zone depth and

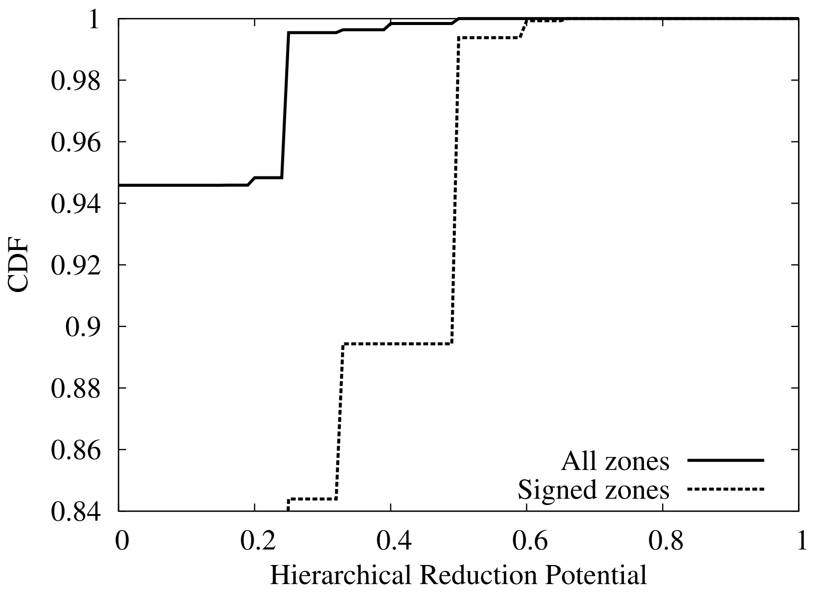

Figure 5.4: The hierarchical reduction potential of all sample zones and the subset of production signed zones.

reduction values (e.g., 1 or 2). In the general zone analysis only about 5% have the possibility for reduction, based on our criteria. However, in the production signed zones 16% might be simplified hierarchically to reduce the potential for failure.

We analyzed the administrative complexity for each zone in our set and each production signed zone. Again we used the `RNAME` field for each server name $u \in NS_z$ authoritative for zone z, as described in Section 5.1.2. The results of our analysis of administrative complexity with $n = 2$ is represented as a CDF in Figure 5.5. The graph shows the administrative complexity for the entire set of sample zones, and the subset of production signed zones. The large vertical jump at 0.5 reflects a common case where just two servers are authoritative for a zone, each administered by a different organization. In such cases it is equally likely to query a server administered by either of the two organizations and hence have a different view of zone data, depending on configuration and data consistency.

Nearly 90% of all zones examined have no administrative complexity, which by definition means that administrative responsibility for the servers authoritative for each zone is handled by a single organization. However, the subset of production

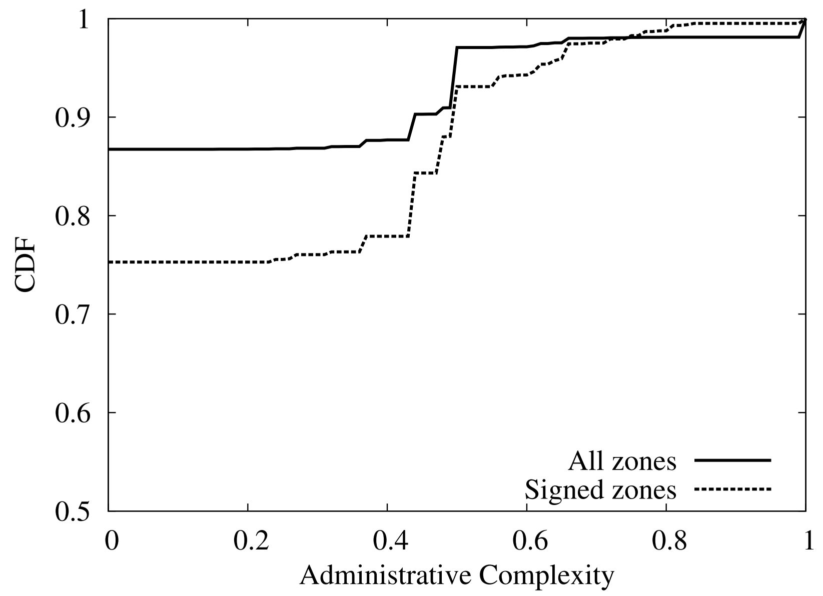

Figure 5.5: The administrative complexity of all sample zones and the subset of production signed zones.

signed zones varies from this trend. Only 75% of these zones have an administrative complexity of 0, and almost 10% have a complexity greater than 0.5.

