A New Look at the Old Domain Name System

Yair Amir[1], Daniel Massey[2], Ciprian Tutu[1] *

Technical Report
CNDS-2003-2
http://www.cnds.jhu.edu

July 18, 2003

Abstract

The Domain Name System (DNS) is undergoing fundamental changes in both design and operations, but these changes are mostly taking place in piecemeal extensions. In this paper we consider how to maintain a simple and robust DNS in the face of these inevitable and essential changes. We consider some of the features that the modern DNS is trying to incorporate and we look at the ensuing problems from a systematic perspective. We identify some key architectural issues and design principles that we believe are essential to the successful integration of such features in the DNS infrastructure. Following these principles, we sketch two possible mechanisms that would improve the availability and timeliness, and decouple the zone management from the query/response data-path of DNS while being deployable in parallel and incrementally with respect to the existing infrastructure.

1 Introduction

The Domain Name System (DNS)[11] provides the essential service of translating names into IP addresses. Virtually all user-level internet services (and many system-level services) rely on the ability to access resources through their Internet name. Typically, a name needs to be translated into an IP address before any packets can be routed to the appropriate destination. In addition to mapping names to IP addresses, the DNS maps IP addresses to names, stores mail exchange (MX) records needed for routing email, and includes a growing list of other mappings.

The DNS can be viewed as the equivalent of a powerswitch for the Internet in the sense that DNS problems can greatly diminish Internet service availability up to the point where it can paralyze the Internet for significant periods of time. If the DNS fails to provide an answer, there is often little that an application can do to recover and even a delay in obtaining the DNS data can result in a perception of poor Internet performance. For example, if the DNS fails to provide an IP address for www.cnds.jhu.edu, there is little the browser can do and if the DNS is slow in mapping the www.cnds.jhu.edu name, the user may perceive there is a problem with the server. Tools such as dsniff [10] use false DNS response as a first step in forging web pages, intercepting email, launching man-in-the-middle attacks against login sessions and so forth. [7, 6] provide a detailed overview of existing DNS vulnerabilities.

The DNS was designed nearly two decades ago and its tremendous success is a credit to the original design. However, the original design focused on addressing simple *fail-stop* faults, such as the loss of a server. The limitations of the fail-stop fault model have been noted in a variety of contexts[4] as the DNS is vulnerable to a wide range of more complex faults and intentional attacks. The addition of Dynamic DNS[15] dramatically changes both the nature of DNS data and the set of entities that manage the DNS. The success

*[1]Johns Hopkins University. Email {yairamir, ciprian}@cnds.jhu.edu [2]USC/ISI. Email: masseyd@isi.edu

of DNS has made it a tempting solution for storing all forms of distributed data and the need for a resilient DNS has only grown in importance. At the same time, changes in the Internet have placed increasing demands on the DNS. A long list of piecemeal enhancements and new services have been added to the DNS to the point where it has been suggested we are "overloading the saddle-bags on an old horse"[8].

We argue that a more systematic approach is needed in order to properly address the wide variety of issues that become evident with the current DNS infrastructure. We identify some of the key design issues shared among the various DNS changes and argue that the existing solutions are still limited in their effectiveness. We suggest two mechanisms that can be deployed incrementally and in parallel with the existing system, and address some of the problems that the DNS is confronted with.

We believe one must tackle the various issues from two perspectives: at the micro-infrastructure level one must address the management of a single zone eliminating the single point of failure and providing a building block for timely propagation of zone updates to all authoritative servers; at the macro-infrastructure level one must address the availability of the DNS system as a whole and guarantee the timely propagation of updates to end resolvers. We show these services help provide an underlying design structure that will allow the DNS to continue its tremendous success and provide a solid base design that encompasses current demands and provides the building blocks for future changes.

In the remainder of this paper, Section 2 describes the changes facing the DNS and identifies common underlying themes that a DNS design update should include. In Section 3, we show how to enhance the micro-infrastructure to meet the new design goals. In Section 4, we consider how the larger infrastructure can also be enhanced. Section 5 concludes the paper by summarizing the challenges and our proposed approach to meeting the challenges.

