Internet Architecture Board (IAB)
Request for Comments: 6950
Category: Informational
ISSN: 2070-1721

J. Peterson
NeuStar, Inc.
O. Kolkman
NLnet Labs
H. Tschofenig
Nokia Siemens Networks
B. Aboba
Skype

October 2013

Architectural Considerations on Application Features in the DNS

Abstract

A number of Internet applications rely on the Domain Name System (DNS) to support their operations. Many applications use the DNS to locate services for a domain; some, for example, transform identifiers other than domain names into formats that the DNS can process, and then fetch application data or service location data from the DNS. Proposals incorporating sophisticated application behavior using DNS as a substrate have raised questions about the role of the DNS as an application platform. This document explores the architectural consequences of using the DNS to implement certain application features, and it provides guidance to future application designers as to the limitations of the DNS as a substrate and the situations in which alternative designs should be considered.

Status of This Memo

This document is not an Internet Standards Track specification; it is published for informational purposes.

This document is a product of the Internet Architecture Board (IAB) and represents information that the IAB has deemed valuable to provide for permanent record. It represents the consensus of the Internet Architecture Board (IAB). Documents approved for publication by the IAB are not a candidate for any level of Internet Standard; see Section 2 of RFC 5741.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at http://www.rfc-editor.org/info/rfc6950.

Table of Contents

1. Motivation

The Domain Name System (DNS) has long provided a general means of translating domain names into Internet Protocol addresses, which makes the Internet easier to use by providing a valuable layer of indirection between names and lower-layer protocol elements. [RFC0974] documented a further use of the DNS: to locate an application service operating in a domain, via the Mail Exchange (MX) Resource Record; these records help email addressed to the domain to find a mail service for the domain sanctioned by the zone administrator.

The seminal MX record served as a prototype for other DNS resource records that supported applications associated with a domain name. The SRV Resource Record [RFC2052] provided a more general mechanism for locating services in a domain, complete with a weighting system and selection among transports. The Naming Authority Pointer (NAPTR) Resource Record (originally described in [RFC2168]), especially as it evolved into the more general Dynamic Delegation Discovery System (DDDS) [RFC3401] framework, added a generic mechanism for storing application data in the DNS. Primarily, this involved a client-side algorithm for transforming a string into a domain name, which might then be resolved by the DNS to find NAPTR records. This enabled the resolution of identifiers that do not have traditional host components through the DNS; the best-known examples of this are telephone numbers, as resolved by the DDDS application ENUM. Recent work, such as DomainKeys Identified Mail (DKIM) [RFC6376], has enabled security features of applications to be advertised through the DNS, via the TXT Resource Record.

The scope of application usage of the DNS has thus increased over time. Applications in many environments require features such as confidentiality, and as the contexts in which applications rely on the DNS have increased, some application protocols have looked to extend the DNS to include these sorts of capabilities. However, some proposed usages of, and extensions to, the DNS have become misaligned with both the DNS architecture and the DNS protocol. If we take the example of confidentiality, we see that in the global public DNS, the resolution of domain names to IP addresses is an exchange of public information with no expectation of confidentiality. Thus, the underlying query/response protocol has no encryption mechanism; typically, any security required by an application or service is invoked after the DNS query, when the resolved service has been contacted. Only in private DNS environments (including split-horizon DNS) where the identity of the querier is assured through some external policy can the DNS maintain confidential records, by providing distinct answers to the private and public users of the DNS. In support of load-balancing or other optimizations, a DNS server may return different addresses in response to queries from different sources, or even no response at all; see Section 3.1.1 for details.

This document provides guidance to application designers and application protocol designers looking to use the DNS to support features in their applications. It provides an overview of past application usage of the DNS as well as a review of proposed new usages. It identifies concerns and trade-offs and provides guidance on the question, "Should I store this information in the DNS, or use some other means?" when that question arises during protocol development. These guidelines remind application protocol designers

of the strengths and weaknesses of the DNS in order to make it easier for designers to decide what features the DNS should provide for their application.

The guidance in this document complements the guidance on extending the DNS given in [RFC5507]. Whereas [RFC5507] considers the preferred ways to add new information to the underlying syntax of the DNS (such as defining new resource records or adding prefixes or suffixes to labels), the current document considers broader implications of applications that rely on the DNS for the implementation of certain features, be it through extending the DNS or simply reusing existing protocol capabilities -- implications that may concern the invocation of the resolver by applications; the behavior of name servers, resolvers, or caches; extensions to the underlying DNS protocol; the operational responsibilities of zone administrators; security; or the overall architecture of names. When existing DNS protocol fields are used in ways that their designers did not intend to handle new applications, those applications may demand further changes and extensions that are fundamentally at odds with the strengths of the DNS.

2. Overview of DNS Application Usages

[RFC0882] identifies the original and fundamental connection between the DNS and applications. It begins by describing how the interdomain scope of applications creates "formidable problems when we wish to create consistent methods for referencing particular resources that are similar but scattered throughout the environment". This motivated transitioning the "mapping between host names... and ARPA Internet addresses" from a global table (the original "hosts" file) to a "distributed database that performs the same function". [RFC0882] also envisioned some ways to find the resources associated with mailboxes in a domain: without these extensions, a user trying to send mail to a foreign domain lacked a discovery mechanism to locate the right host in the remote domain to which to connect.

While a special-purpose service discovery mechanism could be built for each such application protocol that needed this functionality, the universal support for the DNS encourages installing these features into its public tree rather than inventing something new. Thus, over time, several other applications leveraged DNS resource records for locating services in a domain or for storing application data associated with a domain in the DNS. This section gives examples of various types of DNS usage by applications to date.

2.1. Locating Services in a Domain

The MX Resource Record provides the simplest example of an application advertising its domain-level resources in the Domain Name System. The MX Resource Record contains the domain name of a server that receives mail on behalf of the administrative domain in question; that domain name must itself be resolved to one or more IP addresses through the DNS in order to reach the mail server. While naming conventions for applications might serve a similar purpose (a host might be named "mail.example.com", for example), approaching service location through the creation of a new resource record yields important benefits. For example, one can put multiple MX records under the same name, in order to designate backup resources or to load-balance across several such servers (see [RFC1794]); these properties could not easily be captured by naming conventions (see [RFC4367], though more recently DNS-based Service Discovery (DNS-SD) [RFC6763] codifies service instance naming conventions for use across applications to locate services in a domain).

While the MX record represents a substantial improvement over naming conventions as a means of service location, it remains specific to a single application. Thus, the general approach of the MX record was adapted to fit a broader class of applications through the Service (SRV) Resource Record (originally described in [RFC2052]). The SRV record allows DNS resolvers to query for particular services and underlying transports (for example, HTTP running over Transport Layer Security (TLS) [RFC2818]) and to learn a host name and port where that service resides in a given domain. It also provides a weighting mechanism to allow load-balancing across several instances of a service.

