Network Working Group
Request for Comments: 5507
Category: Informational

IAB
P. Faltstrom, Ed.
R. Austein, Ed.
P. Koch, Ed.

April 2009

Design Choices When Expanding the DNS

Status of This Memo

Copyright Notice

Abstract

This note discusses how to extend the DNS with new data for a new application. DNS extension discussions too often focus on reuse of the TXT Resource Record Type. This document lists different mechanisms to extend the DNS, and concludes that the use of a new DNS Resource Record 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 Resource Records) and data that have to do with mappings between domain names and IP addresses (A, AAAA, and PTR Resource Records). There are other categories as well, some of which are tied to specific applications like email (MX Resource Records), while others are generic Resource Record Types used to convey information for multiple protocols (SRV and NAPTR Resource Records).

When storing data in the DNS for a new application, the goal must be to store data in such a way that the application can query for the data it wants, while minimizing both the impact on existing applications and the amount of extra data transferred to the client. This implies that a number of design choices have to be made, where the most important is to ensure that a precise selection of what data to return must be made already in the query. A query consists of a triple: {Owner (or name), Resource Record Class, Resource Record Type}.

Historically, extending the DNS to store application data tied to a domain name has been done in different ways at different times. MX Resource Records were created as a new Resource Record Type specifically designed to support electronic mail. SRV records are a generic type that use a prefixing scheme in combination with a base domain name. NAPTR records add selection data inside the RDATA. It is clear that the methods used to add new data types to the DNS have 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.

This document talks extensively about use of DNS wildcards. Many people might think use of wildcards is not something that happens today. In reality though, wildcards are in use, especially for certain application-specific data such as MX Resource Records. Because of this, the choice has to be made with the existence of wildcards in mind.

Another overall issue that must be taken into account is what the new data in the DNS are to describe. In some cases, they might be completely new data. In other cases, they might be metadata tied to data that already exist in the DNS. Examples of new data are key information for the Secure SHell (SSH) Protocol and data used for authenticating the sender of email messages (metadata tied to MX Resource Records). If the new data are tied to data that already exist in the DNS, an analysis should be made as to whether having (for example) address records and SSH key information in different

DNS zones is a problem or if it is a bonus, and if it is a problem, whether the specification must require all of the related data to be in the same zone. One specific difference between having the records in the same zone or not has to do with maintenance of the records. If they are in the same zone, the same maintainer (from a DNS perspective) manages the two records. Specifically, they must be signed with the same DNSSEC keys if DNSSEC is in use.

This document does not talk about what one should store in the DNS. It also doesn't discuss whether the DNS should be used for service discovery, or whether the DNS should be used for storage of data specific to the service. In general, the DNS is a protocol that, apart from holding metadata that makes the DNS itself function (NS, SOA, DNSSEC Resource Record Types, etc.), only holds references to service locations (SRV, NAPTR, A, AAAA Resource Record Types) -- though there are exceptions, such as MX Resource Records.

2. Background

See RFC 5395 [RFC5395] for a brief summary of the 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 that update, clarify, and extend the base specification.

When composing a DNS query, the parameters used by the protocol are a {owner, class, type} triple. Every Resource Record matching such a triple is said to belong to the same Resource Record Set (RRSet), and the whole RRSet is always returned to the client that queries for it. Splitting an RRSet is a protocol violation (sending a partial RRSet, not truncating the DNS response), because it can result in coherency problems with the DNS caching mechanism. See Section 5 of [RFC2181] for more information.

Some discussions around extensions to the DNS include arguments around MTU size. Note that most discussions about DNS and MTU size are about the size of the whole DNS packet, not about the size of a single RRSet.

Almost all DNS query traffic is carried over UDP, where a DNS message must fit within a single UDP packet. DNS response messages are almost always larger than DNS query messages, so message size issues are almost always about responses, not queries. The base DNS specification limits DNS messages over UDP to 512 octets; EDNS0 [RFC2671] specifies a mechanism by which a client can signal its willingness to receive larger responses, but deployment of EDNS0 is not universal, in part because of firewalls that block fragmented UDP packets or EDNS0. If a response message won't fit in a single packet, the name server returns a truncated response, at which point the client may retry using TCP. DNS queries over TCP are not subject to this length limitation, but TCP imposes significantly higher per-query overhead on name servers than UDP. It is also the case that the policies in deployed firewalls far too often are such that they block DNS over TCP, so using TCP might not in reality be an option. There are also risks (although possibly small) that a change of routing while a TCP flow is open creates problems when the DNS servers are deployed in an anycast environment.