5.4 Previous work

Other studies have been performed to quantify DNSSEC deployment in terms of pervasiveness, availability, and quality. One such project, run by IKS Jena (Information, Communication, and Systems) [37], maintains an ongoing status of DNSSEC signed zones on their Web site. SecSpider [42, 38], another DNSSEC project, discovers signed zones through several means: as discovered by a Web search engine; by user submission to the SecSpider Web interface; and by traversing `NSEC` RRs on known signed domains. It has for several years polled these zones from different world-wide locations, verifying consistency of results from different vantage points. Zone data collected from SecSpider was used perform an assessment of availability, verifiability, and validity of DNSSEC deployments [43].

Our objectives are similar to previous analyses, but our approach differs. In this work we focus on consistency of behavior across authoritative servers from a particular vantage point (under the assumption that our path is reliable), rather than consistency of experience querying a zone from different client locations. SecSpider declares a zone DNSSEC-enabled only if all authoritative servers for a zone serve DNSSEC-type RRs [42]. However, our analysis is based on actual resolver behavior when configured with a valid trust anchor: unless an chain of trust extends from an anchor at the resolver, DNSSEC deployments are irrelevant from the perspective of a resolver.

Other solutions for trust anchor distribution have been proposed, although having a different overall objective. SecSpider, and its sister project, Vantages [44, 45], collect `DNSKEY`s through polling and collaborative sharing over peer-to-peer networks. DLV services, provided by ISC and others [40, 38, 37], provide resolvers access to trust anchor repositories (TARs), such as the Interim TAR maintained by the Internet Assigned Numbers Authority (IANA) [46]. Their collective design is to assist with reliable distribution of trust anchors for zones that would otherwise remain islands of security. However, the objective and methodology of our soft anchoring approach in Section 5.2 is to strengthen chains of trust that already exist for zones that have "opted in" to DNSSEC validation through some sort of registration with parent zone (i.e., via `DS` RRs) or TAR.

5.5 Summary

The DNS is an essential component of the Internet's architecture. DNSSEC deployment is under way to protect its integrity, but the additional complexity of DNSSEC challenges the availability of DNS. Various DNSSEC-related server and zone misconfigurations can affect not only the corresponding zones, but also the entire namespace below.

In this chapter we presented an availability model for domains which have deployed DNSSEC, using the potential for validation failure as a metric. Additionally, we proposed two metrics which quantify complexity for DNS configurations and increase the potential for validation failure: hierarchical reduction potential (HRP) and administrative complexity. We also proposed a system soft anchoring to maintain availability in the presence of misconfiguration. We identified a number of errors plaguing production signed zones and determined that 31% of these errors might have been mitigated by the soft anchoring system we described. We also observed a higher HRP and administrative complexity in production signed zones than in the larger set of zones.

As the deployment of DNSSEC continues, a thorough understanding of the protocol accompanied by proper configuration will allow it to be successful. A system such as soft anchoring will mitigate the impact of misconfigurations resulting from its early deployment. In addition, administrators should be acquainted with the impact associated with misconfigurations.

Chapter 6

Conclusions and Future Work

In this dissertation we have analyzed name and server dependencies in the DNS and introduced metrics to quantify their impact on influence and availability of domain names. We summarize our conclusions and elaborate on future research in this final chapter.

6.1 Conclusions

In Chapter 3 we reviewed DNS protocol specification and implementation details to model name dependencies in the DNS. We identified the trusted computing base (TCB) of a domain name as the set of other names or administering organizations which collectively influence resolution of the name. Using our name resolution model we presented methodology to quantify the level of influence of individual domain names using a value between 0 and 1. As a representative measure we introduced third-party influence as the collective probability that resolution of a domain name is influenced by names not explicitly configured by the zone administrator. We found that on average 92% of influential domains are explicitly configured by administrators. This average shows that zone administrators are generally in control of the namespace affecting resolution of their domain names, more so than suggested by previous research. However, when resolvers use server addresses obtained from authoritative sources, rather than glue records, to query authoritative servers, the average size of the TCB increases, and so does third-party influence. We also observed that the practice of chaining domain name aliases increases third-party influence in production DNS zones.