The reliance of applications on the existence of MX and SRV records has important implications for the way that applications manage identifiers and the way that applications pass domain names to resolvers. Email identifiers of the form "user@domain" rely on MX records to provide the convenience of simply specifying a "domain" component rather than requiring an application to guess which particular host handles mail on behalf of the domain. While naming conventions continue to abound ("www.example.com") for applications like web browsing, SRV records allow applications to query for an application-specific protocol and transport in the domain. For the Lightweight Directory Access Protocol (LDAP), the SRV service name corresponds to the URL scheme of the identifier invoked by the application (e.g., when "ldap://example.com" is the identifier, the SRV query passed to the resolver is for "_ldap._tcp.example.com"); for other applications, the SRV service name that the application passes to the resolver may be implicit in the identifier rather than

explicit. In either case, the application delivers the service name to the DNS to find the location of the host of that service for the domain, the port where the service resides on that host, additional locations or ports for load-balancing and fault tolerance, and related application features.

Locating specific services for a domain was the first major function for which applications started using the DNS beyond simple name resolution. SRV broadened and generalized the precedent of MX to make service location available to any application, rather than just to mail. As applications that acquire MX (or SRV) records might need to perform further queries or transformations in order to arrive at an eventual domain name that will resolve to the IP addresses for the service, [RFC1034] allowed that the Additional (data) section of DNS responses may contain the corresponding address records for the names of services designated by the MX record; this optimization, which requires support in the authoritative server and the resolver, is an initial example of how support for application features requires changes to DNS operation. At the same time, this is an example of an extension of the DNS that cannot be universally relied on: many DNS resolver implementations will ignore the addresses in the additional section of the DNS answers because of the trustworthiness issues described in [RFC2181].

2.2. NAPTR and DDDS

The NAPTR Resource Record evolved to fulfill a need in the transition from Uniform Resource Locators (URLs) to the more mature Uniform Resource Identifier (URI) [RFC3986] framework, which incorporated Uniform Resource Names (URNs). Unlike URLs, URNs typically do not convey enough semantics internally to resolve them through the DNS, and consequently a separate URI-transformation mechanism is required to convert these types of URIs into domain names. This allows identifiers with no recognizable domain component to be treated as domain names for the purpose of name resolution. Once these transformations result in a domain name, applications can retrieve NAPTR records under that name in the DNS. NAPTR records contain a far more rich and complex structure than MX or SRV Resource Records. A NAPTR record contains two different weighting mechanisms ("order" and "preference"), a "service" field to designate the application that the NAPTR record describes, and then two fields that can contain translations: a "replacement" field or a "regexp" (regular expression) field, only one of which appears in a given NAPTR record (see [RFC2168]). A "replacement", like NAPTR's ancestor the PTR record, simply designates another domain name where one would look for records associated with this service in the domain. The

"regexp", on the other hand, allows regular expression transformations on the original URI intended to turn it into an identifier that the DNS can resolve.

As the abstract of [RFC2915] says, "This allows the DNS to be used to lookup services for a wide variety of resource names (including URIs) which are not in domain name syntax". Any sort of hierarchical identifier can potentially be encoded as a domain name, and thus historically the DNS has often been used to resolve identifiers that were never devised as a name for an Internet host. A prominent early example is found in the in-addr domain [RFC0883], in which IPv4 addresses are encoded as domain names by applying a string preparation algorithm that required reversing the octets and treating each individual octet as a label in a domain name -- thus, for example, the address 192.0.2.1 became 1.2.0.192.in-addr.arpa. This allowed resolvers to query the DNS to learn name(s) associated with an IPv4 address. The same mechanism has been applied to IPv6 addresses [RFC3596] and other sorts of identifiers that lack a domain component. Eventually, this idea connected with activities to create a system for resolving telephone numbers on the Internet, which became known as ENUM (originally described in [RFC2916]). ENUM borrowed from an earlier proposal, the "tpc.int" domain [RFC1530], which provided a means for encoding telephone numbers as domain names by applying a string preparation algorithm that required reversing the digits and treating each individual digit as a label in a domain name -- thus, for example, the number +15714345400 became 0.0.4.5.4.3.4.1.7.5.1.tpc.int. In the ENUM system, in place of "tpc.int" the special domain "e164.arpa" was reserved for use.

In the more mature form of the NAPTR standard, in the Dynamic Delegation Discovery System (DDDS) [RFC3401] framework, the initial transformation of an identifier (such as a telephone number) to a domain name was called the "First Well Known Rule". The address-reversing mechanism, whereby a query name is formed by reversing an IPv4 address and prepending it to the in-addr.arpa domain, is generalized for the use of NAPTR: each application defines a "First Well Known Rule" that translates a specific resource into a query name. Its flexibility has inspired a number of proposals beyond ENUM to encode and resolve unorthodox identifiers in the DNS. Provided that the identifiers transformed by the "First Well Known Rule" have some meaningful structure and are not overly lengthy, virtually anything can serve as an input for the DDDS structure: for example, civic addresses. Though [RFC3402] stipulates regarding the identifier that "The lexical structure of this string must imply a unique delegation path", there is no requirement that the identifier be hierarchical nor that the points of delegation in the domain name

created by the "First Well Known Rule" correspond to any points of administrative delegation inherent in the structure of the identifier.

While this ability to look up names "which are not in domain name syntax" does not change the underlying DNS protocol -- the names generated by the DDDS algorithm are still just domain names -- it does change the context in which applications pass name to resolvers and can potentially require very different operational practices of zone administrators (see Section 3.3). In terms of the results of a DNS query, the presence of the "regexp" field of NAPTR records enabled unprecedented flexibility in the types of identifiers that applications could resolve with the DNS. Since the output of the regular expression frequently took the form of a URI (in ENUM resolution, for example, a telephone number might be converted into a SIP URI [RFC3261]), anything that could be encoded as a URI might be the result of resolving a NAPTR record -- which, as the next section explores, essentially means arbitrary data.

2.3. Arbitrary Data in the DNS

URI encoding has ways of encapsulating basically arbitrary data: the most extreme example is a data URL [RFC2397]. Thus, the returned NAPTR record might be interpreted to produce output other than a domain name that would subsequently be resolved to IP addresses and contacted for a particular application -- it could give a literal result that would be consumed by the application. Originally, as discussed in [RFC2168], the intended applicability of the regular expression field in NAPTR was narrower: the "regexp" field contained a "substitution expression that is applied to the original URI in order to construct the next domain name to lookup", in order to "change the host that is contacted to resolve a URI" or as a way of "changing the path or host once the URL has been assigned". The regular expression tools available to NAPTR record authors, however, grant much broader powers to alter the input string, and thus applications began to rely on NAPTR to perform more radical transformations that did not serve any of those aforementioned needs. According to [RFC3402], the output of DDDS is wholly application-specific: "the Application must define what the expected output of the Terminal Rule should be", and the example given in the document is one of identifying automobile parts by inputting a part number and receiving at the end of the process information about the manufacturer.

