

Network Working Group
Internet-Draft
Updates: 7050 (if approved)
Intended status: Standards Track
Expires: September 20, 2020

S. Cheshire
Apple Inc.
D. Schinazi
Google LLC
March 19, 2020

Special Use Domain Name 'ipv4only.arpa'
draft-cheshire-sudn-ipv4only-dot-arpa-17

Abstract

The specification for how a client discovers its local network's NAT64 prefix (RFC7050) defines the special name 'ipv4only.arpa' for this purpose, but in its Domain Name Reservation Considerations section that specification indicates that the name actually has no particularly special properties that would require special handling, and does not request IANA to record the name in the Special-Use Domain Names registry.

Consequently, despite the well articulated special purpose of the name, 'ipv4only.arpa' was not recorded in the Special-Use Domain Names registry as a name with special properties.

This document describes the special treatment required, formally declares the special properties of the name, adds similar declarations for the corresponding reverse mapping names, and updates RFC7050.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 20, 2020.

Copyright Notice

Copyright (c) 2020 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
2. Specialness of 'ipv4only.arpa'	3
3. Consequences of 'ipv4only.arpa' not being declared special .	4
4. Remedies	6
5. Security Considerations	8
6. IANA Considerations	10
7. Domain Name Reservation Considerations	10
8. Acknowledgements	17
9. References	17
Appendix A. Example BIND 9 Configuration	19
Authors' Addresses	20

1. Introduction

The specification for how a client discovers its local network's NAT64 prefix [RFC7050] defines the special name 'ipv4only.arpa' for this purpose, but in its Domain Name Reservation Considerations section that specification indicates that the name actually has no particularly special properties that would require special handling, and does not request IANA to record the name in the Special-Use Domain Names registry [SUDN].

Consequently, despite the well articulated special purpose of the name, 'ipv4only.arpa' was not recorded in the Special-Use Domain Names registry [SUDN] as a name with special properties.

This omission was discussed in the Special-Use Domain Names Problem Statement [RFC8244].

As a result of this omission, in cases where software needs to give this name special treatment in order for it to work correctly, there was no clear mandate authorizing software authors to implement that special treatment. Software implementers were left with the choice between not implementing the special behavior necessary for the name queries to work correctly, or implementing the special behavior and being accused of being noncompliant with some RFC.

This document describes the special treatment required, formally declares the special properties of the name, and adds similar declarations for the corresponding reverse mapping names.

1.1. Conventions and Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

2. Specialness of 'ipv4only.arpa'

The hostname 'ipv4only.arpa' is peculiar in that it was never intended to be treated like a normal hostname.

A typical client never has any reason to look up the IPv4 address records for 'ipv4only.arpa'. No normal user is ever trying to view a web site hosted at that domain name, or trying to send email to an email address at that domain name. The name 'ipv4only.arpa' is already known, by IETF specification [RFC7050], to have exactly two

IPv4 address records, 192.0.0.170 and 192.0.0.171. No client ever has to look up the name in order to learn those two addresses.

In contrast, clients often look up the IPv6 AAAA address records for 'ipv4only.arpa', which is contrary to general DNS expectations, given that it is already known, by IETF specification [RFC7050], that 'ipv4only.arpa' is an IPv4-only name, which has no IPv6 AAAA address records. And yet, clients expect to receive, and do in fact receive, positive answers for these IPv6 AAAA address records that apparently should not exist.

This odd query behaviour comes not because clients are using DNS to learn legitimate answers from the name's legitimate authoritative server. Instead, the DNS protocol has, in effect, been co-opted as an improvised client-to-middlebox communication protocol, to look for a DNS64/NAT64 [RFC6146] [RFC6147] gateway and, if one is present, to request that it disclose the prefix it is using for IPv6 address synthesis.

This use of specially crafted DNS queries as an improvised client-to-middlebox communication protocol has a number of specific consequences, outlined below, which client software needs to take into account if the queries are to produce the desired results, particularly when used on a multi-homed host, or when a VPN tunnel is in use. The name 'ipv4only.arpa' is most definitely a special name, and needs to be listed in IANA's registry along with other DNS names that have special uses [SUDN].

3. Consequences of 'ipv4only.arpa' not being declared special

As a result of the original specification [RFC7050] not formally declaring 'ipv4only.arpa' to have special properties, there was no clear mandate for DNS software to treat this name specially. In particular, this lack of mandate for special treatment is relevant (a) to the name resolution APIs and libraries on client devices, and (b) to DNS64 [RFC6147] implementations. These two aspects are discussed in more detail below.

3.1. Consequences for Name Resolution APIs and Libraries

A serious problem can occur with DNS64/NAT64 when a device is configured to use a recursive resolver other than the one it learned from the network.

A device joining a NAT64 network will learn the recursive resolver recommended for that network, typically via IPv6 Router Advertisement Options for DNS Configuration [RFC8106] or via DNS Configuration options for DHCPv6 [RFC3646]. On a NAT64 network it is essential

that the client use the DNS64 recursive resolver recommended for that network, since only that recursive resolver can be relied upon to know the appropriate prefix(es) to use for synthesizing IPv6 addresses that will be acceptable to that NAT64 gateway.

However, it is becoming increasingly common for users to manually override their default DNS configuration because they wish to use some other public recursive resolver on the Internet, which may offer better speed, better reliability, or better privacy than the local network's default recursive resolver. At the time of writing, examples of widely known public recursive resolver services include Cloudflare Public DNS [DNS1], Google Public DNS [DNS8], and Quad9 [DNS9].

Another common scenario is the use of corporate or personal VPN client software. Both for privacy reasons, and also because the local network's recursive resolver will typically be unable to provide answers for the company's private internal host names, so VPN client software overrides the local network's default configuration, to divert some or all DNS requests to the company's own private internal recursive resolver, reached through the VPN tunnel. As with the case described above of public recursive resolver services, the company's private internal recursive resolver cannot be expected to be able to synthesize IPv6 addresses correctly for use with the local network's NAT64 gateway, because the company's private internal recursive resolver is unlikely to be aware of the NAT64 prefix in use on the NAT64 network to which the client device is currently attached. It is clear that a single recursive resolver cannot meet both needs. The local network's recursive resolver cannot give answers for some company's private internal host names, and some company's private internal recursive resolver cannot give correctly synthesized IPv6 addresses suitable for the local network's NAT64 gateway.

Note that multiple NAT64 services may be simultaneously available to a client. For example, the local network may provide NAT64 service (to allow a IPv6-only client device to access IPv4-only Internet services), while at the same time a corporate VPN may also provide NAT64 service (to allow a client connecting via an IPv6-only VPN tunnel to access IPv4-only corporate services). The NAT64 address synthesis prefixes for the two NAT64 services may be different. In this case it is essential that the NAT64 address synthesis prefix used on the local network be the prefix learned from the local network's recursive resolver, and the NAT64 address synthesis prefix used on the VPN tunnel be the prefix learned from the VPN tunnel's recursive resolver.

The conflict here arises because DNS is being used for two unrelated purposes. The first purpose is retrieving data from a (nominally) global database -- generally retrieving the IP address(es) associated with a hostname. The second purpose is using the DNS protocol as a middlebox communication protocol, to interrogate the local network infrastructure to discover the IPv6 prefix(es) in use by the local NAT64 gateway for address synthesis.

3.2. Consequences for DNS64 Implementations

As a result of there being no mandate for special treatment, queries for 'ipv4only.arpa' had to be handled normally, resulting in DNS64 gateways performing unnecessary IPv6 address record queries (DNS qtype "AAAA", always returning negative responses) and IPv4 address record queries (DNS qtype "A", always returning the same positive responses) to the authoritative 'arpa' name servers.

Having DNS64 gateways around the world issue these queries generated additional load on the authoritative 'arpa' name servers, which was redundant when the name 'ipv4only.arpa' is defined, by IETF specification [RFC7050], to have exactly two IPv4 address records, 192.0.0.170 and 192.0.0.171, and no other IPv4 or IPv6 address records.

Also, at times, for reasons that remain unclear, the authoritative 'arpa' name servers have been observed to be slow or unresponsive. The failures of these 'ipv4only.arpa' queries result in unnecessary failures of the DNS64 gateways and of the client devices that depend on them for DNS64 [RFC6147] address synthesis.

Even when the authoritative 'arpa' name servers are operating correctly, having to perform an unnecessary query to obtain an answer that is already known in advance can add precious milliseconds of delay, affecting user experience on the client devices waiting for those synthesized replies.

4. Remedies

This document leverages operational experience to update the Domain Name Reservation Considerations [RFC6761] section of the earlier specification [RFC7050] with one that more accurately lists the actual special properties of the name 'ipv4only.arpa', so that software can legitimately implement the correct behavior necessary for better performance, better reliability, and correct operation.

These changes affect two bodies of software, (a) the name resolution APIs and libraries on client devices, and (b) DNS64 implementations.

