Network Working Group
Internet-Draft
Expires: September 5, 2005

P. Faltstrom
R. Austein
IAB

March 7, 2005

Design Choices When Expanding DNS
draft-iab-dns-choices-01.txt

Abstract

This note discusses how to extend the DNS with new data for a new application. DNS extension discussion too often circulates around reuse of the TXT record. This document lists different mechanisms to accomplish the extension to DNS, and comes to the conclusion that the use of a new RR Type is the best solution.

Table of Contents

1. Introduction

The DNS stores multiple categories of data. The two most commonly used categories are infrastructure data for the DNS system itself (NS and SOA records) and data which have to do with mappings between domain names and IP addresses (A, AAAA and PTR). There are other categories as well, some of which are tied to specific applications like email (MX), while others are generic record types used to convey information for multiple protocols (SRV, NAPTR).

When storing data in the DNS for a new application, the data are usually tied to a "normal" domain name, so the application can query for the data it wants, while minimizing the impact on existing applications.

Historically, extending DNS to store data for applications tied to a domain name has been done in different ways at different times. MX records were created as a new resource record type specifically designed to support electronic mail. SRV records are a generic type which use a prefixing scheme in combination with a base domain name. Records associated with ENUM use a suffixing scheme. NAPTR records add selection data inside the RDATA. It is clear the way of adding new data to the DNS has been inconsistent, and the purpose of this document is to attempt to clarify the implications of each of these methods, both for the applications that use them and for the rest of the DNS system.

Many parts of this document talk about use of wildcards, and many people might think use of wildcards is not something that happens today. In reality thoug, wildcards are in use, especially for some specific usages like MX records. Because of this, the choice have to be made with existence of wildcards in mind.

Another overall issue that have to be taken into account is what the new data in DNS is to describe. In some cases it might be completely new data. In other cases it might be meta-data that is tied to data that already exists in the DNS. Example of new data is key information for SSH and data used for fighting spam (meta data that is connected to the MX record). If the new data has connection to data that already exists in DNS, an analysis should be made whether having (for example) A record and SSH key information in different zones is a problem, and if it is, whether the owner for the records by design must be the same for both records.

This document do not talk about what one should store in DNS. Whether DNS is used for service discovery, or whether DNS is (also) used for storage of application specific data. In general, DNS is a protocol that part from holding metadata that makes DNS itself

function (NS, SOA, DS RR Types etc) only holds references to where services are located (SRV, NAPTR, A, AAAA RR types).

2. Background

See RFC 2929 [RFC2929] for a brief summary of DNS query structure. Readers interested in the full story should start with the base DNS specification in RFC 1035 [RFC1035], and continue with the various documents which update, clarify, and extend the base specification.

When composing a query into the DNS system, the parameters actually used by the protocol are a triple: a DNS name, a DNS class, and a DNS record type. Every resource record (RR) matching a particular name, type and class is said to belong to the same resource record set (RRSet), and the whole RRSet is always returned to the client which queries for it. Splitting an RRSet is a protocol violation, because it results in coherency problems with the DNS caching mechanism.

Note however that this requirement has nothing to do with the MTU size and choice of UDP or TCP as the transport protocol. The whole RRSet always has to be returned, and if it doesn't fit in the MTU in UDP, TCP has to be used to not break this RRSet atomic requirement.

3. Extension mechanisms

The DNS protocol is intended to be extensible to support new kinds of data. This section examines the various ways in which this sort of extension can be accomplished.

3.1 Place selectors inside the RDATA

For a given query name, one might choose to have a single RRSet (all sharing the same name, type, and class) shared by multiple applications, and have the different applications use selectors within the RR data (RDATA) to determine which records are intended for which applications. This sort of selector mechanism is usually referred to "subtyping", because it is in effect creating an additional type subsystem within a single DNS RR type.

Examples of subtyping include NAPTR RRs (see RFC 3761 [RFC3761]) and the original DNSSEC KEY RR type (RFC 2535 [RFC2535]) (before it was updated by RFC 3445 [RFC3445]).

All DNS subtyping schemes share a common weakness: With subtyping schemes it is impossible for a client to query for just the data it wants. Instead, the client must fetch the entire RRSet, then select the RRs in which it is interested. Furthermore, since DNSSEC signatures operate on complete RRSets, the entire RRSet must be

re-signed if any RR in it changes. As a result, each application that uses a subtyped RR incurs higher overhead than any of the applications would have incurred had they not been using a subtyping scheme. The fact the RRSet is always passed around as an indivisible unit increases the risk the RRSet will not fit in a UDP packet, which in turn increases the risk that the client will have to retry the query with TCP, which substantially increases the load on the name server. More precisely: Having one query fail over to TCP is not a big deal, but since the typical ratio of clients to servers in the DNS system is very high, having a substantial number of DNS messages fail over to TCP it will cause the relevant name servers to be overloaded by TCP overhead.