Historically speaking, NAPTR did not pioneer the storage of arbitrary data in the DNS. At the start, [RFC0882] observed that "it is unlikely that all users of domain names will be able to agree on the set of resources or resource information that names will be used to

retrieve", and consequently places little restriction on the information that DNS records might carry: it might be "host addresses, mailbox data, and other as yet undetermined information". [RFC1035] defined the TXT record, a means to store arbitrary strings in the DNS; [RFC1035] also specifically stipulates that a TXT contains "descriptive text" and that "the semantics of the text depends on the domain where it is found". The existence of TXT records has long provided new applications with a rapid way of storing data associated with a domain name in the DNS, as adding data in this fashion requires no registration process. [RFC1464] experimented with a means of incorporating name/value pairs to the TXT record structure, which allowed applications to distinguish different chunks of data stored in a TXT record -- surely not just "descriptive text" as the TXT originally specified. In this fashion, an application that wants to store additional data in the DNS can do so without registering a new resource record type, though [RFC5507] points out that it is "difficult to reliably distinguish one application's record from others, and for its parser to avoid problems when it encounters other TXT records".

While open policies surrounding the use of the TXT record have resulted in a checkered past for standardizing application usage of TXT, TXT has been used as a technical solution for many applications. Recently, DKIM [RFC6376] sidestepped the problem of TXT ambiguity by storing keys under a specialized DNS naming structure that includes the component "_domainkeys", which serves to restrict the scope of that TXT solely to DKIM use. Storing keys in the DNS became the preferred solution for DKIM for several reasons: notably, because email applications already queried the DNS in their ordinary operations, because the public keys associated with email required wide public distribution, and because email identifiers contain a domain component that applications can easily use to consult the DNS. If the application had to negotiate support for the DKIM mechanism with mail servers, it would give rise to bid-down attacks (where attackers misrepresent that DKIM is unsupported on the originating side) that are not possible if the DNS delivers the keys (provided that DNSSEC [RFC4033] guarantees authenticity of the data). However, there are potential issues with storing large data in the DNS, as discussed in Section 3.2.1, as well as with the DKIM namespace conventions that complicate the use of DNS wildcards (as discussed in Section 6.1.2 of [RFC6376] and in more general terms in [RFC5507]). If prefixes are used to identify TXT records used by an application, potentially the use of wildcards may furthermore cause leakages that other applications will need to detect.

3. Challenges for the DNS

The methods discussed in the previous section for transforming arbitrary identifiers into domain names and returning arbitrary data in response to DNS queries both represent significant departures from the basic function of translating host names to IP addresses, yet neither fundamentally alters the underlying semantics of the DNS. When we consider, however, that the URIs returned by DDDS might be base-64-encoded binary data in a data URL, the DNS could effectively implement the entire application feature set of any simple query-response protocol. Effectively, the DDDS framework considers the DNS a generic database -- indeed, the DDDS framework was designed to work with any sort of underlying database; as [RFC3403] says, the DNS is only one potential database for DDDS to use. Whether the DNS as an underlying database can support the features that some applications of DDDS require, however, is a more complicated question.

As the following subsections will show, the potential for applications to rely on the DNS as a generic database gives rise to additional requirements that one might expect to find in a database access protocol: authentication of the source of queries for comparison to access control lists, formulating complex relational queries, and asking questions about the structure of the database itself. The global public DNS was not designed to provide these sorts of properties, and extending the DNS protocols to encompass them could result in a fundamental alteration to its model. Ultimately, this document concludes that efforts to retrofit these capabilities into the DNS would be better invested in selecting, or if necessary inventing, other Internet services with broader powers than the DNS. If an application protocol designer wants these properties from a database, in general this is a good indication that the DNS cannot, or can only partly, meet the needs of the application in question.

Since many of these new requirements have emerged from the ENUM space, the following sections use ENUM as an illustrative example; however, any application using the DNS as a feature-rich database could easily end up with similar requirements.

3.1. Compound Queries

Traditionally, DNS RRsets are uniquely identified by domain name, resource record type, and class. DNS queries are based on this 3-tuple, and the replies are resource record sets that are to be treated as atomic data elements (see [RFC2181]); to applications, the behavior of the DNS has traditionally been that of an exact-match query-response lookup mechanism. Outside of the DNS space, however, there are plenty of query-response applications that require a

compound or relational search, one taking into account more than one factor in formulating a response or one that uses no single factor as a key to the database. For example, in the telephony space, telephone call routing often takes into account numerous factors aside from the dialed number, including originating trunk groups, interexchange carrier selection, number portability data, time of day, and so on. All are considered simultaneously in generating a route. While in its original conception ENUM hoped to circumvent the traditional Public Switched Telephone Network (PSTN) and route directly to Internet-enabled devices, the infrastructure ENUM effort to support the migration of traditional carrier routing functions to the Internet aspires to achieve feature parity with traditional number routing. However, [RFC3402] explicitly states that "it is an assumption of the DDDS that the lexical element used to make a delegation decision is simple enough to be contained within the Application Unique String itself. The DDDS does not solve the case where a delegation decision is made using knowledge contained outside the AUS and the Rule (time of day, financial transactions, rights management, etc.)". Consequently, some consideration has been given to ways to append additional data to ENUM queries to give the DNS server sufficient information to return a suitable URI (see Section 3.1.1).

From a sheer syntactical perspective, however, domain names do not admit of this sort of rich structure. Several workarounds have attempted to instantiate these sorts of features in DNS queries. For example, the domain name itself could be compounded with the additional parameters: one could take a name like 0.0.4.5.4.3.4.1.7.5.1.e164.arpa and append a trunk group identifier to it, for example, of the form tg011.0.0.4.5.4.3.4.1.7.5.1.e164.arpa. While in this particular case a DNS server can adhere to its traditional behavior in locating resource records, the syntactical viability of encoding additional parameters in this fashion is dubious, especially if more than one additional parameter is required and the presence of parameters is optional so that the application needs multiple queries to assess the completeness of the information it needs to perform its function.

As an alternative, it has been proposed that we piggyback additional query parameters as Extension Mechanisms for DNS (EDNS(0)) extensions (see [RFC6891]). This might be problematic for three reasons. First, supporting EDNS(0) extensions requires significant changes to name server behavior; these changes need to be supported by the authoritative and recursive name servers on which the application relies and might be very hard to realize on a global scale. In addition, the original stated applicability of the EDSN(0) mechanism, as [RFC2671] states, was to "a particular transport level message and not to any actual DNS data", and consequently the OPT Resource

Records it specifies are never to be forwarded. The use of EDNS(0) for compound queries, however, clearly is intended to discriminate actual DNS data rather than to facilitate transport-layer handling. Finally, [RFC6891] also specifies that "OPT RRs MUST NOT be cached, forwarded, or stored" (see the next paragraph). For these reasons, this memo recommends against crafting compound DNS queries by using EDNS(0).

The implications of these sorts of compound queries for recursion and caching are potentially serious. The logic used by the authoritative server to respond to a compound query may not be understood by any recursive servers or caches; intermediaries that naively assume that the response was selected based on the domain name, type, and class alone might serve responses to queries in a different way than the authoritative server intends. Therefore, were EDNS(0) to be employed this way, its attributes would not be transitive, and if this were not considered where intermediaries are employed, as is normally the case in the global DNS, brokenness might occur.