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 of Existing Resource Record Types

For a given query name, one might choose to have a single RRSet (all Resource Records sharing the same {owner, class, type} triple) shared by multiple applications, and have the different applications use selectors within the Resource Record 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 Resource Record Type.

Examples of subtyping include NAPTR Resource Records [RFC3761] and the original DNSSEC KEY Resource Record Type [RFC2535] (which was later updated by RFC 3445 [RFC3445], and obsoleted by RFC 4033 [RFC4033], RFC 4034 [RFC4034] and RFC 4035 [RFC4035]).

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 Resource Records in which it is interested. Furthermore, since DNSSEC signatures operate on complete RRSets, the entire RRSet must be re-signed if any Resource Record in it changes. As a result, each application that uses a subtyped Resource Record 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 today's deployed DNS is very high, having a substantial number of DNS messages fail over to TCP may cause the queried name servers to be overloaded by TCP overhead.

Because of the size limitations, using a subtyping scheme to list a large number of services for a single domain name risks triggering truncation and fallback to TCP, which may in turn force the zone administrator to announce only a subset of available services.

3.2. Add a Prefix to the Owner Name

By adding an application-specific prefix to a domain name, we get a different {owner, class, type} triple, and therefore a different RRSet. 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 (see also RFC 4592 [RFC4592]).

There have been proposals to deal with the problem that DNS wildcards are always terminal records. These proposals introduce an additional set of trade-offs that would need to be taken into account when assessing which extension mechanism to choose. Aspects of extra response time needed to perform the extra queries, costs of pre-calculation of possible answers, or the costs induced to the system as a whole come to mind. At the time of writing, none of these proposals has been published as Standards Track RFCs.

Even when a specific prefix is chosen, the data will still have to be stored in some Resource Record Type. This Resource Record Type can be either a new Resource Record Type or an existing Resource Record Type that has an appropriate format to store the data. One also might need some other selection mechanism, such as the ability to distinguish between the records in an RRSet, given they have the same Resource Record Type. Because of this, one needs to both register a unique prefix and define what Resource Record Type is to be used for this specific service.

If the record has some relationship with another record in the zone, the fact that the two records can be in different zones might have implications on the trust the application has in the records. For example:

```
example.com.      IN MX    10 mail.example.com.
_foo.example.com. IN X-BAR "metadata for the mail service"
```

In this example, the two records might be in two different zones, and as a result might be administered by two different organizations, and signed by two different entities when using DNSSEC. For these two reasons, using a prefix has recently become a very interesting solution for many protocol designers. In some cases, e.g., DomainKeys Identified Mail Signatures [RFC4871], TXT records have been used. In others, such as SRV, entirely new Resource Record Types have been added.

3.3. Add a Suffix to the Owner Name

Adding a suffix to a domain name changes the {owner, class, type} triple, and therefore the RRSet. In this case, since the query name can be set to exactly the data one wants, the size of the RRSet is minimized. The problem with adding a suffix is that it creates a parallel tree within the IN class. Further, there is 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.

One way of solving the administrative issues is by using the DNAME Resource Record Type specified in RFC 2672 [RFC2672].

Even when using a different name, the data will still have to be stored in some Resource Record Type that has an appropriate format to store the data. This implies that one might have to mix the prefix based selection mechanism with some other mechanism so that the right Resource Record can be found out of many in a potential larger RRSet.

In RFC 2163 [RFC2163] an infix token is inserted directly below the Top-Level Domain (TLD), but the result is equivalent to adding a suffix to the owner name (instead of creating a TLD, one is creating a second level domain).

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 Resource Records) 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 [Dyer87] 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. The use of the HS class for such storage of non-sensitive data was, over time, replaced by use of the Lightweight Directory Access Protocol (LDAP) [RFC4511].

Even when using a different class, the data will still have to be stored in some Resource Record Type that has an appropriate format.