The new special rules specified in this document for name resolution APIs and libraries state how they should select which recursive resolver to query to learn the IPv6 address synthesis prefix in use on a particular physical or virtual interface. Specifically: When querying for the name 'ipv4only.arpa', name resolution APIs and libraries should use the recursive resolver recommended by the network for the interface in question, rather than a recursive resolver configured manually, a recursive resolver configured by VPN software, or a full-service recursive resolver running on the local host. Superficially this might seem like a security issue, since the user might have explicitly configured the particular DNS resolver they wish to use, and rather than using that, the name resolution code ignores the user's stated preference and uses untrusted input received from the network instead. However, the 'ipv4only.arpa' query is not really a DNS query in the usual sense; even though it may look like a DNS query, it is actually an improvised client-to-middlebox communication protocol in disguise. For NAT64 to work at all, it is necessary for the interface on which NAT64 translation is being performed to tell the host the address of the DNS64 recursive resolver the host must use to learn the NAT64 prefix being used by that NAT64. This is typically done via IPv6 Router Advertisement Options for DNS Configuration [RFC8106] or via DNS Configuration options for DHCPv6 [RFC3646].

The new special rules specified in this document for DNS64 implementations recommend that they avoid performing run-time network queries for values that are known to be fixed by specification.

A useful property of the way NAT64 Prefix Discovery [RFC7050] was originally specified was that it allowed for incremental deployment. Even if existing DNS64 gateways, that were unaware of the special 'ipv4only.arpa' name, were already deployed, once IANA created the appropriate 'ipv4only.arpa' records, clients could begin to use the new facility immediately. Clients could send their special queries for 'ipv4only.arpa' to an ipv4only-unaware DNS64 gateway, and (after a query to IANA's servers) the DNS64 gateway would then generate the correct synthesized response.

While this was a useful transition strategy to enable rapid adoption, it is not the ideal end situation. For better performance, better reliability, and lower load in IANA's servers, it is preferable for DNS64 gateways to be aware of the special 'ipv4only.arpa' name so that they can avoid issuing unnecessary queries. Network operators who wish to provide reliable, high performance service to their customers are motivated to prefer DNS64 gateways that recognize the special 'ipv4only.arpa' name and apply the appropriate optimizations.

5. Security Considerations

One of the known concerns with DNS64 is that it conflicts with DNSSEC. If DNSSEC is used to assert cryptographically that a name has no IPv6 AAAA records, then this interferes with using DNS64 address synthesis to assert that those nonexistent IPv6 AAAA records do exist.

Section 3 of the DNS64 specification [RFC6147] discusses this:

... DNS64 receives a query with the DO bit set and the CD bit set. In this case, the DNS64 is supposed to pass on all the data it gets to the query initiator. This case will not work with DNS64, unless the validating resolver is prepared to do DNS64 itself.

The NAT64 Prefix Discovery specification [RFC7050] provides the mechanism for the query initiator to learn the NAT64 prefix so that it can do its own validation and DNS64 synthesis as described above. With this mechanism the client can (i) interrogate the local DNS64/NAT64 gateway with an 'ipv4only.arpa' query to learn the IPv6 address synthesis prefix, (ii) query for the (signed) IPv4 address records itself, and validate the response, and then (iii) perform its own IPv6 address synthesis locally, combining the IPv6 address synthesis prefix learned from the local DNS64/NAT64 gateway with the validated DNSSEC-signed data learned from the global Domain Name System.

It is conceivable that over time, if DNSSEC adoption continues to grow, the majority of clients could move to this validate-and-synthesize-locally model, which reduces the DNS64 machinery to the vestigial role of simply responding to the 'ipv4only.arpa' query to report the local IPv6 address synthesis prefix. In no case does the client care what answer(s) the authoritative 'arpa' name servers might give for that query. The 'ipv4only.arpa' query is being used purely as a local client-to-middlebox communication message.

This approach is even more attractive if it does not create an additional dependency on the authoritative 'arpa' name servers to answer a query that is unnecessary because the DNS64/NAT64 gateway already knows the answer before it even issues the query. Avoiding this unnecessary query improves performance and reliability for the client, and reduces unnecessary load for the authoritative 'arpa' name servers.

Hard-coding the known answers for 'ipv4only.arpa' IPv4 address record queries (DNS qtype "A") in recursive resolvers also reduces the risk of malicious devices intercepting those queries and returning incorrect answers. Because the 'ipv4only.arpa' zone has to be an

insecure delegation (see below) DNSSEC cannot be used to protect these answers from tampering by malicious devices on the path.

With respect to the question of whether 'ipv4only.arpa' should be a secure or insecure delegation, we need to consider two paths of information flow through the network: The path from the server authoritative for 'ipv4only.arpa' to the DNS64 recursive resolver, and the path from the DNS64 recursive resolver to the ultimate client.

The path from the authoritative server to the DNS64 recursive resolver (queries for IPv4 address records) need not be protected by DNSSEC, because the DNS64 recursive resolver already knows, by specification, what the answers are. In principle, if this were a secure delegation, and 'ipv4only.arpa' were a signed zone, then the path from the authoritative server to the DNS64 recursive resolver would still work, but DNSSEC is not necessary here. Run-time cryptographic signatures are not needed to verify compile-time constants. Validating the signatures could only serve to introduce potential failures into the system for minimal benefit.

The path from the DNS64 recursive resolver to the ultimate client (queries for IPv6 address records) *cannot* be protected by DNSSEC, because the DNS64 recursive resolver is synthesizing IPv6 address answers, and does not possess the DNSSEC secret key required to sign those answers.

Consequently, the 'ipv4only.arpa' zone **MUST** be an insecure delegation, to give DNS64/NAT64 gateways the freedom to synthesize answers to those queries at will, without the answers being rejected by DNSSEC-capable resolvers. DNSSEC-capable resolvers that follow this specification **MUST NOT** attempt to validate answers received in response to queries for the IPv6 AAAA address records for 'ipv4only.arpa'. Note that the name 'ipv4only.arpa' has no use outside of being used for this special DNS pseudo-query used to learn the DNS64/NAT64 address synthesis prefix, so the lack of DNSSEC security for that name is not a problem.

The original NAT64 Prefix Discovery specification [RFC7050] stated, incorrectly:

A signed "ipv4only.arpa." allows validating DNS64 servers (see [RFC6147] Section 3, Case 5, for example) to detect malicious AAAA resource records. Therefore, the zone serving the well-known name has to be protected with DNSSEC.

This document updates the previous specification [RFC7050] to correct that error. The 'ipv4only.arpa' zone **MUST** be an insecure delegation.

6. IANA Considerations

[Once published] IANA has created an insecure delegation for 'ipv4only.arpa' to allow DNS64 recursive resolvers to create synthesized AAAA answers within that zone.

IANA has recorded the following names in the Special-Use Domain Names registry [SUDN]:

```
ipv4only.arpa.  
170.0.0.192.in-addr.arpa.  
171.0.0.192.in-addr.arpa.
```

IANA has recorded the following IPv4 addresses in the IPv4 Special-Purpose Address Registry [SUv4]:

```
192.0.0.170  
192.0.0.171
```

7. Domain Name Reservation Considerations

7.1. Special Use Domain Name 'ipv4only.arpa'

The name 'ipv4only.arpa' is defined, by IETF specification [RFC7050], to have two IPv4 address records with rdata 192.0.0.170 and 192.0.0.171.

When queried via a DNS64 [RFC6147] recursive resolver, the name 'ipv4only.arpa' is also defined to have IPv6 AAAA records, with rdata synthesized from a combination of the NAT64 IPv6 prefix(es) and the IPv4 addresses 192.0.0.170 and 192.0.0.171. This can return more than one pair of IPv6 addresses if there are multiple NAT64 prefixes.

The name 'ipv4only.arpa' has no other IPv4 or IPv6 address records. There are no subdomains of 'ipv4only.arpa'. All names falling below 'ipv4only.arpa' are defined to be nonexistent (NXDOMAIN).

The name 'ipv4only.arpa' is special to

- (a) client software wishing to perform DNS64 address synthesis,
- (b) APIs responsible for retrieving the correct information, and
- (c) the DNS64 recursive resolver responding to such requests.

These three considerations are listed in items 2, 3 and 4 below:

1. Normal users should never have reason to encounter the 'ipv4only.arpa' domain name. If they do, they should expect queries for 'ipv4only.arpa' to result in the answers required by the specification [RFC7050]. Normal users have no need to know that 'ipv4only.arpa' is special.

2. Application software may explicitly use the name 'ipv4only.arpa' for DNS64/NAT64 address synthesis, and expect to get the answers required by the specification [RFC7050]. If application software encounters the name 'ipv4only.arpa' in the normal course of handling user input, the application software should resolve that name as usual and need not treat it in any special way.
3. Name resolution APIs and libraries MUST recognize 'ipv4only.arpa' as special and MUST give it special treatment.

Learning a network's NAT64 prefix is by its nature an interface-specific operation, and the special DNS query used to learn this interface-specific NAT64 prefix MUST be sent to the DNS recursive resolver address(es) the client learned via the configuration machinery for that specific client interface. The NAT64 prefix is a per-interface property, not a per-device property.