3.1.1. Responses Tailored to the Originator

DNS responses tailored to the identity of their originator, where some sort of administrative identity of the originator must be conveyed to the DNS, constitute the most important subcase of these compound queries. We must first distinguish this from cases where the originating IP address or a similar indication is used to serve a location-specific name. For those sorts of applications, which generally lack security implications, relying on factors like the source IP address introduces little harm; for example, when providing a web portal customized to the region of the client, it would not be a security breach if the client saw the localized portal of the wrong country. Because recursive resolvers may obscure the origination network of the DNS client, a recent proposal suggested introducing a new DNS query parameter to be populated by DNS recursive resolvers in order to preserve the originating IP address (see [EDNS-CLIENT-IP]). However, aside from purely cosmetic uses, these approaches have known limitations due to the prevalence of private IP addresses, VPNs, and so on, which obscure the source IP address and instead supply the IP address of an intermediary that may be very distant from the originating endpoint. Implementing technology such as the one described by [EDNS-CLIENT-IP] would require significant changes in the operation of recursive resolvers and the authoritative servers that would rely on the original source IP address to select resource records, and moreover a fundamental change to caching behavior as well. As a result, such technology cannot be rolled out in an incremental, unilateral fashion but could only be successful when implemented bilaterally (by authoritative server and recursive resolver); this is a significant bar to deployment.

In other deployments in use today, including those based on the BIND "views" feature, the source IP address is used to grant access to a selected, and potentially sensitive, set of resource records. The security implications of trusting the source IP address of a DNS query have prevented most solutions along these lines from being standardized (see [RFC6269]), though the practice remains widespread in "split horizon" private DNS deployments (see Section 4), which typically rely on an underlying security layer, such as a physical network, a clear perimeter demarcation at a network perimeter point (with network-layer anti-spoofing countermeasures), or an IPsec VPN, to prevent spoofing of the source IP address. These deployments do have a confidentiality requirement to prevent information intended for a constrained audience (internal to an enterprise, for example) from leaking to the public Internet -- while these internal network resources may use private IP addresses that should not be useful on the public Internet anyway, in some cases this leakage would reveal topology or other information that the name server administrator hopes to keep private. More recently, TSIG [RFC2845] has been employed as a way of selecting among "views" in BIND; this provides a stronger level of security than merely relying on the source IP address, but typically many users share the same secret to access a given view, and moreover TSIG does not provide confidentiality properties to DNS messages -- without network-layer separation between users of different views, eavesdroppers might capture the DNS queries and responses.

The use of source IP addresses as a discriminator to select DNS resource records, regardless of its lack of acceptance by the standards community, has widespread acceptance in the field. Some applications, however, go even further and propose extending the DNS to add an application-layer identifier of the originator; for example, [EDNS-OPT-CODE] provides a SIP URI in an EDNS(0) parameter. Effectively, this conveyance of application-layer information about the administrative identity of the originator through the DNS is a weak authentication mechanism, on the basis of which the DNS server makes an authorization decision before sharing resource records. This can approximate a confidentiality mechanism per resource record, where only a specific set of originators is permitted to see resource records, or a case where a query for the same name by different entities results in completely different resource record sets. However, without any underlying cryptographic security, this mechanism must rely on external layers for security (such as VPNs) rather than any direct assurance. Again, caching, forwarding, and recursion introduce significant challenges for applications that attempt to offload this responsibility to the DNS. Achieving feature parity with even the simplest authentication mechanisms available at the application layer would likely require significant rearchitecture of the DNS.

3.2. Using DNS as a Generic Database

As previously noted, applications can use a method like the "First Well Known Rule" of DDDS to transform an arbitrary string into a domain name and then receive from the DNS arbitrary data stored in TXT RRs, in the "regexp" of NAPTRs, or even in custom records. Some query-response applications, however, require queries and responses that simply fall outside the syntactic capabilities of the DNS. For example, domain names themselves must consist of labels that do not exceed 63 octets, while the total length of the encoded name may not exceed 255 octets, and applications that use label characters outside the traditional ASCII set may run into problems (however, see the discussion in [RFC6055], Section 3 for definitive guidance on the use of non-ASCII in the DNS). The DNS therefore cannot be a completely generic database. Similar concerns apply to the size of DNS responses.

3.2.1. Large Data in the DNS

While the "data" URL specification [RFC2397] notes that it is "only useful for short values", unfortunately it gives no particular guidance about what "short" might mean. Some applications today use quite large data URLs (containing a megabyte or more of data) as workarounds in environments where only URIs can syntactically appear (for example, in Apple iOS, to pass objects between applications). The meaning of "short" in an application context is probably very different from how we should understand it in a DNS message. Referring to a typical public DNS deployment, [RFC5507] observes that "there's a strong incentive to keep DNS messages short enough to fit in a UDP datagram, preferably a UDP datagram short enough not to require IP fragmentation". And while EDNS(0) allows for mechanisms to negotiate DNS message sizes larger than the traditional 512 octets, there is a high risk that a long payload will cause UDP fragmentation, in particular when the DNS message already carries DNSSEC information. If EDNS(0) is not available, or the negotiated EDNS(0) packet size is too small to fit the data, or UDP fragments are dropped, the DNS may (eventually) resort to using TCP. While TCP allows DNS responses to be quite long, this requires stateful operation of servers, which can be costly in deployments where servers have only fleeting connections to many clients. Ultimately, there are forms of data that an application might store in the DNS that exceed reasonable limits: in the ENUM context, for example, something like storing base-64-encoded mp3 files of custom ringtones.

Designs relying on storage of large amounts of data within DNS RRs furthermore need to minimize the potential damage achievable in a reflection attack (see [RFC4732], Section 3), in which the attacker sends UDP-only DNS queries with a forged source address, and the

victim receives the response. The attacker relies on amplification, where a small query generates a large response directed at the victim. Where the responder supports EDNS(0), an attacker may set the requester maximum payload size to a larger value while querying for a large resource record, such as a certificate [RFC4398]. Thus, the combination of large data stored in DNS RRs and responders supporting large payload sizes has the potential to increase the potential damage achievable in a reflection attack.

Since a reflection attack can be launched from any network that does not implement source address validation, these attacks are difficult to eliminate absent the ubiquitous deployment of source address validation or "heavier" transport protocols such as TCP. The bandwidth that can be mustered in a reflective amplification attack directed by a botnet reflecting off a recursive name server on a high-bandwidth network is sobering. For example, if the responding resolver could be directed to generate a 10KB response in reply to a 50-octet query, then magnification of 200:1 would be attainable. This would enable a botnet controlling 10000 hosts with 1 Mbps of bandwidth to focus 200 Gbps of traffic on the victim, more than sufficient to congest any site on today’s Internet.

DNS reflection attacks typically utilize UDP queries; it is prohibitively difficult to complete a TCP three-way handshake begun from a forged source address for DNS reflection attacks. Unless the attacker uses EDNS(0) [RFC6891] to enlarge the requester’s maximum payload size, a response can only reach 576 octets before the truncate bit is set in the response. This limits the maximum magnification achievable from a DNS query that does not utilize EDNS(0). As the large disparity between the size of a query and size of the response creates this amplification, techniques for mitigating this disparity should be further studied, though this is beyond the scope of this memo (for an analysis of the effects of limiting EDNS(0) responses while still accommodating DNSSEC, see [Lindsay]). For example, some implementations could limit EDNS(0) responses to a specific ratio compared to the request size, where the precise ratio can be configured on a per-deployment basis (taking into account DNSSEC response sizes). Without some means of mitigating the potential for amplification, EDNS(0) could cause significant harm.