We used name and server dependencies as a basis for our availability model in Chapter 4. Using this model domain name availability was quantified using two metrics: minimum servers queried (MSQ) and redundancy. Common misconfigurations, such as inconsistent `NS` RRsets between parent and child zones, affect both of these metrics and ultimately decrease availability of a domain name. We observed that 6.7% of domain names had sub-optimal MSQ values, and 14% experienced redundancy less than that with which administrators explicitly configured. Good configuration practices for maintaining availability include proper synchronization of `NS`

RRsets in parent and child zones and appropriate inclusion of glue records.

We examined DNS availability from a DNSSEC perspective in Chapter 5, using DNSSEC misconfigurations and server inconsistencies as the basis for determining the potential for validation failure. We also introduced complexity metrics which can contribute to increased failure potential: hierarchical reduction potential (HRP) and administrative complexity. The former is a measure of the extent to which zones under a single administrative entity might reasonably be collapsed, resulting in less room for errors caused by maintenance of additional zones or parent/child zone coordination. The latter describes the diversity of organizations administering authoritative servers for a zone. We observed that 12% of the production signed zones we polled averaged a non-zero failure potential over the polling period. Our analysis also shows that the ancestry of 5% of the zones we analyzed might be reduced to minimize hierarchical complexity, and 10% have non-zero administrative complexity.

As the Internet grows and more applications rely on its framework, the DNS will continue to play an integral role in usability. The models and metrics presented in this dissertation can assist DNS administrators in better understanding their DNS deployments and avoiding name resolution failure by properly engineering and maintaining their DNS infrastructures. DNS attacks can be mitigated through security solutions, such as DNSSEC, but availability hinges on proper application and maintenance.

6.2 Future work

The complexities of the DNS and DNSSEC protocols and the novelty of DNSSEC deployment leave a number of open issues to better understand and maintain high availability for DNS name resolution. We discuss in this section several expansions of this work that would contribute to DNS robustness.

6.2.1 Availability analysis

Our analysis can be furthered to more accurately model DNS availability. The availability model from Chapter 4 is based solely on name and server dependencies, and Chapter 5 views availability only from a vantage point of DNSSEC validation failure. Integrating the two models would result in a more comprehensive view of DNS availability, including both DNSSEC and non-DNSSEC dependencies. For example, if an `NS` target for a zone is not in-bailiwick, then a resolver must resolve the name and validate the response to obtain the address, which it can then use for query. In our analysis, we've only considered the case where validation of the response either succeeds or fails altogether (i.e., failure potential is either 0 or 1). However, a failure potential between 0 and 1 may affect the validation of the `NS` target, complicating the availability of the zone for which the server is authoritative.

Also, our analysis of failure potential induced by DNSSEC misconfiguration did not consider alias (i.e., `CNAME`) targets, which must also be resolved and validated for name resolution. When *www.example.com* is an alias for *www.example.net*, its resolution is completely dependent on resolution of *www.example.net*, and is also subject to its failure potential.

The consideration of TTL values in the DNS availability model will provide a temporal view of name resolution availability, as pertinent data expires from resolver caches. The value added with this view is an understanding of the persistence of diminished availability caused by DNS or DNSSEC misconfiguration. For example, an `NS` RRset containing targets with validation issues may linger in caches, even after the authoritative `NS` RRset has been updated to include other targets, which properly validate. Administrators might better estimate the worst case duration of potential availability issues with this extension to the availability model.

A similar circumstance is the caching of bogus data by validating resolvers. RFC 4035 indicates that resolvers may cache bogus data, with the following caveats: the resolver itself must provide a small TTL value to the bogus data to mitigate a denial-of-service attack (since the TTL provided by the bogus data is untrustworthy); and to prevent transient failures, a resolver should only answer from its "bad cache" after a configurable threshold of failures for a given RRset has been exceeded [7]. The effect of the TTL value assigned to bogus data in cache and the failure threshold for caching would provide an interesting perspective to the temporal DNS availability model. Optimal values would minimize overhead while maximizing availability diminished by transient failures.

6.2.2 Operational standards

The DNSSEC-related misconfigurations described in Chapter 5 can be avoided by proper development of and adherence to operational standards. RFC 4641 [25] is one of the first such standardized documents for operational guidelines by zone administrators. Deployment experience since the initial release of this RFC has exposed new configurations, and new procedural concerns. For example, RFC 4641 assumes a ZSK/KSK setup, while in actuality some DNS administrators prefer to use a design involving only a single signing key. Also, stronger cryptographic algorithms have since been introduced for signing zone data, such as RSA with SHA-2, which is "widely believed to be more resilient to attack than SHA-1" [47, 48]. However, upgrading DNSSEC algorithms is a non-trivial procedure not previously covered by RFC 4641.