Regardless of any manual client DNS configuration, DNS overrides configured by VPN client software, or any other mechanisms that influence the choice of the client's recursive resolver address(es) (including client devices that run their own local recursive resolver and use the loopback address as their configured recursive resolver address) all queries for 'ipv4only.arpa' and any subdomains of that name MUST be sent to the recursive resolver learned from the network interface in question via IPv6 Router Advertisement Options for DNS Configuration [RFC8106], DNS Configuration options for DHCPv6 [RFC3646], or other configuration mechanisms. Because DNS queries for 'ipv4only.arpa' are actually a special middlebox communication protocol, it is essential that they go to the correct middlebox for the interface in question, and failure to honor this requirement would cause failure of the NAT64 Prefix Discovery mechanism [RFC7050].

One implication of this is that, on multi-homed devices (devices that allow more than one logical or physical IP interface to be active at the same time, e.g., cellular data and Wi-Fi, or one physical interface plus a VPN connection), clients MUST use interface-aware name resolution APIs. On different (logical or physical) interfaces, different DNS64 answers may be received, and DNS64 answers are only valid for the interface on which they were received. On multi-homed devices (including devices that support VPN), name resolution APIs that do not include interface parameters will not work reliably with NAT64. On single-homed devices, interface-unaware name resolution APIs are acceptable since when only one interface can be active at a time there is no need to specify an interface.

DNSSEC-capable resolvers MUST NOT attempt to validate answers received in response to queries for the IPv6 AAAA address records for 'ipv4only.arpa', since, by definition, any such answers are generated by the local network's DNS64/NAT64 gateway, not the authoritative server responsible for that name.

4. For the purposes of this section, recursive resolvers fall into two categories. The first category is traditional recursive resolvers, which includes **forwarding** recursive resolvers, as commonly implemented in residential home gateways, and **iterative** recursive resolvers, as commonly deployed by ISPs. More information on these terms can be found in DNS Terminology [RFC8499]. The second category is DNS64 recursive resolvers, whose purpose is to synthesize IPv6 address records. These may be **forwarding** DNS64 recursive resolvers or **iterative** DNS64 recursive resolvers, and they work in partnership with a companion NAT64 gateway to communicate the appropriate NAT64 address synthesis prefix to clients.

Traditional forwarding recursive resolvers SHOULD NOT recognize 'ipv4only.arpa' as special or give that name, or subdomains of that name, any special treatment. The rationale for this is that a traditional forwarding recursive resolver, such as built in to a residential home gateway, may itself be downstream of a DNS64 recursive resolver. Passing through the 'ipv4only.arpa' queries to the upstream DNS64 recursive resolver will allow the correct NAT64 prefix to be discovered.

Traditional iterative recursive resolvers that are not explicitly configured to synthesize IPv6 prefixes on behalf of a companion NAT64 gateway need not recognize 'ipv4only.arpa' as special or take any special action.

Forwarding or iterative recursive resolvers that have been explicitly configured to perform DNS64 address synthesis in support of a companion NAT64 gateway (i.e, "DNS64 recursive resolvers") MUST recognize 'ipv4only.arpa' as special. The authoritative name servers for 'ipv4only.arpa' cannot be expected to know the local network's NAT64 address synthesis prefix, so consulting the authoritative name servers for IPv6 address records for 'ipv4only.arpa' is futile. All DNS64 recursive resolvers MUST recognize 'ipv4only.arpa' (and all of its subdomains) as special, and MUST NOT attempt to look up NS records for 'ipv4only.arpa', or otherwise query authoritative name servers in an attempt to resolve this name. Instead, DNS64 recursive resolvers MUST act as authoritative for this zone, by generating immediate responses for all queries for 'ipv4only.arpa' (and any subdomain of 'ipv4only.arpa'), with the

one exception of queries for the DS record. Queries for the DS record are resolved the usual way to allow a client to securely verify that the 'ipv4only.arpa' zone has an insecure delegation. Note that this exception is not expected to receive widespread usage, since any client compliant with this specification already knows that 'ipv4only.arpa' is an insecure delegation and will not attempt DNSSEC validation for this name.

DNS64 recursive resolvers MUST generate the 192.0.0.170 and 192.0.0.171 responses for IPv4 address queries (DNS qtype "A"), the appropriate synthesized IPv6 address record responses for IPv6 address queries (DNS qtype "AAAA"), and a negative ("no error no answer") response for all other query types except DS.

For all subdomains of 'ipv4only.arpa', DNS64 recursive resolvers MUST generate immediate NXDOMAIN responses. All names falling below 'ipv4only.arpa' are defined to be nonexistent.

An example configuration for BIND 9 showing how to achieve the desired result is given in Appendix A.

Note that this is **not** a locally served zone in the usual sense of that term [RFC6303] because this rule applies **only** to DNS64 recursive resolvers, not to forwarding DNS recursive resolvers.

5. Authoritative name server software need not recognize 'ipv4only.arpa' as special or handle it in any special way.
6. Generally speaking, operators of authoritative name servers need not know anything about the name 'ipv4only.arpa', just as they do not need to know anything about any other names they are not responsible for. Only the administrators of the 'arpa' namespace need to be aware of this name's purpose and how it should be configured. In particular, 'ipv4only.arpa' MUST have the required records, and MUST be an insecure delegation, to allow DNS64 recursive resolvers to create synthesized AAAA answers within that zone. Making the 'ipv4only.arpa' zone a secure delegation would make it impossible for DNS64 recursive resolvers to create synthesized AAAA answers that will be accepted by DNSSEC validating clients, thereby defeating the entire purpose of the 'ipv4only.arpa' name.
7. DNS Registries/Registrars need not know anything about the name 'ipv4only.arpa', just as they do not need to know anything about any other name they are not responsible for.

7.2. Names '170.0.0.192.in-addr.arpa' and '171.0.0.192.in-addr.arpa'

Since the IPv4 addresses 192.0.0.170 and 192.0.0.171 are defined to be special, and are listed in the IPv4 Special-Purpose Address Registry [SUv4], the corresponding reverse mapping names in the in-addr.arpa domain are similarly special.

The name '170.0.0.192.in-addr.arpa' is defined, by IETF specification [RFC7050], to have only one DNS record, type PTR, with rdata 'ipv4only.arpa'.

The name '171.0.0.192.in-addr.arpa' is defined, by IETF specification [RFC7050], to have only one DNS record, type PTR, with rdata 'ipv4only.arpa'.

There are no subdomains of '170.0.0.192.in-addr.arpa' or '171.0.0.192.in-addr.arpa'. All names falling below these names are defined to be nonexistent (NXDOMAIN).

Practically speaking these two names are rarely used, but to the extent that they may be, they are special only to resolver APIs and libraries, as described in item 3 below:

1. Normal users should never have reason to encounter these two reverse mapping names. However, if they do, queries for these reverse mapping names should return the expected answer 'ipv4only.arpa'. Normal users have no need to know that these reverse mapping names are special.
2. Application software SHOULD NOT recognize these two reverse mapping names as special, and SHOULD NOT treat them differently. For example, if the user were to issue the Unix command "host 192.0.0.170" then the "host" command should call the name resolution API or library as usual and display the result that is returned.
3. Name resolution APIs and libraries SHOULD recognize these two reverse mapping names as special and generate the required responses locally. For the names '170.0.0.192.in-addr.arpa' and '171.0.0.192.in-addr.arpa' PTR queries yield the result 'ipv4only.arpa'; all other query types yield a negative ("no error no answer") response. For all subdomains of these two reverse mapping domains, all queries yield an NXDOMAIN response. All names falling below these two reverse mapping domains are defined to be nonexistent.

This local self-contained generation of these responses is to avoid placing unnecessary load on the authoritative 'in-addr.arpa' name servers.

4. Recursive resolvers SHOULD NOT recognize these two reverse mapping names as special and SHOULD NOT, by default, give them any special treatment.
5. Authoritative name server software need not recognize these two reverse mapping names as special or handle them in any special way.
6. Generally speaking, most operators of authoritative name servers need not know anything about these two reverse mapping names, just as they do not need to know anything about any other names they are not responsible for. Only the operators of the authoritative name servers for these two reverse mapping names need to be aware that these names are special, and require fixed answers specified by IETF specification.
7. DNS Registries/Registrars need not know anything about these two reverse mapping names, just as they do not need to know anything about any other name they are not responsible for.

7.2.1. ip6.arpa Reverse Mapping PTR Records

For all IPv6 addresses synthesized by a DNS64 recursive resolver, the DNS64 recursive resolver is responsible for synthesizing the appropriate 'ip6.arpa' reverse mapping PTR records too, if it chooses to provide reverse mapping PTR records. The same applies to the synthesized IPv6 addresses corresponding to the IPv4 addresses 192.0.0.170 and 192.0.0.171.

Generally a DNS64 recursive resolver synthesizes appropriate 'ip6.arpa' reverse mapping PTR records by extracting the embedded IPv4 address from the encoded IPv6 address, performing a reverse mapping PTR query for that IPv4 address, and then synthesizing a corresponding 'ip6.arpa' reverse mapping PTR record containing the same rdata.