In summary, there are two operational forces that tend to drive the practically available EDNS(0) sizes down: possible UDP fragmentation and minimizing amplification in case of reflection attacks. DNSSEC data will use a significant fraction of the available space in a DNS packet. Therefore -- appreciating that given the current DNSSEC and

EDNS(0) deployment experience, precise numbers are impossible to give -- the generic payload available to other DNS data, given the premise that TCP fallback is to be minimized, is likely to be closer to several hundred octets than a few thousand octets.

3.3. Administrative Structures Misaligned with the DNS

While the DDDS framework enables any sort of alphanumeric data to serve as a domain name through the application of the "First Well Known Rule", the delegative structure of the resulting domain name may not reflect any administrative division of responsibilities inherent in the original data. While [RFC3402] requires only that the "Application Unique String has some kind of regular, lexical structure that the rules can be applied to", DDDS is first and foremost a delegation system: its abstract stipulates that "Well-formed transformation rules will reflect the delegation of management of information associated with the string". Telephone numbers in the United States, for example, are assigned and delegated in a relatively complex manner. Historically, the first six digits of a nationally specific number (called the "NPA/NXX") reflected a point of administrative delegation from the number assignment agency to a carrier; from these blocks of ten thousand numbers, the carrier would in turn delegate individual assignments of the last four digits (the "XXXX" portion) to particular customers. However, after the rise of North American telephone number portability in the 1990s, the first point of delegation went away: the delegation is effectively from the number authority to the carrier for each complete ten-digit number (NPA/NXX-XXXX). While technical implementation details differ from nation to nation, number portability systems with similar administrative delegations now exist worldwide.

To render these identifiers as domain names in accordance with the DDDS Rule for ENUM yields a large flat administrative domain with no points of administrative delegation from the country-code administrator, e.g., 1.e164.arpa, down to the ultimate assignee of a number. Under the assumption that one administrative domain is maintained within one DNS zone containing potentially over five billion names, scalability difficulties manifest in a number of areas: the scalability that results from caching depends on these points of delegation, so that cached results for intermediate servers take the load off higher-tier servers. If there are no such delegations, then as in the telephone number example the zone apex server must bear the entire load for queries. Worse still, number portability also introduces far more dynamism in number assignment, where in some regions updated assignees for ported numbers must propagate within fifteen minutes of a change in administration. Jointly, these two problems make caching the zone extremely problematic. Moreover, traditional tools for DNS replication, such

as the zone transfer protocols AXFR [RFC1034] and IXFR [RFC1995], might not scale to accommodate zones with these dimensions and properties.

In practice, the maximum sizes of telephone number administrative domains are closer to 300M (the current amount of allocated telephone numbers in the United States today -- still more than three times the number of .com domain names), and one can alleviate some of the scalability issues mentioned above by artificially dividing the administrative domain into a hierarchy of DNS zones. Still, the fact that traditional DNS management tools no longer apply to the structures that an application tries to provision in the DNS is a clue that the DNS might not be the right place for an application to store its data.

While DDDS is the most obvious example of these concerns, the point is more generic: for example, were address portability to be implemented for IP addresses and their administration thus to become non-hierarchical, the same concerns would apply to in-addr.arpa names. The difficulty of mapping the DNS to administrative structures can even occur with traditional domain names, where applications expect clients to infer or locate zone cuts. In the web context, for example, it can be difficult for applications to determine whether two domains represent the same "site" when comparing security credentials with URLs (see Section 3.4 below for more on this). This has also caused known problems in determining the scope of web cookies, in contexts where applications must infer where administrative domains end in order to grant cookies that are as narrowly scoped as possible.

In summary, the "First Well Known Rule" of DDDS provides a capability that transforms arbitrary strings into domain names, but those names play well with the DNS only when the input strings have an administrative structure that maps to DNS delegations. In the first place, delegation implies some sort of hierarchical structure. Any mechanism to map a hierarchical identifier into a domain name should be constructed such that the resulting domain name does match the natural hierarchy of the original identifier. Although telephone numbers, even in North America, have some hierarchical qualities (like the geographical areas corresponding to their first three digits), after the implementation of number portability these points no longer mapped onto an administrative delegation. If the input string to the DDDS does not have a hierarchical structure representing administrative delegations that can map onto the DNS distribution system, then that string probably is not a good candidate for translating into a domain name.

3.3.1. Metadata about Tree Structure

There are also other ways in which the delegative structure of an identifier may not map well onto the DNS. Traditionally, DNS resolvers assume that when they receive a domain name from an application the name is complete -- that it is not a fragment of a domain name that a user is still in the middle of typing. For some communications systems, however, this assumption does not apply. ENUM use cases have surfaced a couple of optimization requirements to reduce unnecessary calls and queries; proposed solutions include metadata in the DNS that describes the contents and structure of ENUM DNS trees to help applications handle incomplete queries or queries for domains not in use.

In particular, the "send-n" proposal [ENUM-Send-N] hoped to reduce the number of DNS queries sent in regions with variable-length numbering plans. When a dialed number potentially has a variable length, a telephone switch ordinarily cannot anticipate when a dialed number is complete, as only the numbering plan administrator (who may be a national regulator, or even in open number plans a private branch exchange) knows how long a telephone number needs to be. Consequently, a switch trying to resolve such a number through a system like ENUM might send a query for a telephone number that has only partially been dialed in order to test its completeness. The send-n proposal installs in the DNS a hint informing the telephone switch of the minimum number of digits that must be collected by placing in zones corresponding to incomplete telephone numbers some resource records that state how many more digits are required -- effectively how many steps down the DNS tree one must take before querying the DNS again. Unsurprisingly, those boundaries reflect points of administrative delegation, where the parent in a number plan yields authority to a child. With this information, the application is not required to query the DNS every time a new digit is dialed but can wait to collect sufficient digits to receive a response. As an optimization, this practice thus saves the resources of the DNS server, though the call cannot complete until all digits are collected, so this mechanism simply reduces the time the system will wait before sending an ENUM query after collecting the final dialed digit. A tangentially related proposal, [ENUM-UNUSED], similarly places resource records in the DNS that tell the application that it need not attempt to reach a number on the telephone network, as the number is unassigned -- a comparable general DNS mechanism would identify, for a domain name with no records available in the DNS, whether or not the domain had been allocated by a registry to a registrant (which is a different condition than a name merely being unresolvable, per the semantics of NXDOMAIN).

Both proposals optimize application behavior by placing metadata in the DNS that predicts the success of future queries or application invocation by identifying points of administrative delegation or assignment in the tree. In some respects, marking a point in the tree where a zone begins or ends has some features in common with the traditional parent zone use of the NS record type, except that instead of pointing to a child zone these metadata proposals point to distant grandchildren. While this does not change resolver behavior as such (instead, it changes the way that applications invoke the resolver), it does have implications for the practices for zone administrators. Metadata in one administrative domain would need to remain synchronized with the state of the resources it predicts in another administrative domain in the DNS namespace, in a case like overlap dialing where the carrier delegates to a zone controlled by an enterprise. When dealing with external resources associated with other namespaces, like number assignments in the PSTN or the databases of a registry operator, other synchronization requirements arise; maintaining that synchronization requires that the DNS have "semi-real time" updates that may conflict with scale and caching mechanisms of the DNS.