2 Modern DNS and Its Challenges

The DNS was originally designed as a replacement for "hosts.txt" files that stored centralized names to IP address mappings. Today, the DNS is arguably one of the most successful distributed applications. There have been several fundamental changes since the DNS was designed nearly two decades ago and, in most cases, the simple DNS design has adapted extremely well. For example, the original DNS design envisioned a deep tree structure with many levels, but market forces instead generated an extremely flat topology where nodes like ".com" have over 22 million child nodes. This model is perhaps somewhat different from the original design, but it works nevertheless. It is important to note that the DNS has also adjusted. A large zone such as ".com" is no longer managed by a single entity. Instead, a registry maintains the database and servers, and changes to the zone are passed through any one of over 100 authorized registrars. The DNS works, but the structure and operations are unlike those that were originally envisioned.

The success of the DNS has attracted more applications and added more demands. New uses for the DNS range from using DNS responses for load balancing to providing a type of PKI for IPSEC public keys[13]. Within the IETF, the registrar/registry issues have led to the formation of the Provisioning Registry Protocol working group, the Internationalized Domain Name working group examines issues related to international names, the DNS Operations working group is struggling with the process of adding IPv6 addresses to the DNS, and the DNS Extensions working group continues to be active in a number of other areas. Some results of these groups efforts include the (now experimental) A6 record that not only added IPv6 addresses, but provided a technique for fast IPv6 renumbering after a provider change. SRV records were introduced to provide pointers from the DNS to other servers such LDAP servers. Dynamic DNS changes the update model for the DNS and is implemented in most modern servers. DNS security also remains an active topic.

It can be argued that some of these "improvements" are ill-advised, but at least two changes have gained wide support in both standards communities such as IETF and have been incorporated in leading DNS server implementations [9]. These changes include improving DNS security [5] and enabling Dynamic DNS[15]. We discuss each of these in more detail and then identify common threads shared by the various DNS enhancements.

2.1 DNS Security

With the increasing number of entities managed by and depending on the DNS, security has become an obvious issue and its impact was evidentiated by several exploits which hijacked DNS entries of popular sites redirecting client queries. The DNS security extensions (DNSSEC) add authentication to the DNS and provide resolvers the opportunity to verify the validity of a response.

To support DNSSEC, operators must sign their zones using public key cryptography. Signatures need to be regenerated at periodic intervals and the existence of older data (such as public keys or signatures which should have expired) can present problems for the DNS. In addition, DNSSEC increases the need for synchronization between parent and child zones since the parent zone provides a ”secure entry point” to the child zone. DNSSEC entails a change in the DNS design, DNS operations at servers, and DNS behavior at resolvers.

However, naive deployment can actually *decrease* DNS survivability and availability. A DNSSEC query requires timely signatures and public keys. Expired signatures, added load at servers due to cryptographic burdens and increased message sizes, and increased opportunity for administrative errors add new potential for a delayed answer or lack of answer. This result in new denial of service opportunities when authentication fails or the proper records are missing from a response (sometimes caused by legitimate behaviour by old caches). As a result, resolvers that prefer DNSSEC often choose to readily accept any answer even when a secure answer should be provided. Accepting an unsigned answer is essential for availability, but such a policy eliminates most of the DNSSEC advantages and potentially introduces higher risks by creating a false sense of security.

2.2 Dynamic DNS

The addition of “dynamic” DNS also changes the underlying DNS data model. While the previous model of DNS operation often revolves around simply editing a text file, dynamic DNS allows an entity to update a DNS record on the server “on the fly”. For example, after a laptop is assigned a new IP address by a DHCP, it may send an update to the “isi.edu” DNS servers to update the ”mylaptop.isi.edu” IP address records. Dynamic DNS is supported by major DNS server implementations such as BIND [9].

Dynamic updates can create caching issues and also reveal additional problems if the updates must be sent to a single master server, since this places added load on that single point of failure. At the same time, Dynamic DNS vastly expands the number of entities administering the DNS and it is becoming more common that services are run from mobile users who require access to resources that are restricted to a certain name domain.