The final result of using a subtyping scheme might be that the zone administrator has to choose which of the services tied to one domain name can actually be used, because not all of them can be announced at the same time.

3.2 Add a prefix to the owner name

By adding an application-specific prefix to a domain name, we will get different name/class/type triples, and therefore different RRSets. One problem with adding prefixes has to do with wildcards, especially if one has records like

```
*.example.com. IN MX 1 mail.example.com.
```

and one wants records tied to those names. Suppose one creates the prefix "_mail". One would then have to say something like

```
_mail.*.example.com. IN X-FOO A B C D
```

but DNS wildcards only work with the "*" as the leftmost token in the domain name.

Even when a specific prefix is chosen, the data will still have to be stored in some RR type. This RR type can either be a record type that can store arbitrary data or a new RR type. This implies that some other selection mechanism has to be applied as well, such as ability to distinguish between the records in an RR set given they have the same RR Type (see also draft-ietf-dnsext-wcard-clarify [wcardclarify] regarding use of wildcards and DNS). Because of this one need to register both a unique prefix and define what RR type is to be used for this specific service.

If the record has some relationship with another record, the fact the original record and this with a prefix can be in different zones might have implications on the trust in the application have on the

records. Example:

```
example.com.      IN X-FOO  "some good thing"
_foo.example.com. IN X-BAR  "why the good thing is good"
```

In this example, the two records might be in two different zones, and because of this signed by two different organizations when using DNSSEC.

3.3 Add a suffix to the owner name

Adding a suffix to a domain name changes the name/class/type triple, and therefore the RRSet. The query name can be set to exactly the data one wants, and the size of the RRSet is minimized. The problem with adding a suffix is that it creates a parallel tree within the IN class. There will be no technical mechanism to ensure that the delegation for "example.com" and "example.com._bar" are made to the same organization. Furthermore, data associated with a single entity will now be stored in two different zones, such as "example.com" and "example.com._bar", which, depending on who controls "_bar", can create new synchronization and update authorization issues.

Even when using a different name, the data will still have to be stored in some RR type. This RR type can either be a "kitchen-sink record" or a new RR type. This implies that some other mechanism has to be applied as well, with implications detailed in other parts of this note.

In RFC 2163 [RFC2163] an infix token is inserted directly below the TLD, but the result is the same as adding a suffix to the owner name (and because of that creation of a new TLD).

3.4 Add a new Class

DNS zones are class-specific, in the sense that all the records in that zone share the same class as the zone's SOA record, and the existence of a zone in one class does not guarantee the existence of the zone in any other class. In practice, only the IN class has ever seen widespread deployment, and the administrative overhead of deploying an additional class would almost certainly be prohibitive.

Nevertheless, one could in theory use the DNS class mechanism to distinguish between different kinds of data. However, since the DNS delegation tree (represented by NS RRs) is itself tied to a specific class, attempting to resolve a query by crossing a class boundary may produce unexpected results, because there is no guarantee that the name servers for the zone in the new class will be the same as the name servers in the IN class. The MIT Hesiod system used a scheme

like this for storing data in the HS class, but only on a very small scale (within a single institution), and with an administrative fiat requiring that the delegation trees for the IN and HS trees be identical.

Even when using a different class, the data will still have to be stored in some RR type or another. This RR type can either be a "kitchen-sink record" or a new RR type. This implies that some other mechanism has to be applied as well, with implications detailed in other parts of this note.

3.5 Add a new Resource Record Type

When adding a new Resource Record type to the system, entities in four different roles have to be able to handle the new type:

1. There must be a way to insert the new resource records into the zone of the Primary Master name server. For some server implementations, the user interface only accepts record types which it understands (perhaps so that the implementation can attempt to validate the data). Other implementations allow the zone administrator to enter an integer for the resource record type code and the RDATA in Base64 or hexadecimal encoding (or even as raw data). RFC 3597 [RFC3597] specifies a standard generic encoding for this purpose.
2. A slave authoritative name server must be able to do a zone transfer, receive the data from some other authoritative name server, and serve data from the zone even though the zone includes records of unknown types. Historically, some implementations have had problems parsing stored copies of the zone file after restarting, but those problems have not been seen for a few years.
3. A full service resolver will cache the records which are responses to queries. As mentioned in RFC 3597 [RFC3597],there are various pitfalls where a recursive name server might end up having problems.
4. The application must be able to get the record with a new resource record type. The application itself may understand the RDATA, but the resolver library might not. Support for a generic interface for retrieving arbitrary DNS RR types has been a requirement since 1989 (see RFC 1123 [RFC1123] Section 6.1.4.2). Some stub resolver library implementations neglect to provide this functionality and cannot handle unknown RR types, but implementation of a new stub resolver library is not particularly difficult, and open source libraries that already provide this functionality are available.