Placing metadata in the DNS may also raise questions about the authority and delegation model. Who gets to supply records for unassigned names? While in the original but little-used e164.arpa root of ENUM this would almost certainly be a numbering plan administrator, it is far less clear who that would be in the more common and successful "infrastructure" ENUM models (see Section 4). Ultimately, these metadata proposals share some qualities with DNS redirection services offered by ISPs (for example, [DNS-REDIRECT]) that "help" web users who try to browse to sites that do not exist. Similarly, metadata proposals like [ENUM-UNUSED] create DNS records for unallocated zones that redirect to a service provider's web page. However, in the [DNS-REDIRECT] cases, at least the existence or non-existence of a domain name is a fact about the Internet namespace, rather than about an external namespace like the telephony E.164 namespace (which must be synchronized with the DNS tree in the metadata proposals). In send-n, different leaf zones that administer telephone numbers of different lengths can only provision their hints at their own apex, which provides an imperfect optimization: they cannot install it themselves in a parent, both because they lack the authority and because different zones would want to provision contradictory data. The later the hint appears in the tree, however, the less optimization will result. An application protocol designer managing identifiers whose administrative model does not map well onto the DNS namespace and delegations structure would be better served to implement such features outside the DNS.

3.4. Domain Redirection

Most Internet application services provide a redirection feature -- when one attempts to contact a service, the service may refer the person to a different service instance, potentially in another domain, that is for whatever reason better suited to service a request. In HTTP and SIP, for example, this feature is implemented by the 300 class responses containing one or more URIs, which may indicate that a resource has moved temporarily or permanently to another service. Several tools in the DNS, including the SRV record, can provide a similar feature at a DNS level, and consequently some applications as an optimization offload the responsibility for redirection to the DNS; NAPTR can also provide this capability on a per-application basis, and numerous DNS resource records can provide redirection on a per-domain basis. This can prevent the unnecessary expenditure of application resources on a function that could be performed as a component of a DNS lookup that is already a prerequisite for contacting the service. Consequently, in some deployment architectures this DNS-layer redirection is used for virtual hosting services.

Implementing domain redirection in the DNS, however, has important consequences for application security. In the absence of universal DNSSEC, applications must blindly trust that their request has not been hijacked at the DNS layer and redirected to a potentially malicious domain, unless some subsequent application mechanism can provide the necessary assurance. By way of contrast, application-layer protocols supporting redirection, such as HTTP and SIP, have available security mechanisms, including TLS, that can use certificates to attest that a 300 response came from the domain that the originator initially hoped to contact.

A number of applications have attempted to provide an after-the-fact security mechanism that verifies the authority of a DNS delegation in the absence of DNSSEC. The specification for dereferencing SIP URIs ([RFC3263], reaffirmed in [RFC5922]), requires that during TLS establishment the site eventually reached by a SIP request present a certificate corresponding to the original URI expected by the user; this requires a virtual hosting service to possess a certificate corresponding to the hosted domain. (In other words, if example.com redirects to example.net in the DNS, this mechanism expects that example.net will supply a certificate for example.com in TLS, per the HTTP precedent in [RFC2818]). This restriction rules out many styles of hosting deployments common in the web world today, however. [HARD-PROBLEM] explores this problem space. [RFC6125] proposes a solution for all applications that use TLS, which relies on new application-specific identifiers in certificates, as does [RFC4985]); note, however, that support for such certificates would require

changes to existing certificate authority practices as well as application behavior. With DNSSEC in place, DNS-based Authentication of Named Entities (DANE) [RFC6394] offers another way to bind certificates to particular applications and services.

All of these application-layer measures attempt to mirror the delegation of administrative authority in the DNS, when the DNS itself serves as the ultimate authority on how domains are delegated. (Note: changing the technical delegation structure by changing NS records in the DNS is not the same as administrative delegation, e.g., when a domain changes ownership.) Synchronizing a static instrument like a certificate with a delegation in the DNS, however, is problematic because delegations are not static: revoking and reissuing a certificate every time an administrative delegation changes is cumbersome operationally. In environments where DNSSEC is not available, the problems with securing DNS-layer redirections would be avoided by performing redirections in the application layer. This inevitably gives rise to various design trade-offs involving performance, load, and related factors, but in these application environments, the security properties typically take priority.

4. Private DNS and Split Horizon

The classic view of the uniqueness of domain names in the DNS is given in [RFC2826]:

> DNS names are designed to be globally unique, that is, for any one DNS name at any one time there must be a single set of DNS records uniquely describing protocol addresses, network resources and services associated with that DNS name. All of the applications deployed on the Internet which use the DNS assume this, and Internet users expect such behavior from DNS names.

[RFC2826] "does not preclude private networks from operating their own private name spaces" but notes that if private networks "wish to make use of names uniquely defined for the global Internet, they have to fetch that information from the global DNS naming hierarchy". There are various DNS deployments outside of the global public DNS, including "split horizon" deployments and DNS usages on private (or virtual private) networks. In a split horizon, an authoritative server gives different responses to queries from the public Internet than they do to internal resolvers; while some deployments differentiate internal queries from public queries by the source IP address, the concerns in Section 3.1.1 relating to trusting source IP addresses apply to such deployments. When the internal address space range is private [RFC1918], this makes it both easier for the server to discriminate public from private and harder for public entities to impersonate nodes in the private network. Networks that are made

private physically, or logically by cryptographic tunnels, make these private deployments more secure. The most complex deployments along these lines use multiple virtual private networks to serve different answers for the same name to many distinct networks.

The use cases that motivate split-horizon DNS typically involve restricting access to some network services -- intranet resources such as internal web sites, development servers, or directories, for example -- while preserving the ease of use offered by domain names for internal users. While for many of these resources the split horizon would not return answers to public resolvers for those internal resources (those records are kept confidential from the public), in some cases the same name (e.g., "mail.example.com") might resolve to one host internally but another externally. The requirements for multiple-VPN private deployments, however, are different: in this case the authoritative server gives different, and confidential, answers to a set of resolvers querying for the same name. While these sorts of use cases rarely arise for traditional domain names, where, as [RFC2826] says, users and applications expect a unique resolution for a name, they can easily arise when other sorts of identifiers have been translated by a mechanism such as the "First Well Known Rule" of DDDS into "domain name syntax". Telephone calls, for example, are traditionally routed through highly mediated networks, in which an attempt to find a route for a call often requires finding an appropriate intermediary based on the source network and location rather than finding an endpoint (see the distinction between the Look-Up Function and Location Routing Function in [RFC5486]). Moreover, the need for responses tailored to the originator, and for confidentiality, is easily motivated when the data returned by the DNS is no longer "describing protocol addresses, network resources and services" [RFC2826] but instead is arbitrary data. Although, for example, ENUM was originally intended for deployment in the global public root of the DNS (under e164.arpa), the requirements of maintaining telephone identifiers in the DNS quickly steered operators into private deployments.