2.3 Underlying Challenges

The question we are faced with is not whether the DNS will be changed. The DNS is already undergoing fundamental change in both design and operations. These changes are mostly taking place in piecemeal extensions. The real question is can we maintain a simple and robust DNS in the face of desired (and some less desired) changes. In the previous section we enumerated a few of the features that the modern DNS is trying to incorporate. Looking at the ensuing problems from a systematic perspective, we identified three key architectural issues that we believe are essential to the successful integration of such features in the DNS infrastructure.

- Availability
- Timeliness of both DNS data and operations
- Decoupling DNS management from DNS query/response

Availability is the hallmark of the DNS. The DNS is successful because its data is highly available. DNS availability can be assessed in several ways. First, availability for queries is what clients(resolvers) experience directly. Normally this is enhanced by redundant servers within the DNS hierarchy and through use of caches, although caches also introduce a tradeoff with respect to the accuracy of the response. Second, availability of updates is another issue: updates are performed on a master server then distributed out to a number of slave servers. If the master is unavailable, the update cannot be performed at that time. Furthermore, the propagation of a new update is executed using pull requests by the secondary servers. A master server can send a notification message to its slaves

encouraging them to execute an early pull, but this notification message may be lost if there is no direct connectivity between the master and its slaves at that time. Server partitions are a fact of life in the Internet and the DNS must work in difficult network conditions as well as in ideal conditions.

Timeliness applies to both the freshness of the DNS data returned in response to a query and the speed at which DNS operational changes propagate. Elements such as DNS security introduce signatures with specific lifetimes. DNS resolvers may reject data with old signatures and create denial of service. To change the cryptographic keys used to sign DNS data, coordination is required among the multiple servers of a single zone as well as between a zone and its parent. The addition of Dynamic DNS increases the frequency of changes as well as increasing the expectation a change is seen quickly. Dynamic DNS requests entered at one point should quickly synchronize with other servers. The potential for multiple updates from different entities authorized to make dynamic changes introduces questions of ordering. As the DNS is used to store data other than IP addresses, these new data types introduce their own timing issues. Overall, new services add new timing issues and a robust underlying infrastructure design should provide a structure that reflects a need to manage timing issues.

Decoupling DNS management from DNS query/response becomes increasingly important as operations become more complex. A simple view of a single administrator editing a simple ”zone file” in text format and infrequently changing an IP address is rapidly becoming obsolete. DNS security adds many new operational tasks and adds more frequent communication between parent and child zones. Dynamic DNS creates automatic updates and many new sets of entities that may update portions of a zone. The addition of new data types (such as storing IPSEC public keys in the DNS) adds new management requirements related to those types. Essentially, DNS reuses its tree hierarchical structure for authority delegation, zone management, and serving client requests. Caches are one step that decouples the query/response data path from the management hierarchy. However, stronger decoupling needs to be achieved if the DNS is to become an efficient infrastructure for the variety of new uses.

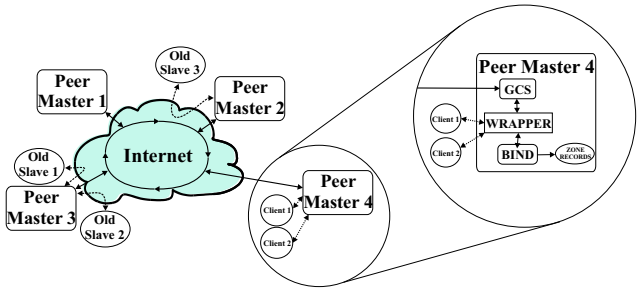

Figure 1. DNS Peer Zone Management

A practical implementation requirement is that any design change *must be incrementally deployable* in the current system. The DNS is an essential infrastructure and any change will only occur gradually. At all times, a new design or extension must assume some servers and resolvers will continue to use the current DNS. This is true of both existing work such as DNSSEC or Dynamic DNS and applies equally well to any future change. In addition, a solid solution should add benefits for those who have deployed the change, regardless of how widespread the solution is adopted. In the following sections we present some initial work on the design of two mechanisms that attempt to address the key issues listed above.