At the time of this writing, a second revision of RFC 4641 is being drafted as an updated operational document. While the updated procedures and text will be more comprehensive, the procedures are nonetheless complex. Various zone management tools have been developed to ease operational compliance [49], but because operational procedures are still being standardized, consistent compliance is a challenge.

Although specific operational procedures are helpful, the coordination between parent and child zone required for seamless KSK rollovers remains an area prone to errors. Nearly one third of the validation errors we encountered in our analysis were the result of unsuccessful KSK rollovers. A protocol that might ease this operational gap is an in-band SEP key rollover between parent and child zones, comparable to trust anchor rollovers specified in RFC 5011 [35]. The advantage of this protocol is that it would require communication only between two entities, the parent and child zones, as opposed to a potentially large number of individually operated resolvers.

6.2.3 Mitigation techniques

Several techniques can be employed by resolvers to mitigate availability issues caused by validation failures. Validation diligence (i.e., successively querying each authoritative server until a valid answer is received) has been implemented in some resolver implementations, as mentioned in Chapter 5. This technique only improves availability when a validating resolver is able to query authoritative servers directly (i.e., is not behind a proxy resolver), and when failure potential is between 0 and 1. The trust model presented in Chapter 5 may be foundational for an deriving an algorithm that is optimally efficient for such diligence when a resolver encounters missing or invalid data. This would mitigate availability issues without inhibiting performance, as was observed in early versions of BIND validating resolvers [50].

The soft anchoring proposal in Chapter 5 is an additional technique that could be implemented by administrators of DNS resolvers to mitigate the effects of DNSSEC misconfiguration. It validation problems for zones whose signed ancestor zones are misconfigured, but only if the zones were previously authenticated by the resolver. We intend to formalize the idea into an Internet draft to gain support from the global DNS community in the hope that it might alleviate growing pains with early DNSSEC deployment.

6.2.4 Preemptive diagnostic advisory

Much of this dissertation has focused on analyzing configurations of DNS deployments and improving or mitigating existing availability issues caused by misconfiguration. Another approach is to preemptively detect and flag configuration changes that might decrease the robustness or security posture of a domain name, before the changes become production. A preemptive diagnostic advisory tool would consider several properties and metrics from models presented in this research in relation to the domain name whose zone is being modified:

- *Third-party zones or organizations* in the name's TCB. Changes to RRs, such as the zone's `NS` RRset or a `CNAME` RR, may introduce third-party zones (i.e., influential zones not explicitly configured by zone administrators) that aren't immediately apparent to zone administrators. Third-party zones might be compared against a policy that restricts the name's TCB to zones known to be trustworthy, or explicitly prohibits zones known to be untrustworthy. The reasoning might be to avoid dependencies that are known to be unreliable, vulnerable, adversarial or competitive. A diagnostic tool warn an administrator if zones in the TCB violate policy.

- *Third-party influence.* Zone administrators may wish to upper bound for an acceptable level of third-party influence on the name. For example, setting a threshold of 0.5 would alarm administrators when in the proposed setup there is a 0.5 probability that resolution of the name is influenced by third party zones or organizations.

- *MSQ.* A change resulting in a sub-optimal MSQ for the name indicates that more prerequisite lookups than what might be possible are required for resolution of the name. This may not be problematic, but in such a case, bringing the servers comprising the MSQ to the attention of the administrator would allow him the opportunity to assess the situation. Such a result might be attributed to a misconfiguration, such as a missing glue record.

- *Redundancy.* If a change results in false redundancy for the name, the cause may be misconfiguration, such as a missing glue record or multiple targets in the `NS` RRset resolving to the same address. It might also be due to a smaller level of redundancy in a downstream dependency than that being introduced for the zone in question.

- *Validation failure potential.* A non-zero failure potential of the name, caused by misconfiguration in the zone itself or one of its dependencies, is problematic for validating resolvers. This might be due to changes that break the chain of trust or that result in failed validation of required dependencies.

Because the purpose of such a preemptive advisory tool is to raise a flag to administrators that a change might compromise availability or security of a domain name, it is only effective if the alert comes before the change propagates to production DNS. A natural extension of this is the ability for an administrator to run several scenarios through the tool to determine the solution that best meets the local policy. For example, a DNS administrator for *example.com* might have several options for servers which might host its authoritative data for high availability, each administered by a different organization. While the *example.org* and *example.net* namespaces may both be trustworthy, an analysis by the advisory tool may indicate that *example.net* is influenced by *untrustworthy.net*, so using an *example.org* name server is the safer choice for the DNS administrator.