In the case of synthesized IPv6 addresses corresponding to the IPv4 addresses 192.0.0.170 and 192.0.0.171, the DNS64 recursive resolver does not issue reverse mapping queries for those IPv4 addresses, but instead, according to rule 3 above, immediately returns the answer 'ipv4only.arpa'.

In the case of a client that uses the 'ipv4only.arpa' query to discover the IPv6 prefixes in use by the local NAT64 gateway, and then proceeds to perform its own address synthesis locally (which has benefits such as allowing DNSSEC validation), that client MUST also synthesize 'ip6.arpa' reverse mapping PTR records for those discovered prefix(es), according to the rules above: When a client's name resolution APIs and libraries receive a request to look up an 'ip6.arpa' reverse mapping PTR record for an address that falls within one of the discovered NAT64 address synthesis prefixes, the software extracts the embedded IPv4 address and then, for IPv4 addresses 192.0.0.170 and 192.0.0.171, returns the fixed answer 'ipv4only.arpa', and for all other IPv4 addresses performs a reverse mapping PTR query for the IPv4 address, and then synthesizes a corresponding 'ip6.arpa' reverse mapping PTR record containing the same rdata.

8. Acknowledgements

Thanks to Jouni Korhonen, Teemu Savolainen, and Dan Wing, for devising the NAT64 Prefix Discovery mechanism [RFC7050], and for their feedback on this document.

Thanks to Geoff Huston for his feedback on this document.

Thanks to Erik Kline for pointing out that the in-addr.arpa names are special too.

Thanks to Mark Andrews for conclusively pointing out the reasons why the 'ipv4only.arpa' zone must be an insecure delegation in order for the NAT64 Prefix Discovery mechanism [RFC7050] to work, and many other very helpful comments.

Thanks particularly to Lorenzo Colitti for an especially spirited hallway discussion at IETF 96 in Berlin, which lead directly to significant improvements in how this document presents the issues.

Thanks to Scott Bradner, Bernie Volz, Barry Leiba, Mirja Kuehlewind, Suresh Krishnan, Benjamin Kaduk, Roman Danyliw, Eric Vyncke and the other IESG reviewers for their thoughtful feedback.

Thanks to Dave Thaler and Warren Kumari for generously helping shepherd this document through the publication process.

9. References

9.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC3646] Droms, R., Ed., "DNS Configuration options for Dynamic Host Configuration Protocol for IPv6 (DHCPv6)", RFC 3646, DOI 10.17487/RFC3646, December 2003, <<https://www.rfc-editor.org/info/rfc3646>>.
- [RFC6146] Bagnulo, M., Matthews, P., and I. van Beijnum, "Stateful NAT64: Network Address and Protocol Translation from IPv6 Clients to IPv4 Servers", RFC 6146, DOI 10.17487/RFC6146, April 2011, <<https://www.rfc-editor.org/info/rfc6146>>.

- [RFC6147] Bagnulo, M., Sullivan, A., Matthews, P., and I. van Beijnum, "DNS64: DNS Extensions for Network Address Translation from IPv6 Clients to IPv4 Servers", RFC 6147, DOI 10.17487/RFC6147, April 2011, <<https://www.rfc-editor.org/info/rfc6147>>.
- [RFC6761] Cheshire, S. and M. Krochmal, "Special-Use Domain Names", RFC 6761, DOI 10.17487/RFC6761, February 2013, <<https://www.rfc-editor.org/info/rfc6761>>.
- [RFC7050] Savolainen, T., Korhonen, J., and D. Wing, "Discovery of the IPv6 Prefix Used for IPv6 Address Synthesis", RFC 7050, DOI 10.17487/RFC7050, November 2013, <<https://www.rfc-editor.org/info/rfc7050>>.
- [RFC8106] Jeong, J., Park, S., Beloeil, L., and S. Madanapalli, "IPv6 Router Advertisement Options for DNS Configuration", RFC 8106, DOI 10.17487/RFC8106, March 2017, <<https://www.rfc-editor.org/info/rfc8106>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.

9.2. Informative References

- [RFC6303] Andrews, M., "Locally Served DNS Zones", BCP 163, RFC 6303, DOI 10.17487/RFC6303, July 2011, <<https://www.rfc-editor.org/info/rfc6303>>.
- [RFC8244] Lemon, T., Droms, R., and W. Kumari, "Special-Use Domain Names Problem Statement", RFC 8244, DOI 10.17487/RFC8244, October 2017, <<https://www.rfc-editor.org/info/rfc8244>>.
- [RFC8499] Hoffman, P., Sullivan, A., and K. Fujiwara, "DNS Terminology", BCP 219, RFC 8499, DOI 10.17487/RFC8499, January 2019, <<https://www.rfc-editor.org/info/rfc8499>>.
- [SUDN] "Special-Use Domain Names Registry", <<https://www.iana.org/assignments/special-use-domain-names/>>.
- [SUv4] "IANA IPv4 Special-Purpose Address Registry", <<https://www.iana.org/assignments/iana-ipv4-special-registry/>>.
- [DNS1] "1.1.1.1 - The free app that makes your Internet safer", <<https://1.1.1.1/>>.

- [DNS8] "Google Public DNS",
<<https://developers.google.com/speed/public-dns/>>.
- [DNS9] "Quad9 - Internet Security and Privacy In a Few Easy
Steps", <<https://quad9.net/>>.

Appendix A. Example BIND 9 Configuration

A BIND 9 recursive resolver can be configured to act as authoritative for the necessary DNS64 names as described below.

In /etc/named.conf the following line is added:

```
zone "ipv4only.arpa"           { type master; file "ipv4only"; };
```

The file /var/named/ipv4only is created with the following content:

```
$TTL 86400                      ; Default TTL 24 hours
@ IN SOA nameserver.example. admin.nameserver.example. (
    2016052400                  ; Serial
    7200                        ; Refresh ( 7200 = 2 hours)
    3600                        ; Retry   ( 3600 = 1 hour)
    15724800                    ; Expire  (15724800 = 6 months)
    60                          ; Minimum
)
@ IN NS  nameserver.example.

@ IN A    192.0.0.170
@ IN A    192.0.0.171
@ IN AAAA 64:ff9b::192.0.0.170 ; If not using Well-Known Prefix
@ IN AAAA 64:ff9b::192.0.0.171 ; place chosen NAT64 prefix here
```

Authors' Addresses

Stuart Cheshire
Apple Inc.
One Apple Park Way
Cupertino, California 95014
USA

Phone: +1 (408) 996-1010
Email: cheshire@apple.com

David Schinazi
Google LLC
1600 Amphitheatre Parkway
Mountain View, California 94043
USA

Email: dschinazi.ietf@gmail.com

dnsop
Internet-Draft
Intended status: Standards Track
Expires: May 4, 2017

J. Dickinson
J. Hague
S. Dickinson
Sinodun IT
T. Manderson
J. Bond
ICANN
October 31, 2016

C-DNS: A DNS Packet Capture Format
draft-dickinson-dnsop-dns-capture-format-00

Abstract

This document describes a data representation for collections of DNS messages. The format is designed for efficient storage of large packet captures of DNS traffic; it attempts to minimize the size of such packet capture files but retain the full DNS message contents along with the most useful transport meta data. It is intended to assist with the development of DNS traffic monitoring applications and provide a more efficient data exchange format than alternatives such as PCAP files.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on May 4, 2017.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents

(<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
2. Requirements Terminology	4
3. Data Collection Use Cases	4
4. Design Considerations	6
5. C-DNS conceptual overview	7
6. Choice of CBOR	8
7. C-DNS CBOR format	8
7.1. CDDL definition	8
7.2. Format overview	8
7.3. File header contents	9
7.4. File preamble contents	9
7.5. Configuration contents	10
7.6. Block contents	11
7.7. Block preamble map	12
7.8. Block table map	12
7.9. IP address table	13
7.10. Class/Type table	13
7.11. Name/RDATA table	14
7.12. Query Signature table	14
7.13. Question table	16
7.14. Resource Record (RR) table	16
7.15. Question list table	17
7.16. Resource Record list table	17
7.17. Query/Response data	17
7.18. Address Event counts	20
8. C-DNS to PCAP	21
8.1. Name Compression	22
9. Data Collection	23
9.1. Matching algorithm	23
9.2. Message identifiers	24
9.2.1. Primary ID (required)	24
9.2.2. Secondary ID (optional)	24
9.3. Algorithm Parameters	24
9.4. Algorithm Requirements	24
9.5. Algorithm Limitations	25
9.6. Workspace	25
9.7. Output	25
9.8. Post Processing	25

10. IANA Considerations	25
11. Security Considerations	25
12. Acknowledgements	25
13. References	26
13.1. Normative References	26
13.2. Informative References	26
13.3. URIs	27
Appendix A. CDDL	27
Appendix B. DNS Name compression example	33
B.1. NSD compression algorithm	34
B.2. Knot Authoritative compression algorithm	34
B.3. Observed differences	35
Appendix C. Comparison of Binary Formats	35
Appendix D. Sample data on the C-DNS format	35
D.1. Comparison to full PCAPS	35
D.2. Block size choices	35
D.3. Blocking vs more simple output	36
Authors' Addresses	37

1. Introduction

There has long been a need to collect DNS queries and responses on authoritative and recursive name servers for monitoring and analysis. This data is used in a number of ways including traffic monitoring, analyzing network attacks and DITL [ditl].