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 at the Primary Master name server. For some server implementations, the user interface only accepts Resource Record Types that 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 Resource Record 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. Some implementations use an alternate

mechanism (e.g., LDAP) to transfer Resource Records in a zone, and are primarily used within corporate environments; in this case, name servers must be able to transfer new Resource Record Types using whatever mechanism is used. However, today this alternative mechanism may not support unknown Resource Record Types. Hence, in Internet environments, unknown Resource Record Types are supported, but in corporate environments they are problematic.

3. A caching resolver (most commonly a recursive name server) will cache the records that 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 RRSet 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 Resource Record Types has been a requirement since 1989 (see Section 6.1.4.2 of [RFC1123]). Some stub resolver library implementations neglect to provide this functionality and cannot handle unknown Resource Record Types, but implementation of a new stub resolver library is not particularly difficult, and open source libraries that already provide this functionality are available.

Historically, adding a new Resource Record Type has been very problematic. The review process has been cumbersome, DNS servers have not been able to handle new Resource Record Types, and firewalls have dropped queries or responses with Resource Record Types that are unknown to the firewall. This is, for example, one of the reasons the ENUM standard reuses the NAPTR Resource Record, a decision that today might have gone to creating a new Resource Record Type instead.

Today, there is a requirement that DNS software handle unknown Resource Record Types, and investigations have shown that software that is deployed, in general, does support it, except in some alternate mechanisms for transferring Resource Records such as LDAP, as noted above. Also, the approval process for new Resource Record Types has been updated [RFC5395] so the effort that is needed for various Resource Record Types is more predictable.

4. Zone Boundaries are Invisible to Applications

Regardless of the possible choices above, we have seen a number of cases where the application made assumptions about the structure of the namespace and the location where specific information resides. We take a small sidestep to argue against such approaches.

The DNS namespace is a hierarchy, technically speaking. However, this only refers to the way names are built from multiple labels. DNS hierarchy neither follows nor implies administrative hierarchy. Because of that, it cannot be assumed that data attached to a node in the DNS tree is valid for the whole subtree. Technically, there are zone boundaries partitioning the namespace, and administrative boundaries (or policy boundaries) may even exist elsewhere.

The false assumption has lead to an approach called "tree climbing", where a query that does not receive a positive response (either the requested RRSet was missing or the name did not exist) is retried by repeatedly stripping off the leftmost label (climbing towards the root) until the root domain is reached. Sometimes these proposals try to avoid the query for the root or the TLD level, but still this approach has severe drawbacks:

- Technically, the DNS was built as a query-response tool without any search capability [RFC3467]. Adding the search mechanism imposes additional burden on the technical infrastructure, in the worst case on TLD and root name servers.

- For reasons similar to those outlined in RFC 1535 [RFC1535], querying for information in a domain outside the control of the intended entity may lead to incorrect results and may also put security at risk. Finding the exact policy boundary is impossible without an explicit marker, which does not exist at present. At best, software can detect zone boundaries (e.g., by looking for SOA Resource Records), but some TLD registries register names starting at the second level (e.g., CO.UK), and there are various other "registry" types at second, third, or other level domains that cannot be identified as such without policy knowledge external to the DNS.

To restate, the zone boundary is purely a boundary that exists in the DNS for administrative purposes, and applications should be careful not to draw unwarranted conclusions from zone boundaries. A different way of stating this is that the DNS does not support inheritance, e.g., an MX RRSet for a TLD will not be valid for any subdomain of that particular TLD.

5. Why Adding a New Resource Record Type Is the Preferred Solution

By now, the astute reader might be wondering what conclusions to draw from 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 data in the DNS. We’ll show how such a designer almost inevitably hits upon the idea of just using a TXT Resource Record, why this is a bad thing, and why a new Resource Record Type should be allocated instead. We'll also explain how the reuse of an existing Resource Record, including TXT, can be made less harmful.

The overall problem with most solutions has to do with two main issues:

o No semantics to prevent collision with other use

o Space considerations in the DNS message

A typical application designer is not interested in the DNS for its own sake, but rather regards it 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 and does not require major changes to DNS servers.

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 Resource Record 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. Consequently, this option is often rejected. Note that according to RFC 5395 [RFC5395], some Types require IETF Consensus, while others only require a specification.

There is a drawback to defining new RR types that is worth mentioning. The Resource Record Type (RRTYPE) is a 16-bit value and hence is a limited resource. In order to prevent hoarding the registry has a review-based allocation policy [RFC5395]; however, this may not be sufficient if extension of the DNS by addition of new RR types takes up significantly and the registry starts nearing completion. In that case, the trade-offs with respect to choosing an extension mechanism may need to change.