A tool for preemptive diagnostic advisory may sufficiently discourage administrators from implementing changes that decrease the availability of their DNS setup. However, changes made to downstream dependencies may be deployed independently and without the knowledge of the administrator of a dependent name. Such changes may affect the properties of availability observed by a DNS administrator after having initially received no warnings from a diagnostic tool. Additionally, with a DNSSEC deployment availability may be degraded over time, without intervention, by the expiration of `RRSIG`s. We propose the following two extensions to diagnostic advisory that might be used in addition to preemptive checks on a domain name:

- *Reverse dependency analysis.* While it is impossible to know the extent of all reverse dependencies of a zone, a reverse dependency analysis might be conducted using as much information as is available. Such an analysis would be performed in addition to that of forward dependencies before a change is introduced, so administrators can be alerted of how the proposed deployment might affect the availability of dependent names.

- *Periodic diagnostic polling.* Periodic re-assessment of a DNS setup would allow a DNS administrator to monitor changes to availability properties, caused either by changes made to downstream dependencies or expiring `RRSIG` RRs.

The incorporation of these assessments into a preemptive and polling diagnostic tool, such as that described in this section, may give administrators confidence in making changes to working DNS deployments and assurance that availability is maintained over time.

Bibliography

[1] 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 of the 4th ACM SIGCOMM conference on Internet measurement.* New York, NY, USA: ACM, 2004, pp. 1–14.

[2] V. Ramasubramanian and E. G. Sirer, "Perils of transitive trust in the domain name system," in *IMC '05: Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement*, USENIX Association. Berkeley, CA, USA: USENIX Association, 2005, pp. 379–384.

[3] C. Deccio, C.-C. Chen, J. Sedayao, K. Kant, and P. Mohapatra, "Quality of name resolution in the domain name system," in *Network Protocols, 2009. ICNP 2009. 17th IEEE International Conference on*, October 2009, pp. 113 – 122.

[4] V. Pappas, Z. Xu, S. Lu, D. Massey, A. Terzis, and L. Zhang, "Impact of configuration errors on DNS robustness," in *SIGCOMM '04: Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, ACM. New York, NY, USA: ACM, 2004, pp. 319–330.

[5] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose, "RFC 4033: DNS security introduction and requirements," 2005.

[6] ——, "RFC 4034: Resource records for the DNS security extensions," 2005.

[7] ——, "RFC 4035: Protocol modifications for the DNS security extensions," 2005.

[8] R. Chandramouli and S. Rose, "Open issues in secure DNS deployment," *Security & Privacy, IEEE*, vol. 7, pp. 29 – 35, September – October 2009.

[9] K. Lambert, O. Souleymane, K. Tiemoman, A. Brice, and T. Pierre, "Deployment of DNSSEC: Problems and outlines," in *Research Challenges in Information Science, 2008. RCIS 2008. Second International Conference on*, June 2008, pp. 343 – 348.

[10] C. Deccio, J. Sedayao, K. Kant, and P. Mohapatra, "Measuring availability in the domain name system," in *INFOCOM, 2010 Proceedings IEEE*, March 2010, pp. 1 – 5.

[11] P. Mockapetris, "RFC 1034: Domain names - concepts and facilities," 1987.

[12] ——, "RFC 1035: domain names - implementation and specification," 1987.

[13] D. Barr, "RFC 1912 - common DNS operational and configuration errors," 1996.

[14] D. Dagon, N. Provos, C. P. Lee, and W. Lee, "Corrupted DNS resolution paths," in *Proceedings of Network and Distributed Security Symposium (NDSS '08)*, 2008.

[15] D. Dagon, M. Antonakakis, K. Day, X. Luo, C. P. Lee, , and W. Lee, "Recursive DNS architectures and vulnerability implications," in *Proceedings of Network and Distributed Security Symposium (NDSS '09)*, 2009.

[16] US-CERT, "Vulnerability note VU#457875: Various DNS service implementations generate multiple simultaneous queries for the same resource record."

[17] ——, "Vulnerability note VU#800113: Multiple DNS implementations vulnerable to cache poisoning."

[18] "The birthday problem." [Online]. Available: http://en.wikipedia.org/wiki/Birthday_problem

[19] DNSCurve. [Online]. Available: http://dnscurve.org/

[20] D. Dagon, M. Antonakakis, P. Vixie, T. Jinmei, and W. Lee, "Increased DNS forgery resistance through 0x20-bit encoding: security via leet queries," in *Proceedings of the 15th ACM conference on Computer and communications security.* ACM, 2008, pp. 211 – 222.