A wide variety of tools already exist to facilitate the collection of DNS traffic data. DSC [dsc], packetq [packetq], dnscap [dnscap] and dnstap [dnstap]. However, there is no standard exchange format for large DNS packet captures and PCAP [pcap] or PCAP-NG [pcapng] are typically used in practice.

There has also been work on using other text based formats to describe DNS packets [I-D.daley-dnsxml], [I-D.hoffman-dns-in-json] but these are largely aimed at producing convenient representations of single messages.

Many DNS operators may receive 100's of thousands of queries per second on a single name server instance so a mechanism to minimize the storage size (and therefore upload overhead) of the data collected is highly desirable.

This documents focusses on the problem of capturing and storing large packet capture files of DNS traffic.

This document contains

- o A discussion of the some common use cases in which such DNS data is collected. See Section 3.
- o A discussion of the major design considerations in developing an efficient data representation for collections of DNS messages. See Section 4.
- o A definition of a CBOR [RFC7049] representation of a collection of DNS messages. This will be referred to as the C-DNS format (Compacted-DNS). See Section 7.
- o Notes on converting C-DNS back to PCAP format. See Section 8.
- o Some high level implementation considerations for applications designed to produce C-DNS, e.g. a query response matching algorithm. See Section 9.

2. Requirements Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

3. Data Collection Use Cases

In an ideal world it would be optimal to collect full packet captures of all packets going in or out of a name server. However, there are several design choices or other limitations that are common to many DNS installations and operators.

- o Servers are hosted in a variety of situations
 - * Operator self hosted servers
 - * Third party hosting (including multiple third parties)
 - * Third party hardware (including multiple third parties)
- o Data is collected under different conditions
 - * On well provisioned servers running in a steady state.
 - * On heavily loaded servers
 - * On virtualized servers
 - * On servers that are under attack

- * On servers that are unwitting intermediaries in attacks
- o Traffic can be collected via a variety of mechanisms
 - * On the same hardware as the name server itself
 - * Using a network tap to listen in from another server
 - * Using port mirroring to listen in from another server
- o The capabilities of data collection (and upload) networks vary
 - * Out-of-band networks with the same capacity as the in-band network
 - * Out-of-band networks with less capacity than the in-band network
 - * Everything on the in-band network

Clearly, there is a wide range of use cases from very limited data collection environments (third party hardware, servers that are under attack, packet capture on the name server itself and no out-of-band network) to 'limitless' environments (self hosted, well provisioned servers, using a network tap or port mirroring with an out-of-band networks with the same capacity as the in-band network). In the former, it is unfeasible to reliably collect full PCAPS especially if the server is under attack. In the latter case, collection of full PCAPS may be reasonable.

As a result of these restrictions the data format discussed below was designed with the most limited use case in mind such that

- o Data collection will occur on the same hardware as the name server itself.
- o Collected data will be stored on the same hardware as the name server itself, at least temporarily.
- o Collected data being returned to some central analysis system will use the same network interface as the DNS queries and responses.
- o There are multiple third party servers involved.

and therefore minimal storage size of the capture files is a major factor.

Another consideration for any application that records DNS traffic is that the running of the name server software and the transmission of DNS queries and responses is the most important job of a name server. Any data collection system co-located with the name server will need to be intelligent enough to carefully manage its CPU, disk, memory and network utilization. Hence this use case benefits from a format that has a relatively low overhead to produce and minimizes the requirement for further potentially costly compression.

However, it was also essential that interoperability with less restricted infrastructure was maintained. In particular it is highly desirable that the resulting collection format should facilitate the re-creation of common formats (such as PCAPs) that are as close to the original as is realistic given the restrictions above.

4. Design Considerations

This section presents some of the major design considerations used in the development of the C-DNS format.

- o The basic unit of data is a combined DNS Query and the associated Response (a 'Q/R data item'). The same structure will be used for unmatched queries and responses. Queries without responses will be captured omitting the Response data. Responses without queries will be captured omitting the Query data (but using the Query section from the Response, if present, as an identifying QNAME).

Rationale: A Query and Response represents the basic level of a clients interaction with the server. Also, combining the Query and Response into one item lowers storage requirements due to commonality in the data in most cases.

- o Each Q/R data item will comprise a default Q/R data description and a set of optional sections. Inclusion of optional sections shall be configurable.

Rationale: Different users will have different requirements for data to be available for analysis. Users with minimal requirements should not have to pay the cost of recording full data, however this will limit the ability to reconstruct PCAPS. For example omitting the Resource Records from a Response will reduce the files size, and in principle responses can be synthesized if there is enough context.

- o Multiple Q/R items will be collected into blocks in the format. Common data in a block will be abstracted and referenced from individual Q/R items by indexing. The maximum number of Q/R items in a block will be configurable.

Rationale: This blocking and indexing provides a significant reduction in the volume of file data generated. Whilst introducing complexity it provides compression of the data that makes use of knowledge of the DNS packet structure.

[TODO: Further discussion on commonality between DNS packets e.g.

- o common query signatures
- o for the authoritative case there are a finite set of valid responses and much commonality in NXDOMAIN responses]

It is anticipated that the files produced will be subject to further compression using general purpose compression tools. Measurements show that blocking significantly reduces the CPU required to perform such strong compression. See Appendix D.

- o Meta-data about other packets received should also be included in each block. For example counts of malformed DNS packets and non-DNS packets (e.g. ICMP, TCP resets) sent to the server are of interest.

It should be noted that any structured capture format that does not capture the DNS payload byte for byte will likely be limited to some extent in that it cannot represent 'malformed' DNS packets. Only those packets that can be transformed reasonably into the structured format can be represented by it. So if a query is malformed this will lead to the (well formed) DNS responses with error code FORMERR appearing as 'unmatched'.

[TODO: Need further discussion of well-formed vs malformed packets and how name servers view this definition.]

Packets such as those described above can be separately recorded in a PCAP file for later analysis.

5. C-DNS conceptual overview

The following figures show purely schematic representations of the C-DNS format to convey the high-level structure of the C-DNS format. Section 7 provides a detailed discussion of the CBOR representation and individual elements.

Figure showing the C-DNS format (PNG) [1]

Figure showing the C-DNS format (SVG) [2]

Figure showing the Q/R data item and Block tables format (PNG) [3]

Figure showing the Q/R data item and Block tables format (SVG) [4]

6. Choice of CBOR

This document presents a detailed format description using CBOR, the Concise Binary Object Representation defined in [RFC7049].

The choice of CBOR was made taking a number of factors into account.

- o CBOR is a binary representation, and so economical in storage space.
- o Other similar representations were investigated, and whilst all had attractive features, none had a significant advantage over CBOR. See Appendix C and Appendix D - for some discussion of this.
- o CBOR is an IETF Standard and familiar to IETF participants, and being based on the successful JSON text format, requires very little familiarization for those in the wider industry.
- o CBOR can also be easily converted to JSON for debugging and other human inspection requirements.
- o CBOR data schemas can be described using CDDL [I-D.greevenbosch-appsawg-cbor-cddl].

7. C-DNS CBOR format

7.1. CDDL definition

The CDDL definition for the C-DNS format is given in Appendix A.

7.2. Format overview

A C-DNS file begins with a file header containing a file type identifier and preamble. The preamble contains information on the collection settings.

This is followed by a series of data blocks.

A block consists of a block header, containing various tables of common data, and some statistics for the traffic received over the block. The block header is then followed by a list of the Q/R pairs detailing the queries and responses received during the block. The list of Q/R pairs is in turn followed by a list of per-client counts of particular IP events that occurred during collection of the block data.

The exact nature of the DNS data will affect what block size is the best fit, however sample data for a root server indicated that block sizes in the low 1000's give good results. See Appendix D.2 for more details.

If no field type is specified then the field is unsigned.

In the following

- o For all quantities that contain bit flags, bit 0 indicates the least significant bit.
- o Items described as indexes are the index of the data item in the referenced table. Indexes are 1-based. An index value of 0 is reserved to mean not present.

7.3. File header contents

The file header contains the following:

Field	Type	Description
File type ID	Text string	String identifying the file type
File preamble	Map of items	Collection information for the whole file.
File Blocks	Array of Blocks	The data blocks

7.4. File preamble contents

The file preamble contains the following:

Field	Type	Description
Format version	Unsigned	Indicates version of format used in file.
Configuration	Map of items	The collection configuration. Optional.
Generator ID	Text string	String identifying the collection program. Optional.
Host ID	Text string	String identifying the collecting host. Blank if converting an existing PCAP file. Optional.

7.5. Configuration contents

The collection configuration contains the following items. All are optional.

Field	Type	Description
Query timeout	Unsigned	To be matched with a query, a response must arrive within this number of seconds.
Skew timeout	Unsigned	The network stack may report a response before the corresponding query. A response is not considered to be missing a query until after this many micro-seconds.
Snap length	Unsigned	Collect up to this many bytes per packet.
Promiscuous mode	Unsigned	1 if promiscuous mode was enabled on the interface, 0 otherwise.
Interfaces	Array of text strings	Identifiers of the interfaces used for collection.
VLAN IDs	Array of unsigned	Identifiers of VLANs selected for collection.