The application designer is thus left with the prospect of reusing some existing DNS Types 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 from reusing existing Resource Record Types in ways incompatible with their defined semantics, but it does tend to steer application designers away from this approach.

For example, Resource Record Type 40 was registered for the SINK Resource Record Type. This Resource Record 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 because of the risk of encouraging application designers to use the SINK Resource Record Type instead of registering a new Resource Record Type, which would result in infeasibly large SINK RRsets.

Eliminating all of the above leaves the TXT Resource Record Type in the IN class. The TXT RDATA format is free form text, and there are no existing semantics to get in the way. Some attempts have been made, for example, in [DNSEXT-DNS-SD], to specify a structured format for TXT Resource Record Types, but no such attempt has reached RFC status. Furthermore, the TXT Resource Record 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 Resource Records are the "obvious" choice. Unfortunately, this conclusion, while understandable, is also problematic, for several reasons.

The first reason why TXT Resource Records are not well suited to such use is precisely what makes them so attractive: the lack of pre-defined common syntax or structure. As a result, each application that uses them creates its own syntax/structure, and that makes it difficult to reliably distinguish one application’s record from others, and for its parser to avoid problems when it encounters other TXT records.

Arguably, the TXT Resource Record is misnamed, and should have been called the Local Container record, because a TXT Resource Record means only what the data producer says it means. This is fine, so long as TXT Resource Records 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. If TXT records are used without one of the naming modifications discussed earlier (and in some cases even if one uses such naming mechanisms), there is nothing to prevent collisions with some other incompatible use of TXT Resource Records.

This is even worse than the general subtyping problem described in Section 3.1 because TXT Resource Records don’t even have a standardized selector field in which to store the subtype. RFC 1464 [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 Resource Records as data intended for a different application. Any attempt to impose a standardized format within the TXT Resource Record format would be at least fifteen years too late, even if it were put into effect immediately; at best, one can restrict the syntax that a particular application uses within a TXT Resource Record and accept the risk that unrelated TXT Resource Record uses will collide with it.

Using one of the naming modifications discussed in Section 3.2 and Section 3.3 would address the subtyping problem, (and have been used in combinations with reuse of TXT record, such as for the dns/txt lookup mechanism in Domain Keys Identified Mail (DKIM)) but each of these approaches brings in new problems of its own. The prefix approach (that for example SRV Resource Records use) does not work well with wildcards, which is a particular problem for mail-related applications, since MX Resource Records 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 namespace.

The next reason why TXT Resource Records 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 Resource 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.

6. Conclusion and Recommendation

Given the problems detailed in Section 5, it is worth reexamining the oft-jumped-to conclusion that specifying a new Resource Record Type is hard. Historically, this was indeed the case, but recent surveys suggest that support for unknown Resource Record Types [RFC3597] is now widespread in the public Internet, and because of that, the DNS infrastructure can handle new Resource Record Types. The lack of support for unknown Types remains an issue for relatively old provisioning software and in corporate environments.

Of all the issues detailed in Section 3.5, provisioning the data is in some respects the most difficult. Investigations with zone transfers show that the problem is less difficult for 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, Resource Record type (even if TXT Resource Record Types are in use), or naming scheme. Adapting front-end systems to support a new Resource Record 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:

- there is no magic solution that allows a completely painless addition of new data to the DNS, but
- on the whole, the best solution is still to use the DNS Resource Record Type mechanism designed for precisely this purpose, whenever possible, and
- of all the alternate solutions, the "obvious" approach of using TXT Resource Records for arbitrary names is almost certainly the worst, especially for the two reasons outlined above (lack of semantics and its implementations, and size leading to the need to use TCP).

7. Creating a New Resource Record Type

The process for creating a new Resource Record Type is specified in RFC 5395 [RFC5395].

8. 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. DNSSEC signatures significantly increase the size of the messages transported, and because of this, the DNS message size issues discussed in Sections 3.1 and 5 are more serious than they might at first appear.