3 Peer Zone Management

At the micro-infrastructure level, we consider a design where the DNS servers responsible for a given zone act as peer servers instead of using a master-slave model. Zone updates, including dynamic updates, can be submitted to any of the servers and the updates are propagated immediately to all the peers using a group communication infrastructure. In case the set of peers becomes partitioned, updates can be performed in either of the disconnected components. When two disconnected components remerge, the updates performed in each partition are combined and all the servers converge to the same state of the zone records. The reconciliation process relies on the fact that the DNS zone-updates exhibit a commutative semantics. Every update is disseminated to all servers and each server can decide, based on the logical timestamp associated with each update, what is the most current value for each DNS resource record.

To validate our model, we have implemented a wrapper around one of the most widely used DNS

servers (BIND). The wrappers communicate using the Spread [14, 3] group communication system which provides efficient message delivery guarantees and membership notifications in a wide-area network setup. Each wrapper joins the same group associated with the zone that is managed. Clients that attempt to perform dynamic updates can connect to any wrapper, thinking that they are talking directly to the DNS server. The wrapper that receives an update sends it to the local DNS server and also to the group of wrappers, together with a logical timestamp. Upon receiving an update from the group, each wrapper checks the attached timestamp against the last recorded timestamp for the given record and submits the update to the bind daemon if it is more recent. The algorithm employed is very similar to the one described in [1]. Figure 1 illustrates the resulting zone management architecture. Note that resolver queries continue to be serviced transparently as the wrapper just forwards them to the DNS server and propagates the response back to the client without any additional interference. We have deployed the wrapper in some of our local DNS zones.

Additionally, the peer servers can coexist (see figure 1) in a zone with standard slave servers which will continue to recognize only one of the peers as the master for the zone and will be updated by regularly polling the master for changes. This allows for easy deployment of the new solution since DNS best practices recommend that some slaves for a given zone are under different management and this management may not adhere to the new system. The deployment of the peer zone management system for a given zone will provide instant benefit to the organization using it, regardless of the method employed by its parent zones.

The peer architecture reduces the strict dependency of the entire zone management on a single master server. Peer zone administrators can submit changes to any peer server even when disconnected from the previously designated "master" and can thus increase the *availability* of the service, since employing the group communication membership service combined with a distributed data replication algorithm creates an infrastructure resilient to network partitions and remerges as well as server crashes and recovery. The peer architecture also enhances the *timeliness* of propagating updates to the zone servers. This new architecture makes no change to the DNS query/response behavior of the zone, but does change the DNS management model within the zone thus benefiting from the ability to decouple DNS management from DNS query/response.

4 DNS Superchaches

Addressing zone management at the micro-infrastructure level is only a building block towards addressing the issues mentioned in section 2.3. Addressing the wider range of problems is more challenging and requires a different global approach. A potential macro-level approach considers a set of servers that will cache the information from the root and top level domain (TLD) portions of the DNS hierarchy. This includes the name server information needed to reach the top level domains and the information needed to reach second level domains such as *.com (but does include all resource records from within the second level domains).[1] We call these servers "supercaches". The supercaches are equal peers and they redundantly store the same information. Our preliminary work is exploring supercaches on the order of a few hundreds, up to a thousand, servers, that cache data for tens of millions of zones.

A local nameserver attempting to resolve a request will check its local cache and then consult one of the supercaches instead of going to the root servers (or other top level servers). The supercache provides the necessary referrals into the appropriate lower level zone and also provides the necessary DNS security data that currently might require additional queries to the top level servers. The supercaches can run on top of an overlay network to ensure the correct and efficient data replication. The nameservers for the TLD's and secondary level domains should inform one of the supercaches of the zone update. If the zone is maintained with a peer system, one server from the zone will be in charge of updating the supercache. To support the nameservers that do not implement the peer zone management, the supercaches divide among themselves the space of DNS names that they are caching in a redundant manner, so that each name is covered by several supercaches. Each supercache will be responsible for monitoring its subset of the namespace by regularly polling each nameserver at

[1] Specifically, we include NS, DS, NXT, and KEY records along with their associated SIGs and any glue A records

intervals smaller than the recommended ttl in order to help faster propagation of the updates.