Filter	Text string	"tcpdump" [pcap] style filter for input.
Query collection options	Unsigned	Bit flags indicating sections in Query packets to be collected. Bit 0. Collect second and subsequent question sections. Bit 1. Collect Answer sections. Bit 2. Collect Authority sections. Bit 3. Collection Additional sections.
Response collection options	Unsigned	Bit flags indicating sections in Response packets to be collected. Bit 0. Collect second and subsequent question sections. Bit 1. Collect Answer sections. Bit 2. Collect Authority sections. Bit 3. Collection Additional sections.
Accept RR types	Array of text strings	A set of RR type names [rrtypes]. If not empty, only the nominated RR types are collected.
Ignore RR types	Array of text strings	A set of RR type names [rrtypes]. If not empty, all RR types are collected except those listed. If present, this item must be empty if a non-empty list of Accept RR types is present.

7.6. Block contents

Each block contains the following:

Field	Type	Description
Block preamble	Map of items	Overall information for the block.
Block statistics	Map of statistics	Statistics about the block.
Block tables	Map of tables	The tables containing data referenced by individual Q/R entries.
Q/Rs	Array of Q/Rs	Details of individual Q/R pairs.
Address Event Counts	Array of Address Event counts	Per client counts of ICMP messages and TCP resets.

7.7. Block preamble map

The block preamble map contains overall information for the block.

Field	Type	Description
Timestamp	Array of unsigned	A timestamp for the earliest record in the block. The timestamp is specified as a CBOR array with two elements as in Posix struct timeval. The first element is an unsigned integer time_t and the second is an unsigned integer number of microseconds. The latter is always a value between 0 and 999,999.

7.8. Block table map

The block table map contains the block tables. Each element, or table, is an array. The following tables detail the contents of each block table.

The Present column in the following tables indicates the circumstances when an optional field will be present. A Q/R pair may be:

- o A Query plus a Response.

- o A Query without a Response.
- o A Response without a Query.

Also:

- o A Query and/or a Response may contain an OPT section.
- o A Question may or may not be present. If the Query is available, the Question section of the Query is used. If no Query is available, the Question section of the Response is used. Unless otherwise noted, a Question refers to the first Question in the Question section.

So, for example, a field listed with a Present value of QUERY is present whenever the Q/R pair contains a Query. If the pair contains a Response only, the field will not be present.

7.9. IP address table

This table holds all client and server IP addresses in the block. Each item in the table is a single IP address.

Field	Type	Description
Address	Byte string	The IP address, in network byte order. The string is 4 bytes long for an IPv4 address, 16 bytes long for an IPv6 address.

7.10. Class/Type table

This table holds pairs of RR CLASS and TYPE values. Each item in the table is a CBOR map.

Field	Description
Class	CLASS value.
Type	TYPE value.

7.11. Name/RDATA table

This table holds the contents of all NAME or RDATA items in the block. Each item in the table is the content of a single NAME or RDATA.

Field	Type	Description
Data	Byte string	The NAME or RDATA contents. NAMEs, and labels within RDATA contents, are in uncompressed label format.

7.12. Query Signature table

This table holds elements of the Q/R data that are often common to between different individual Q/R records. Each item in the table is a CBOR map. Each item in the map has an unsigned value and an unsigned key.

The following abbreviations are used in the Present (P) column

- o Q = QUERY
- o A = Always
- o QT = QUESTION
- o QO = QUERY, OPT
- o QR = QUERY & RESPONSE
- o R = RESPONSE

Field	P	Description
Server address	A	The index in the IP address table of the server IP address.
Server port	A	The server port.
Transport flags	A	Bit flags describing the protocol used to service the query. Bit 0 is the least significant bit. Bit 0. Transport type. 0 = UDP, 1 = TCP.

			Bit 1. IP type. 0 = IPv4, 1 = IPv6.
Q/R signature flags	A		<p>Bit flags indicating information present in this Q/R pair. Bit 0 is the least significant bit.</p> <p>Bit 0. 1 if a Query is present.</p> <p>Bit 1. 1 if a Response is present.</p> <p>Bit 2. 1 if one or more Question is present.</p> <p>Bit 3. 1 if a Query is present and it has an OPT Resource Record.</p> <p>Bit 4. 1 if a Response is present and it has an OPT Resource Record.</p> <p>Bit 5. 1 if a Response is present but has no Question.</p>
Query OPCODE	Q		Query OPCODE.
Q/R DNS flags	A		<p>Bit flags with values from the Query and Response DNS flags. Bit 0 is the least significant bit. Flag values are 0 if the Query or Response is not present.</p> <p>Bit 0. Query Checking Disabled (CD) flag.</p> <p>Bit 1. Query Authenticated Data (AD) flag.</p> <p>Bit 2. Query reserved (Z) flag.</p> <p>Bit 3. Query Recursion Available (RA) flag.</p> <p>Bit 4. Query Recursion Desired (RD) flag.</p> <p>Bit 5. Query TrunCation (TC) flag.</p> <p>Bit 6. Query Authoritative Answer (AA) flag.</p> <p>Bit 7. Query DNSSEC answer OK (DO) flag.</p> <p>Bit 8. Response Checking Disabled (CD) flag.</p> <p>Bit 9. Response Authenticated Data (AD) flag.</p> <p>Bit 10. Response reserved (Z) flag.</p> <p>Bit 11. Response Recursion Available (RA) flag.</p> <p>Bit 12. Response Recursion Desired (RD) flag.</p> <p>Bit 13. Response TrunCation (TC) flag.</p> <p>Bit 14. Response Authoritative Answer (AA) flag.</p>
Query RCODE	Q		Query RCODE. If the Query contains OPT, this value incorporates any EXTENDED_RCODE_VALUE.
Question Class/Type	QT		The index in the Class/Type table of the CLASS and TYPE of the first Question.
Question QDCOUNT	QT		The QDCOUNT in the Query, or Response if no Query present.

Query ANCOUNT	Q	Query ANCOUNT.
Query ARCOUNT	Q	Query ARCOUNT.
Query NSCOUNT	Q	Query NSCOUNT.
Query EDNS version	QO	The Query EDNS version.
EDNS UDP size	QO	The Query EDNS sender's UDO payload size
Query OPT RDATA	QO	The index in the NAME/RDATA table of the OPT RDATA.
Response RCODE	R	Response RCODE. If the Response contains OPT, this value incorporates any EXTENDED_RCODE_VALUE.

7.13. Question table

This table holds details on individual Questions in a Question section. Each item in the table is a CBOR map containing a single Question. Each item in the map has an unsigned value and an unsigned key. This data is optionally collected.

Field	Description
QNAME	The index in the NAME/RDATA table of the QNAME.
Class/Type	The index in the Class/Type table of the CLASS and TYPE of the Question.

7.14. Resource Record (RR) table

This table holds details on individual Resource Records in RR sections. Each item in the table is a CBOR map containing a single Resource Record. This data is optionally collected.

Field	Description
NAME	The index in the NAME/RDATA table of the NAME.
Class/Type	The index in the Class/Type table of the CLASS and TYPE of the RR.
TTL	The RR Time to Live.
RDATA	The index in the NAME/RDATA table of the RR RDATA.

7.15. Question list table

This table holds a list of second and subsequent individual Questions in a Question section. Each item in the table is a CBOR unsigned. This data is optionally collected.

Field	Description
Question	The index in the Question table of the individual Question.

7.16. Resource Record list table

This table holds a list of individual Resource Records in a Answer, Authority or Additional section. Each item in the table is a CBOR unsigned. This data is optionally collected.

Field	Description
RR	The index in the Resource Record table of the individual Resource Record.

7.17. Query/Response data

The block Q/R data is a CBOR array of individual Q/R items. Each item in the array is a CBOR map containing details on the individual Q/R pair.

Note that there is no requirement that the elements of the Q/R array are presented in strict chronological order.

The following abbreviations are used in the Present (P) column

- o Q = QUERY
- o A = Always
- o QT = QUESTION
- o QO = QUERY, OPT
- o QR = QUERY & RESPONSE
- o R = RESPONSE

Each item in the map has an unsigned value (with the exception of those listed below) and an unsigned key.

- o Query extended information and Response extended information which are of Type Extended Information.
- o Response delay which is an integer (This can be negative if the network stack/capture library returns them out of order.)

Field	P	Description
Time offset	A	Q/R timestamp as an offset in microseconds from the Block pre-amble Timestamp. The timestamp is the timestamp of the Query, or the Response if there is no Query.
Client address	A	The index in the IP address table of the client IP address.
Client port	A	The client port.
Transaction ID	A	DNS transaction identifier.
Query signature	A	The index of the more information on the Q/R in the Query Signature table.
Client hoplimit	Q	The IPv4 TTL or IPv6 Hoplimit from the Query packet.
Response delay	QR	The time different between Query and Response, in microseconds.
Question NAME	QT	The index in the NAME/RDATA table of the QNAME for the first Question.
Response size	R	The size of the DNS message (not the packet containing the message, just the DNS message) that forms the Response.
Query extended information	Q	Extended Query information. This item is only present if collection of extra Query information is configured.
Response extended information	R	Extended Response information. This item is only present if collection of extra Query information is configured.