4. Conclusion (why adding a new RR type is the answer)

By now, the astute reader will be wondering about how to use the issues presented so far. We will now attempt to clear up the reader's confusion by following the thought processes of a typical application designer who wishes to store stuff in the DNS, showing how such a designer almost inevitably hits upon the idea of just using TXT RR, and why this is a bad thing, and instead why a new RR type is to be allocated.

A typical application designer is not interested in the DNS for its own sake, but rather as a distributed database in which application data can be stored. As a result, the designer of a new application is usually looking for the easiest way to add whatever new data the application needs to the DNS in a way that naturally associates the data with a DNS name.

As explained in Section 3.4, using the DNS class system as an extension mechanism is not really an option, and in fact most users of the system don't even realize that the mechanism exists. As a practical matter, therefore any extension is likely to be within the IN class.

Adding a new RR type is the technically correct answer from the DNS protocol standpoint (more on this below), but doing so requires some DNS expertise, due to the issues listed in Section 3.5. As a result, this option is usually considered too hard. Note that according to RFC2929 [RFC2929], some types require IETF Consensus, while others only require a specification.

The application designer is thus left with the prospect of reusing some existing DNS type within the IN class, but when the designer looks at the existing types, almost all of them have well-defined semantics, none of which quite match the needs of the new application. This has not completely prevented proposals to reuse existing RR types in ways incompatible with their defined semantics, but it does tend to steer application designers away from this approach.

RR Type 40 was registered for the SINK record type. This RR Type was discussed in the DNSIND working group of the IETF, and it was decided at the 46th IETF to not move the I-D forward to become an RFC. Reason was the risk for large RR sets and the ability for application creators to use the SINK RR Type instead of registering a new RR Type.

Eliminating all of the above leaves the TXT RR type in the IN class. The TXT RDATA format is free form text, and there are no existing

semantics to get in the way. Furthermore, the TXT RR can obviously just be used as a bucket in which to carry around data to be used by some higher level parser, perhaps in some human readable programming or markup language. Thus, for many applications, TXT RRs are the "obvious" choice. Unfortunately, this conclusion, while understandable, is also wrong, for several reasons.

The first reason why TXT RRs are not well suited to such use is precisely the lack of defined semantics that make them so attractive. Arguably, the TXT RR is misnamed, and should have been called the Local Container record, because the lack of defined semantics means that a TXT RR means precisely what the data producer says it means. This is fine, so long as TXT RRs are being used by human beings or by private agreement between data producer and data consumer. However, it becomes a problem once one starts using them for standardized protocols in which there is no prior relationship between data producer and data consumer. Reason for this is that there is nothing to prevent collisions with some other incompatible use of TXT RRs. This is even worse than the general subtyping problem described in Section 3.1, because TXT RRs don't even have a standardized selector field in which to store the subtype. RFC1464 [RFC1464] tried, but it was not a success. At best a definition of a subtype is reduced to hoping that whatever scheme one has come up with will not accidently conflict with somebody else's subtyping scheme, and that it will not be possible to mis-parse one application's use of TXT RRs as data intended for a different application. Any attempt to come up with a standardized format within the TXT RR format would be at least fifteen years too late even if it were put into effect immediately.

Using one of the naming modifications discussed in Section 3.2 and Section 3.3 would address the subtyping problem, but each of these approaches brings in new problems of its own. The prefix approach (such as SRV RRs use) does not work well with wildcards, which is a particular problem for mail-related applications, since MX RRs are probably the most common use of DNS wildcards. The suffix approach doesn't have wildcard issues, but, as noted previously, it does have synchronization and update authorization issues, since it works by creating a second subtree in a different part of the global DNS name space.

The next reason why TXT RRs are not well suited to protocol use has to do with the limited data space available in a DNS message. As alluded to briefly in Section 3.1, typical DNS query traffic patterns involve a very large number of DNS clients sending queries to a relatively small number of DNS servers. Normal path MTU discovery schemes do little good here, because, from the server's perspective, there isn't enough repeat traffic from any one client for it to be worth retaining state. UDP-based DNS is an idempotent query, whereas

TCP-based DNS requires the server to keep state (in the form of TCP connection state, usually in the server's kernel) and roughly triples the traffic load. Thus, 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.