[21] R. Perdisci, M. Antonakakis, X. Luo, and W. Lee, "WSEC DNS: Protecting recursive DNS resolvers from poisoning attack," in *Dependable Systems & Networks, 2009. DSN '09. IEEE/IFIP International Conference on.* IEEE, July 2009, pp. 3 – 12.

[22] V. Ramasubramanian and E. G. Sirer, "The design and implementation of a next generation name service for the internet," in *SIGCOMM '04: Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications.* New York, NY, USA: ACM, 2004, pp. 331–342.

[23] L. Yuan, K. Kant, P. Mohapatra, and C.-N. Chuah, "DoX: A peer-to-peer antidote for DNS cache poisoning attacks," in *ICC '06: IEEE International Conference on Communications*, vol. 5. IEEE, June 2006, pp. 2345–2350.

[24] B. Laurie, G. Sisson, R. Arends, and D. Blacka, "RFC 5155: DNS security (DNSSEC) hashed authenticated denial of existence," 2008.

[25] O. Kolkman and R. Gieben, "RFC 4641: DNSSEC operational practices," 2006.

[26] ISC BIND. [Online]. Available: http://www.isc.org/products/BIND/

[27] djbdns. [Online]. Available: http://cr.yp.to/djbdns.html

[28] R. Elz and R. Bush, "RFC 2181 - clarifications to the DNS specification," 1997.

[29] Open Directory Project. [Online]. Available: http://www.dmoz.org/

[30] SC08: The International Conference for High-performance Computing, Networking, Storage and Analysis. [Online]. Available: http://sc08.supercomputing.org/

[31] PlanetLab. [Online]. Available: http://www.planet-lab.org/

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

[33] “The measurement factory.” [Online]. Available: http://dns.measurement-factory.com/surveys/

[34] R. Liston, S. Srinivasan, and E. Zegura, “Diversity in DNS performance measures,” in *Proceedings of the SIGCOMM ‘02 Symposium on Communications Architectures and Protocols.* New York, NY, USA: ACM, 2002, pp. 19–31.

[35] M. StJohns, “RFC 5011: Automated updates of DNS security (DNSSEC) trust anchors,” 2007.

[36] Sandia National Laboratories, “DNSViz: A DNS visualization tool.” [Online]. Available: http://dnsviz.net/

[37] IKS, “DNSSEC.” [Online]. Available: https://www.iks-jena.de/leistungen/dnssec.php

[38] “SecSpider.” [Online]. Available: http://secspider.cs.ucla.edu/

[39] National Institute of Standards and Technology, “Secure naming infrastructure pilot.” [Online]. Available: http://www.dnsops.gov/

[40] Internet Systems Consortium, “DNSSEC look-aside validation registry.” [Online]. Available: https://dlv.isc.org/

[41] S. Weiler, “RFC 5074: DNSSEC lookaside validation,” 2007.

[42] E. Osterweil, D. Massey, and L. Zhang, “Deploying and monitoring DNS security (DNSSEC),” in *25th Annual Computer Security Applications Conference (ACSAC ’09)*, December 2009.

[43] E. Osterweil, M. Ryan, D. Massey, and L. Zhang, “Quantifying the operational status of the DNSSEC deployment,” in *Proceedings of the 6th ACM/USENIX Internet Measurement Conference (IMC’08)*, October 2008.

[44] E. Osterweil and L. Zhang, "Interadministrative challenges in managing DNSKEYs," *Security and Privacy Magazine: Securing the Domain Name System*, September 2009.

[45] "Vantages." [Online]. Available: http://www.vantage-points.org/

[46] Internet Assigned Numbers Authority, "Interim trust anchor repository." [Online]. Available: https://itar.iana.org/

[47] J. Jansen, "RFC 5702: Use of SHA-2 algorithms with RSA in DNSKEY and RRSIG resource records for DNSSEC," 2009.

[48] D. E. 3rd, "RFC 3110: RSA/SHA-1 SIGs and RSA KEYs in the domain name system (DNS)," 2001.

[49] A. Nilsson, "A review of administrative tools for DNSSEC—spring 2010," 2010. [Online]. Available: http://www.iis.se/docs/DNSSEC-Admin-tools-review-Final.pdf

[50] G. Michaelson, P. Wallström, R. Arends, and G. Huston, "Roll over and die?" [Online]. Available: http://www.potaroo.net/ispcol/2010-02/rollover.html