The collector always collects basic Q/R information. It may be configured to collect details on Question, Answer, Authority and Additional sections of the Query, the Response or both. Note that only the second and subsequent Questions of any Question section are collected (the details of the first are in the basic information), and that OPT Records are not collected in the Additional section.

The Extended information is a CBOR map as follows. Each item in the map is present only if collection of the relevant details is configured. Each item in the map has an unsigned value and an unsigned key.

Field	Description
Question	The index in the Questions list table of the entry listing the second and subsequent Question sections for the Query or Response.
Answer	The index in the RR list table of the entry listing the Answer Resource Record sections for the Query or Response.
Authority	The index in the RR list table of the entry listing the Authority Resource Record sections for the Query or Response.
Additional	The index in the RR list table of the entry listing the Additional Resource Record sections for the Query or Response.

7.18. Address Event counts

This table holds counts of various IP related events relating to traffic with individual client addresses.

Field	Type	Description
Event type	Unsigned	The type of event. The following events types are currently defined: 0. TCP reset. 1. ICMP time exceeded. 2. ICMP destination unreachable. 3. ICMPv6 time exceeded. 4. ICMPv6 destination unreachable. 5. ICMPv6 packet too big.
Event code	Unsigned	A code relating to the event. Optional.
Address index	Unsigned	The index in the IP address table of the client address.
Count	Unsigned	The number of occurrences of this event during the block collection period.

8. C-DNS to PCAP

It is possible to re-construct PCAP files from the C-DNS format. However this is a lossy process and some of the issues with reconstructing both the DNS payload and the full packet stream are outlined here.

Firstly the reconstruction depends on whether or not all the optional sections of both the query and response were captured in the C-DNS file. Clearly if they were not all captured the reconstruction is imperfect.

Secondly, even if all sections of the response were captured name compression presents a challenge in reconstructing the DNS response payload byte for byte. Section 8.1 discusses this in more detail.

Thirdly, not all transport information is captured in the C-DNS format. For example, the following aspects of the original packet stream cannot be re-constructed from the C-DNS format:

- o IP Fragmentation
- o TCP stream information:
 - * Multiple DNS messages may have been sent in a single TCP segment

- * A DNS payload may have be split across multiple TCP segments
- * Multiple DNS messages may have be sent on a single TCP session
- o Malformed DNS messages and non-DNS packets

Simple assumptions can be made on the reconstruction - fragmented and DNS-over-TCP messages can be reconstructed into 'single' packets and a single TCP session can be constructed for each TCP packet.

Additionally if the malformed and non-DNS packets are captured separately into PCAPs they can be merged with PCAPs reconstructed from C-DNS to produce a more complete packet stream.

8.1. Name Compression

All the names stored in the C-DNS format are full domain names; no DNS style name compression is used on the individual names within the format. Therefore when reconstructing a packet name compression must be used in order to re-produce the on the wire representation of the packet.

[RFC1035] name compression works by substituting trailing sections of a name with a reference back to the occurrence of those sections earlier in the packet. Not all name server software uses the same algorithm when compressing domain names within the responses. Some attempt maximum recompression at the expense of runtime resources, others use heuristics to balance compression and speed and others use different rules for what is a valid compression target.

This means that responses to the same question from different name server software which match in terms of DNS payload content (header, counts, RRs with name compression removed) do not necessarily match byte for byte on the wire.

From the C-DNS format it is not possible to ensure that the DNS response payload is reconstructed byte for byte. However it can at least, in principle, be reconstructed to have the correct payload length (since the original response length is captured) if there is enough knowledge of the commonly implemented name compression algorithms. For example, a simplistic approach would be to try each algorithm in turn to see if it reproduces the original length, stopping at the first match. This would not guarantee the correct algorithm has been used as it is possible to match the length whilst still not matching the on the wire bytes but without further information added to the C-BOR this is the best that can be achieved.

Appendix B presents an example of two differing compression algorithms used by well known name server software.

9. Data Collection

This section describes a non-normative proposed algorithm for the processing of a captured stream of DNS queries and responses and matching queries/responses where possible.

For the purposes of this discussion, it is assumed that the input has been pre-processed such that:

1. All IP fragmentation reassembly, TCP stream reassembly etc. has already been performed
2. Each message is associated with transport meta-data required to generate the Primary ID (see below)
3. Each message has a well-formed DNS header of 12 bytes and (if present) the first RR in the query section can be parsed to generate the Secondary ID (see below).
 - * As noted earlier, this requirement can result in a malformed query being removed in the pre-processing stage, but the correctly formed response with RCODE of FORMERR being present

DNS messages are processed in the order they are delivered to the application.

- o It should be noted that packet capture libraries do not necessary provide packets in strict chronological order.

[TODO: Discuss the corner cases resulting from this in more detail.]

9.1. Matching algorithm

A schematic representation of the algorithm for matching Q/R pairs is shown in the following diagram:

Figure showing the packet matching algorithm format (PNG) [5]

Figure showing the packet matching algorithm format (SVG) [6]

and further details of the algorithm are given in the following sections.

9.2. Message identifiers

9.2.1. Primary ID (required)

A Primary ID can be constructed for each message which is composed of the following data:

1. Source IP Address
2. Destination IP Address
3. Source Port
4. Destination Port
5. Transport
6. DNS Message ID

9.2.2. Secondary ID (optional)

If present, the first question in the Question section is used as a secondary ID for each message. Note that there may be well formed DNS queries that have a QDCOUNT of 0, and some responses may have a QDCOUNT of 0 (for example, RCODE=FORMERR or NOTIMP)

9.3. Algorithm Parameters

1. Configurable timeout

9.4. Algorithm Requirements

The algorithm is designed to handle the following input data:

1. Multiple queries with the same Primary ID (but different Secondary ID) arriving before any responses for these queries are seen.
2. Multiple queries with the same Primary and Secondary ID arriving before any responses for these queries are seen.
3. Queries for which no later response can be found within the specified timeout.
4. Responses for which no previous query can be found within the specified timeout.

9.5. Algorithm Limitations

For cases 1 and 2 listed in the above requirements, it is not possible to unambiguously match queries with responses. The solution to this employed in this algorithm is to match to the earliest query with the correct Primary and Secondary ID.

9.6. Workspace

A FIFO structure is used to hold the Q/R items during processing.

9.7. Output

The output is a list of Q/R data items. Both the Query and Response elements are optional in these items, therefore Q/R items have one of three types of content:

1. Paired Q/R messages
2. A query message (no response)
3. A response message (no query)

The timestamp of a list item is that of the query for cases 1 and 2 and that of the response for case 3.

9.8. Post Processing

When ending capture, all remaining entries in the Q/R FIFO should be treated as timed out queries.

10. IANA Considerations

None

11. Security Considerations

Any control interface MUST perform authentication and encryption.

Any data upload MUST be authenticated and encrypted.

12. Acknowledgements

The authors wish to thank CZ.NIC, in particular Tomas Gavenciak, for many useful discussions on binary formats, compression and packet matching. Also Jan Vcelak and Wouter Wijngaards for discussions on name compression.

Also, Miek Gieben for mmark [7]

13. References

13.1. Normative References

- [RFC1035] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, DOI 10.17487/RFC1035, November 1987, <<http://www.rfc-editor.org/info/rfc1035>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <<http://www.rfc-editor.org/info/rfc7049>>.

13.2. Informative References

- [ditl] DNS-OARC, "DITL", 2016, <<https://www.dns-oarc.net/oarc/data/ditl>>.
- [dnscap] DNS-OARC, "DNSCAP", 2016, <<https://www.dns-oarc.net/tools/dnscap>>.
- [dnstap] dnstap.io, "dnstap", 2016, <<http://dnstap.info/>>.
- [dsc] Wessels, D. and J. Lundstrom, "DSC", 2016, <<https://www.dns-oarc.net/tools/dsc>>.
- [I-D.daley-dnsxml] Daley, J., Morris, S., and J. Dickinson, "dnsxml - A standard XML representation of DNS data", draft-daley-dnsxml-00 (work in progress), July 2013.
- [I-D.greevenbosch-appsawg-cbor-cddl] Vigano, C. and H. Birkholz, "CBOR data definition language (CDDL): a notational convention to express CBOR data structures", draft-greevenbosch-appsawg-cbor-cddl-09 (work in progress), September 2016.
- [I-D.hoffman-dns-in-json] Hoffman, P., "Representing DNS Messages in JSON", draft-hoffman-dns-in-json-09 (work in progress), October 2016.