In private environments, it is also easier to deploy any necessary extensions than it is in the public DNS: in the first place, proprietary non-standard extensions and parameters can more easily be integrated into DNS queries or responses, as the implementations of resolvers and servers can likely be controlled; secondly, confidentiality and custom responses can be provided by deploying, respectively, underlying physical or virtual private networks to shield the private tree from public queries, and effectively different virtual DNS trees for each administrative entity that might launch a query; thirdly, in these constrained environments, caching and recursive resolvers can be managed or eliminated in order to prevent any unexpected intermediary behavior. While these private

deployments serve an important role in the marketplace, there are risks in using techniques intended only for deployment in private and constrained environments as the basis of a standard solution. When proprietary parameters or extensions are deployed in private environments, experience teaches us that these implementations will begin to interact with the public DNS and that the private practices will leak.

While such leakage is not a problem when using the mechanisms described in Sections 3.1, 3.2, and 3.5 (with private RR types) of [RFC5507], other extension mechanisms might cause confusion or harm if leaked. The use of a dedicated suffix (Section 3.3 of [RFC5507]) in a private environment might cause confusion if leaked to the public Internet, for example.

That this kind of leakage of protocol elements from private deployments to public deployments does happen has been demonstrated, for example, with SIP: SIP implemented a category of supposedly private extensions (the "P-" headers) that saw widespread success and use outside of the constrained environments for which they were specifically designed. There is no reason to think that implementations with similar "private" extensions to the DNS protocols would not similarly end up in use in public environments.

5. Principles and Guidance

The success of the global public DNS relies on the fact that it is a distributed database, one that has the property that it is loosely coherent and offers lookup instead of search functionality. Loose coherency means that answers to queries are coherent within the bounds of data replication between authoritative servers (as controlled by the administrator of the zone) and caching behavior by recursive name servers. Today, this term increasingly must also include load-balancing or related features that rely on the source IP address of the resolver. It is critical that application designers who intend to use the DNS to support their applications consider the implications that their uses have for the behavior of resolvers; intermediaries, including caches and recursive resolvers; and authoritative servers.

It is likely that the DNS provides a good match whenever the needs of applications are aligned with the following properties:

- Data stored in the DNS can be propagated and cached using conventional DNS mechanisms, without intermediaries needing to understand exceptional logic (considerations beyond the name, type, and class of the query) used by the authoritative server to formulate responses

- Data stored in the DNS is indexed by keys that do not violate the syntax or semantics of domain names

- Applications invoke the DNS to resolve complete names, not fragments

- Answers do not depend on an application-layer identity of the entity doing the query

- Ultimately, applications invoke the DNS to assist in communicating with a service whose name is resolved through the DNS

Whenever one of the five properties above does not apply to one's data, one should seriously consider whether the DNS is the best place to store actual data. On the other hand, if one has to worry about the following items, then these items are good indicators that the DNS is not the appropriate tool for solving problems:

- Populating metadata about domain boundaries within the tree -- the points of administrative delegation in the DNS are something that applications are not in general aware of

- Domain names derived from identifiers that do not share a semantic or administrative model compatible with the DNS

- Selective disclosure of data stored in and provided by the DNS

- DNS responses not fitting into UDP packets, unless EDNS(0) is available, and only then with the caveats discussed in Section 3.2.1

In cases where applications require these sorts of features, they are likely better instantiated by independent application-layer protocols than the DNS. For example, the objects that HTTP can carry in both queries and responses can easily contain the necessary structure to manage compound queries. Many applications already use HTTP because of widespread support for it in middleboxes. Similarly, HTTP has numerous ways to provide the necessary authentication, authorization, and confidentiality properties that some features require, as well as

the redirection properties discussed in Section 3.4. These differences highlight the fact that the DNS and HTTP offer very different services and have different applicabilities; while both are query-response protocols, HTTP should not be doing the job of DNS, and DNS should not be doing the job of HTTP. Similarly, DNS should not be doing the job of Diameter, LDAP, or other application-layer protocols. The overhead of using any application-layer protocol may not be appropriate for all environments, of course, but even in environments where a more lightweight protocol is appropriate, DNS is usually not the only alternative.

Where the administrative delegations of the DNS form a necessary component in the instantiation of an application feature, there are various ways that the DNS can bootstrap access to an independent application-layer protocol better suited to field the queries in question. For example, since NAPTR or URI [URI-RR] Resource Records can contain URIs, those URIs can in turn point to an external query-response service such as an HTTP service where more syntactically and semantically rich queries and answers might be exchanged. Any protocol designer considering where to implement features must perform their own gap analysis and determine whether or not implementing some features is worth the potential cost in increased network state, latency, and so on, but implementing some features simply requires heavier structures than others.

6. Security Considerations

Many of the concerns about how applications use the DNS discussed in this document involve security. The perceived need to authenticate the source of DNS queries (see Section 3.1.1) and authorize access to particular resource records also illustrates the fundamental security principles that arise from offloading certain application features to the DNS. As Section 3.2.1 observes, large data in the DNS can provide a means of generating reflection attacks, and without the remedies discussed in that section (regarding the use of EDNS(0) and TCP) the presence of large sets of records (e.g., ANY queries) is not recommended. Section 3.4 discusses a security problem concerning redirection that has surfaced in a number of protocols (see [HARD-PROBLEM]).

While DNSSEC, were it deployed universally, can play an important part in securing application redirection in the DNS, DNSSEC does not provide a means for a resolver to authenticate itself to a server, nor a framework for servers to return selective answers based on the authenticated identity of resolvers, nor a confidential mechanism. Some implementations may support authenticating users through TSIG, provided that the security association with a compliant server has been pre-established, though authentication is typically not needed

for queries in the global public DNS. The existing feature set of DNSSEC is, however, sufficient for providing security for most of the ways that applications traditionally have used the DNS. The deployment of DNSSEC ([RFC4033] and related specifications) is heartily encouraged. Nothing in this document is intended to discourage the implementation, deployment, or use of Secure DNS Dynamic Updates [RFC3007], though this document does recommend that large data in the DNS be treated in accordance with the guidance in Section 3.2.1.

7. IAB Members at the Time of Approval

Internet Architecture Board Members at the time this document was approved were:

Bernard Aboba
Jari Arkko
Marc Blanchet
Ross Callon
Alissa Cooper
Spencer Dawkins
Joel Halpern
Russ Housley
David Kessens
Danny McPherson
Jon Peterson
Dave Thaler
Hannes Tschofenig

8. Acknowledgements

The IAB appreciates the comments and often spirited disagreements of Eric Osterweil, John Levine, Stephane Bortzmeyer, Ed Lewis, Dave Crocker, Ray Bellis, Lawrence Conroy, Ran Atkinson, Patrik Faltstrom, and Eliot Lear.

9. Informative References

[DNS-REDIRECT]
Creighton, T., Griffiths, C., Livingood, J., and R. Weber, "DNS Redirect Use by Service Providers", Work in Progress, October 2010.

[EDNS-CLIENT-IP]
Contavalli, C., van der Gaast, W., Leach, S., and D. Rodden, "Client IP information in DNS requests", Work in Progress, May 2010.