Subtyping schemes are therefore again problematic, because they produce larger RRSets than necessary, but verbose text encodings of data are also wasteful, since the data they hold can usually be represented more compactly in a resource record designed specifically to support the application's particular data needs. If the data that need to be carried are so large that there is no way to make them fit comfortably into the DNS regardless of encoding, it is probably better to move the data somewhere else, and just use the DNS as a pointer to the data, as with NAPTR.

5. Conclusion and Recommendation

Given the problems detailed in Section 4, it is worth reexamining the oft-jumped-to conclusion that specifying a new RR type is hard. Historically, this was indeed the case, but recent surveys suggest that support for unknown RR types [RFC3597] is now widespread, and that lack of support for unknown types is mostly an issue for relatively old software that would probably need to be upgraded in any case as part of supporting a new application. One should also remember that deployed DNS software today should support DNSSEC, and software recent enough to do so will have higher chance of being able to also support RFC3597.

Of all the issues detailed in Section 3.5, provisioning the data is in some respects the most difficult. The problem here is less the authoritative name servers themselves than the front-end systems used to enter (and perhaps validate) the data. Hand editing does not work well for maintenance of large zones, so some sort of tool is necessary, and the tool may not be tightly coupled to the name server implementation itself. Note, however, that this provisioning problem exists to some degree with any new form of data to be stored in the DNS, regardless of data format, RR type, or naming scheme. Adapting front-end systems to support a new RR type may be a bit more difficult than reusing an existing type, but this appears to be a minor difference in degree rather than a difference in kind.

Given the various issues described in this note, we believe that:
- o there is no magic solution which allows a completely painless addition of new data to the DNS, but
- o on the whole, the best solution is still to use the DNS type mechanism designed for precisely this purpose, and

o of all the alternate solutions, the "obvious" approach of using TXT RRs is almost certainly the worst.

6. IANA Considerations

This document does not require any IANA actions.

7. Security Considerations

DNS RRSets can be signed using DNSSEC. DNSSEC is almost certainly necessary for any application mechanism that stores authorization data in the DNS itself. DNSSEC signatures significantly increase the size of the messages transported, and because of this, the DNS message size issues discussed in Section 3.1 and Section 4 are more serious than they might at first appear.

Adding new RR types (as discussed in Section 3.5) might conceivably trigger bugs and other bad behavior in software which is not compliant with RFC 3597 [RFC3597], but most such software is old enough and insecure enough that it should be updated for other reasons in any case. Basic API support for retrieving arbitrary RR types has been a requirement since 1989 (see RFC 1123 [RFC1123]).

Any new protocol that proposes to use the DNS to store data used to make authorization decisions would be well advised not only to use DNSSEC but also to encourage upgrades to DNS server software recent enough not to be riddled with well-known exploitable bugs. Because of this, support for new RR Types will not be as hard as people might think at first.

8. References

8.1 Normative References

[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.

[RFC2535] Eastlake, D., "Domain Name System Security Extensions", RFC 2535, March 1999.

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

[RFC3597] Gustafsson, A., "Handling of Unknown DNS Resource Record (RR) Types", RFC 3597, September 2003.

8.2 Informative References

[RFC1123] Braden, R., "Requirements for Internet Hosts - Application and Support", STD 3, RFC 1123, October 1989.

[RFC2163] Allocchio, C., "Using the Internet DNS to Distribute MIXER Conformant Global Address Mapping (MCGAM)", RFC 2163, January 1998.

[RFC2606] Eastlake, D. and A. Panitz, "Reserved Top Level DNS Names", BCP 32, RFC 2606, June 1999.

[RFC2929] Eastlake, D., Brunner-Williams, E. and B. Manning, "Domain Name System (DNS) IANA Considerations", BCP 42, RFC 2929, September 2000.

[RFC3232] Reynolds, J., "Assigned Numbers: RFC 1700 is Replaced by an On-line Database", RFC 3232, January 2002.

[RFC3445] Massey, D. and S. Rose, "Limiting the Scope of the KEY Resource Record (RR)", RFC 3445, December 2002.

[RFC3761] Faltstrom, P. and M. Mealling, "The E.164 to Uniform Resource Identifiers (URI) Dynamic Delegation Discovery System (DDDS) Application (ENUM)", RFC 3761, April 2004.

[wcardclarify]
Halley, B. and E. Lewis, "draft-ietf-dnsext-wcard-clarify-05.txt, The Role of Wild Card Domains in the Domain Name System, work in progress", September 2003.

Authors' Addresses

Patrik Faltstrom
IAB

EMail: paf@cisco.com

Rob Austein
IAB

EMail: sra@isc.org

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.