Adding new Resource Record Types (as discussed in Section 3.5) can create two different kinds of problems: in the DNS software and in applications. In the DNS software, it might conceivably trigger bugs and other bad behavior in software that is not compliant with RFC 3597 [RFC3597], but most such DNS software is old enough and insecure enough that it should be updated for other reasons in any case. In applications and provisioning software, the changes for the new features that need the new data in the DNS can be updated to understand the structure of the new data format (regardless of whether a new Resource Record Type is used or some other mechanism is chosen). Basic API support for retrieving arbitrary Resource Record Types has been a requirement since 1989 [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.

9. Acknowledgements

This document has been created over a number of years, with input from many people. The question on how to expand and use the DNS is sensitive, and a document like this can not please everyone. The goal is instead to describe the architecture and tradeoffs, and make some recommendations about best practices.

People that have helped include: Dean Anderson, Mark Andrews, John Angelmo, Roy Badami, Dan Bernstein, Alex Bligh, Nathaniel Borenstein, Stephane Bortzmeyer, Brian Carpenter, Leslie Daigle, Elwyn Davies, Mark Delany, Richard Draves, Martin Duerst, Donald Eastlake, Robert Elz, Jim Fenton, Tony Finch, Jim Gilroy, Olafur Gudmundsson, Eric Hall, Phillip Hallam-Baker, Ted Hardie, Bob Hinden, Paul Hoffman, Geoff Houston, Christian Huitema, Johan Ihren, John Klensin, Ben Laurie, William Leibzon, John Levine, Edward Lewis, David MacQuigg, Allison Mankin, Bill Manning, David Meyer, Pekka Nikander, Mans Nilsson, Masataka Ohta, Douglas Otis, Michael Patton, Jonathan Rosenberg, Anders Rundgren, Miriam Sapiro, Carsten Strotmann, Pekka Savola, Chip Sharp, James Snell, Michael Thomas, Paul Vixie, Sam Weiler, Florian Weimer, Bert Wijnen, and Dan Wing.

10. IAB Members at the Time of This Writing

11. References

11.1. 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.

[RFC5395] Eastlake, D., "Domain Name System (DNS) IANA Considerations", BCP 42, RFC 5395, November 2008.

11.2. Informative References

[DNSEXT-DNS-SD] Cheshire, S. and M. Krochmal, "DNS-Based Service Discovery", Work in Progress, September 2008.

[Dyer87] Dyer, S. and F. Hsu, "Hesiod, Project Athena Technical Plan - Name Service", Version 1.9, April 1987.

[RFC1123]	Braden, R., "Requirements for Internet Hosts - Application and Support", STD 3, RFC 1123, October 1989.
[RFC1535]	Gavron, E., "A Security Problem and Proposed Correction With Widely Deployed DNS Software", RFC 1535, October 1993.
[RFC2163]	Allocchio, C., "Using the Internet DNS to Distribute MIXER Conformant Global Address Mapping (MCGAM)", RFC 2163, January 1998.
[RFC2181]	Elz, R. and R. Bush, "Clarifications to the DNS Specification", RFC 2181, July 1997.
[RFC2672]	Crawford, M., "Non-Terminal DNS Name Redirection", RFC 2672, August 1999.
[RFC3445]	Massey, D. and S. Rose, "Limiting the Scope of the KEY Resource Record (RR)", RFC 3445, December 2002.
[RFC3467]	Klensin, J., "Role of the Domain Name System (DNS)", RFC 3467, February 2003.
[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.
[RFC4033]	Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "DNS Security Introduction and Requirements", RFC 4033, March 2005.
[RFC4034]	Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "Resource Records for the DNS Security Extensions", RFC 4034, March 2005.
[RFC4035]	Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "Protocol Modifications for the DNS Security Extensions", RFC 4035, March 2005.
[RFC4511]	Sermersheim, J., "Lightweight Directory Access Protocol (LDAP): The Protocol", RFC 4511, June 2006.
[RFC4592]	Lewis, E., "The Role of Wildcards in the Domain Name System", RFC 4592, July 2006.

[RFC4871] Allman, E., Callas, J., Delany, M., Libbey, M., Fenton, J., and M. Thomas, "DomainKeys Identified Mail (DKIM) Signatures", RFC 4871, May 2007.

Authors' Addresses

Internet Architecture Board

EMail: iab@iab.org

Patrik Faltstrom (editor)

EMail: paf@cisco.com

Rob Austein (editor)

EMail: sra@isc.org

Peter Koch (editor)

EMail: pk@denic.de