[EDNS-OPT-CODE]
Kaplan, H., Walter, R., Gorman, P., and M. Maharishi, "EDNS Option Code for SIP and PSTN Source Reference Info", Work in Progress, October 2011.

[ENUM-Send-N]
Bellis, R., "IANA Registrations for the 'Send-N' Enumservice", Work in Progress, June 2008.

[ENUM-UNUSED]
Stastny, R., Conroy, L., and J. Reid, "IANA Registration for Enumservice UNUSED", Work in Progress, March 2008.

[HARD-PROBLEM]
Barnes, R. and P. Saint-Andre, "High Assurance Re-Direction (HARD) Problem Statement", Work in Progress, July 2010.

[Lindsay] Lindsay, G., "DNSSEC and DNS Amplification Attacks", April 2012.

[RFC0882] Mockapetris, P., "Domain names: Concepts and facilities", RFC 882, November 1983.

[RFC0883] Mockapetris, P., "Domain names: Implementation specification", RFC 883, November 1983.

[RFC0974] Partridge, C., "Mail routing and the domain system", STD 10, RFC 974, January 1986.

[RFC1034] Mockapetris, P., "Domain names - concepts and facilities", STD 13, RFC 1034, November 1987.

[RFC1035] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, November 1987.

[RFC1464] Rosenbaum, R., "Using the Domain Name System To Store Arbitrary String Attributes", RFC 1464, May 1993.

[RFC1530] Malamud, C. and M. Rose, "Principles of Operation for the TPC.INT Subdomain: General Principles and Policy", RFC 1530, October 1993.

[RFC1794] Brisco, T., "DNS Support for Load Balancing", RFC 1794, April 1995.

[RFC1918] Rekhter, Y., Moskowitz, B., Karrenberg, D., de Groot, G., and E. Lear, "Address Allocation for Private Internets", BCP 5, RFC 1918, February 1996.

[RFC1995] Ohta, M., "Incremental Zone Transfer in DNS", RFC 1995, August 1996.

[RFC2052] Gulbrandsen, A. and P. Vixie, "A DNS RR for specifying the location of services (DNS SRV)", RFC 2052, October 1996.

[RFC2168] Daniel, R. and M. Mealling, "Resolution of Uniform Resource Identifiers using the Domain Name System", RFC 2168, June 1997.

[RFC2181] Elz, R. and R. Bush, "Clarifications to the DNS Specification", RFC 2181, July 1997.

[RFC2397] Masinter, L., "The "data" URL scheme", RFC 2397, August 1998.

[RFC2671] Vixie, P., "Extension Mechanisms for DNS (EDNS0)", RFC 2671, August 1999.

[RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, May 2000.

[RFC2826] Internet Architecture Board, "IAB Technical Comment on the Unique DNS Root", RFC 2826, May 2000.

[RFC2845] Vixie, P., Gudmundsson, O., Eastlake 3rd, D., and B. Wellington, "Secret Key Transaction Authentication for DNS (TSIG)", RFC 2845, May 2000.

[RFC2915] Mealling, M. and R. Daniel, "The Naming Authority Pointer (NAPTR) DNS Resource Record", RFC 2915, September 2000.

[RFC2916] Faltstrom, P., "E.164 number and DNS", RFC 2916, September 2000.

[RFC3007] Wellington, B., "Secure Domain Name System (DNS) Dynamic Update", RFC 3007, November 2000.

[RFC3261] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.

[RFC3263] Rosenberg, J. and H. Schulzrinne, "Session Initiation Protocol (SIP): Locating SIP Servers", RFC 3263, June 2002.

[RFC3401] Mealling, M., "Dynamic Delegation Discovery System (DDDS) Part One: The Comprehensive DDDS", RFC 3401, October 2002.

[RFC3402] Mealling, M., "Dynamic Delegation Discovery System (DDDS) Part Two: The Algorithm", RFC 3402, October 2002.

[RFC3403] Mealling, M., "Dynamic Delegation Discovery System (DDDS) Part Three: The Domain Name System (DNS) Database", RFC 3403, October 2002.

[RFC3596] Thomson, S., Huitema, C., Ksinant, V., and M. Souissi, "DNS Extensions to Support IP Version 6", RFC 3596, October 2003.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.

[RFC4033] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "DNS Security Introduction and Requirements", RFC 4033, March 2005.

[RFC4367] Rosenberg, J., Ed., and IAB, "What’s in a Name: False Assumptions about DNS Names", RFC 4367, February 2006.

[RFC4398] Josefsson, S., "Storing Certificates in the Domain Name System (DNS)", RFC 4398, March 2006.

[RFC4732] Handley, M., Ed., Rescorla, E., Ed., and IAB, "Internet Denial-of-Service Considerations", RFC 4732, December 2006.

[RFC4985] Santesson, S., "Internet X.509 Public Key Infrastructure Subject Alternative Name for Expression of Service Name", RFC 4985, August 2007.

[RFC5486] Malas, D., Ed., and D. Meyer, Ed., "Session Peering for Multimedia Interconnect (SPEERMINT) Terminology", RFC 5486, March 2009.

[RFC5507] IAB, Faltstrom, P., Ed., Austein, R., Ed., and P. Koch, Ed., "Design Choices When Expanding the DNS", RFC 5507, April 2009.

[RFC5922] Gurbani, V., Lawrence, S., and A. Jeffrey, "Domain Certificates in the Session Initiation Protocol (SIP)", RFC 5922, June 2010.

[RFC6055] Thaler, D., Klensin, J., and S. Cheshire, "IAB Thoughts on Encodings for Internationalized Domain Names", RFC 6055, February 2011.

[RFC6125] Saint-Andre, P. and J. Hodges, "Representation and Verification of Domain-Based Application Service Identity within Internet Public Key Infrastructure Using X.509 (PKIX) Certificates in the Context of Transport Layer Security (TLS)", RFC 6125, March 2011.

[RFC6269] Ford, M., Ed., Boucadair, M., Durand, A., Levis, P., and P. Roberts, "Issues with IP Address Sharing", RFC 6269, June 2011.

[RFC6376] Crocker, D., Ed., Hansen, T., Ed., and M. Kucherawy, Ed., "DomainKeys Identified Mail (DKIM) Signatures", RFC 6376, September 2011.

[RFC6394] Barnes, R., "Use Cases and Requirements for DNS-Based Authentication of Named Entities (DANE)", RFC 6394, October 2011.

[RFC6763] Cheshire, S. and M. Krochmal, "DNS-Based Service Discovery", RFC 6763, February 2013.

[RFC6891] Damas, J., Graff, M., and P. Vixie, "Extension Mechanisms for DNS (EDNS(0))", STD 75, RFC 6891, April 2013.

[URI-RR] Faltstrom, P. and O. Kolkman, "The Uniform Resource Identifier (URI) DNS Resource Record", Work in Progress, July 2013.

Authors' Addresses

Jon Peterson
NeuStar, Inc.

EMail: jon.peterson@neustar.biz

Olaf Kolkman
NLnet Labs

EMail: olaf@nlnetlabs.nl

Hannes Tschofenig
Nokia Siemens Networks

EMail: Hannes.Tschofenig@gmx.net

Bernard Aboba
Skype

EMail: Bernard_aboba@hotmail.com