There are a number of concerns that need to be carefully addressed by such a design. From a systems perspective, scalability, populating the caches, timeliness of the update propagation are issues that can be addressed using scalable overlay network techniques [12, 2] and efficient distributed algorithms. From a security perspective several problems need to be addressed as well. First, the communication protocol between the zone managers (either a standard master or a peer) has to be authenticated and it must also take care to ensure the most recent data is transmitted. Second, the data propagation among the supercaches needs to prevent the possibility of a bad supercache poisoning the other supercaches with bad or stale data. Finally, the communication between client resolvers and supercaches also has to be protected against stale/replayed responses. Our preliminary analysis suggests all these aspects can be addressed by leveraging from the existing DNSSEC infrastructure.

Through usage of the supercache layer, we are essentially *decoupling* the mechanism for serving queries (supercache) from the mechanism used for management delegation (the tree hierarchy). The suggested architecture also presents much better *timeliness* guarantees since a few steps will be skipped in moving down the DNS tree hierarchy. In addition, a number of DNSSEC related steps can be combined efficiently. The timeliness guarantees can be further tuned depending on the requirements of each zone and depending on the data types that are stored in the zones. Furthermore, the many supercaches can provide a more reliable setup as they are more readily *available* than the root servers.

5 Conclusions

We have shown that the DNS is undergoing fundamental changes in both design and operations. In some cases, these changes are essential for the DNS to continue its critical role as the backbone of the Internet. We identified key architectural issues and design principles that we believe are essential to both the successful integration of these immediate changes and to provide a strong underlying infrastructure that would readily support future demands. We argue that the future DNS needs to provide increased availability and timeliness guarantees. In addition, the design is strengthened by decoupling the zone management and the query/response mechanisms. For practical reasons, we also note that any design change needs to be incrementally and even parallelly deployable with the existing infrastructure. We introduced two possible design alternatives that follow the principles identified above.

References

[1] Ofir Amir, Yair Amir, and Danny Dolev. A highly available application in the Transis environment. In *Hardware and Software Architectures for Fault Tolerance*, pages 125–139, 1993.

[2] Y. Amir and C. Danilov. Reliable communication in overlay networks. In *Proceedings of the IEEE International Conference on Dependable Systems and Networks*, June 2003.

[3] Y. Amir and J. Stanton. The Spread wide area group communication system. Technical Report CNDS 98-4, Johns Hopkins University, Center for Networking and Distributed Systems, 1998.

[4] T. Anderson, S. Shenker, I. Stoica, and D. Wetherall. Design guidelines for robust internet protocols. In *Proceedings of HotNets-I*, pages 125–130, Ocotber 2002.

[5] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. DNS security introduction and requirements. *Work in progress: draft-ietf-dnsext-dnssec-intro-05*, February 2003.

[6] D. Atkins and R. Austein. Threat analysis of the domain name system. *Work in progress: draft-ietf-dnsext-dns-threats-03*, June 2003.

[7] Steven M. Bellovin. Using the domain name system for system break-ins. In *Proceedings of the fifth Usenix UNIX Security Symposium*, pages 199–208, Salt Lake City, UT, Jun 1995.

[8] Randy Bush. The DNS today: Are we overloading the saddlebags on an old horse?". *49th IETF Meeting Plenary Presentation*, December 2000.

[9] BIND Berkeley Internet Name Domain. http://www.isc.org/products/bind/.

[10] dsniff. http://www.monkey.org/ dugsong/dsniff/.

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

[12] The Spines Overlay Network. http://www.spines.org.

[13] M. Richardson. A method for storing ipsec keying material in dns. *Work in progress: draft-ietf-ipseckey-rr-05*, June 2003.

[14] The Spread Toolkit. http://www.spread.org.

[15] P. Vixie, S. Thomson, Y. Rekhter, and J Bound. Dynamic updates in the domain name system. *RFC 2136*, April 1997.