- [packetq] .SE - The Internet Infrastructure Foundation, "PacketQ", 2014, <<https://github.com/dotse/PacketQ>>.
- [pcap] tcpdump.org, "PCAP", 2016, <<http://www.tcpdump.org/>>.
- [pcapng] Tuexen, M., Risso, F., Bongertz, J., Combs, G., and G. Harris, "pcap-ng", 2016, <<https://github.com/pcapng/pcapng>>.
- [rrtypes] IANA, "RR types", 2016, <<http://www.iana.org/assignments/dns-parameters/dns-parameters.xhtml#dns-parameters-4>>.

13.3. URIs

- [1] https://github.com/dns-stats/draft-dns-capture-format/blob/master/cdns_format.png
- [2] https://github.com/dns-stats/draft-dns-capture-format/blob/master/cdns_format.svg
- [3] https://github.com/dns-stats/draft-dns-capture-format/blob/master/qr_data_format.png
- [4] https://github.com/dns-stats/draft-dns-capture-format/blob/master/qr_data_format.svg
- [5] https://github.com/dns-stats/draft-dns-capture-format/blob/master/packet_matching.png
- [6] https://github.com/dns-stats/draft-dns-capture-format/blob/master/packet_matching.svg
- [7] <https://github.com/miekg/mmark>
- [8] <https://www.nlnetlabs.nl/projects/nsd/>
- [9] <https://www.knot-dns.cz/>

Appendix A. CDDL

; CDDL specification of the file format for C-DNS,
; which describes a collection of DNS Query/Response pairs.

```
File = [  
    file-type-id   : tstr,           ; "DNS-STAT"  
    file-preamble  : FilePreamble,  
    file-blocks    : [* Block],  
]
```

```
FilePreamble = {  
    format-version => uint,  
    ? configuration => Configuration,  
    ? generator-id  => tstr,  
    ? host-id       => tstr,  
}
```

```
format-version = 0  
configuration   = 1  
generator-id    = 2  
host-id        = 3
```

```
Configuration = {  
    ? query-timeout      => uint,  
    ? skew-timeout      => uint,  
    ? snaplen           => uint,  
    ? promisc           => uint,  
    ? interfaces        => [* tstr],  
    ? vlan-ids          => [* uint],  
    ? filter            => tstr,  
    ? query-options     => uint,      ; See below  
    ? response-options  => uint,  
    ? accept-rr-types   => [* tstr],  
    ? ignore-rr-types   => [* tstr],  
}
```

```
; query-options and response-options are bitmasks. A bit set adds in the  
; specified sections.  
;
```

```
; second & subsequent question sections = 1  
; answer sections = 2  
; authority sections = 4  
; additional sections = 8
```

```
query-timeout      = 0  
skew-timeout       = 1  
snaplen            = 2  
promisc            = 3  
interfaces         = 4  
vlan-ids           = 5  
filter             = 6  
query-options      = 7  
response-options   = 8  
accept-rr-types    = 9;  
ignore-rr-types    = 10;
```

```
Block = {
```



```
preamble          => BlockPreamble,
? statistics       => BlockStatistics, ; Much of this could be derived
tables            => BlockTables,
queries           => [* QueryResponse],
address-event-counts => [* AddressEventCount],
}

preamble          = 0
statistics        = 1
tables            = 2
queries           = 3
address-event-counts = 4

BlockPreamble = {
    start-time => Timeval
}

start-time = 1

Timeval = [
    seconds      : uint,
    microseconds : uint,
]

BlockStatistics = {
    ? total-packets      => uint,
    ? total-pairs        => uint,
    ? unmatched_queries  => uint,
    ? unmatched_responses => uint,
    ? malformed-packets  => uint,
    ? non-dns-packets    => uint,
    ? out-of-order-packets => uint,
    ? missing-pairs      => uint,
    ? missing-packets    => uint,
    ? missing-non-dns    => uint,
}

total-packets      = 0
total-pairs        = 1
unmatched_queries  = 2
unmatched_responses = 3
malformed-packets  = 4
non-dns-packets    = 5
out-of-order-packets = 6
missing-pairs      = 7
missing-packets    = 8
missing-non-dns    = 9
```

```
BlockTables = {
    ip-address => [* bstr],
    classtype  => [* ClassType],
    name-rdata => [* bstr],           ; Holds both Name RDATA and RDATA
    query_sig  => [* QuerySignature]
    ? qlist    => [* QuestionList],
    ? qrr      => [* Question],
    ? rrlist   => [* RRList],
    ? rr       => [* RR],
}

ip-address = 0
classtype  = 1
name-rdata = 2
query_sig  = 3
qlist      = 4
qrr        = 5
rrlist     = 6
rr         = 7

QueryResponse = {
    time-useconds      => int,           ; Time offset from start of block
    client-address-index => uint,
    client-port        => uint,
    transaction-id     => uint,
    query-signature-index => uint,
    ? client-hoplimit  => uint,
    ? delay-useconds   => int,           ; Times may be -ve at capture
    ? query-name-index => uint,
    ? response-size    => uint,         ; DNS size of response
    ? query-extended   => QueryResponseExtended,
    ? response-extended => QueryResponseExtended,
}

time-useconds      = 0
client-address-index = 1
client-port        = 2
transaction-id     = 3
query-signature-index = 4
client-hoplimit    = 5
delay-useconds     = 6
query-name-index   = 7
response-size      = 8
query-extended     = 9
response-extended  = 10

ClassType = {
    type => uint,
```

```

    class => uint,
}

type = 0
class = 1

QuerySignature = {
    server-address-index    => uint,
    server-port             => uint,
    transport-flags        => uint,
    qr-sig-flags            => uint,
    ? query-opcode          => uint,
    qr-dns-flags            => uint,
    ? query-rcode           => uint,
    ? query-classtype-index => uint,
    ? query-qd-count        => uint,
    ? query-an-count        => uint,
    ? query-ar-count        => uint,
    ? query-ns-count        => uint,
    ? edns-version          => uint,
    ? udp-buf-size          => uint,
    ? opt-rdata-index       => uint,
    ? response-rcode        => uint,
}

server-address-index = 0
server-port          = 1
transport-flags      = 2
qr-sig-flags         = 3
query-opcode         = 4
qr-dns-flags         = 5
query-rcode          = 6
query-classtype-index = 7
query-qd-count       = 8
query-an-count       = 9
query-ar-count       = 10
query-ns-count       = 11
edns-version         = 12
udp-buf-size         = 13
opt-rdata-index      = 14
response-rcode       = 15

QuestionList = [
    * uint,                                ; Index of Question
]

Question = {
    name-index    => uint,                ; Second and subsequent questions
                                           ; Index to a name in the name-rdata table

```

```
    classtype-index => uint,
}

name-index      = 0
classtype-index = 1

RRList = [
    * uint,                ; Index of RR
]

RR = {
    name-index      => uint,                ; Index to a name in the name-rdata table
    classtype-index => uint,
    ttl             => uint,
    rdata-index     => uint,                ; Index to RDATA in the name-rdata table
}

ttl             = 2
rdata-index     = 3

QueryResponseExtended = {
    ? question-index => uint,                ; Index of QuestionList
    ? answer-index  => uint,                ; Index of RRList
    ? authority-index => uint,
    ? additional-index => uint,
}

question-index  = 0
answer-index    = 1
authority-index = 2
additional-index = 3

AddressEventCount = {
    ae-type       => &AddressEventType,
    ? ae-code     => uint,
    ae-address-index => uint,
    ae-count      => uint,
}

ae-type         = 0
ae-code         = 1
ae-address-index = 2
ae-count        = 3

AddressEventType = (
    tcp-reset: 0,
    icmp-time-exceeded : 1,
    icmp-dest-unreachable : 2,
```

```

icmpv6-time-exceeded      : 3,
icmpv6-dest-unreachable: 4,
icmpv6-packet-too-big    : 5,
)

```

Appendix B. DNS Name compression example

The basic algorithm which follows the guidance in [RFC1035] is simply to collect each name, and the offset in the packet at which it starts, during packet construction. As each name is added, it is offered to each of the collected names in order of collection, starting from the first name. If labels at the end of the name can be replaced with a reference back to part (or all) of the earlier name, and if the uncompressed part of the name is shorter than any compression already found, the earlier name is noted as the compression target for the name.

The following tables illustrate the process. In an example packet, the first name is example.com.

N	Name	Uncompressed	Compression Target
1	example.com		

The next name added is bar.com. This is matched against example.com. The com part of this can be used as a compression target, with the remaining uncompressed part of the name being bar.

N	Name	Uncompressed	Compression Target
1	example.com		
2	bar.com	bar	1 + offset to com

The third name added is www.bar.com. This is first matched against example.com, and as before this is recorded as a compression target, with the remaining uncompressed part of the name being www.bar. It is then matched against the second name, which again can be a compression target. Because the remaining uncompressed part of the name is www, this is an improved compression, and so it is adopted.

N	Name	Uncompressed	Compression Target
1	example.com		
2	bar.com	bar	1 + offset to com
3	www.bar.com	www	2

As an optimization, if a name is already perfectly compressed - in other words, the uncompressed part of the name is empty - no further names will be considered for compression.

B.1. NSD compression algorithm

Using the above basic algorithm the packet lengths of responses generated by NSD [8] can be matched almost exactly. At the time of writing, a tiny number (<.01%) of the reconstructed packets had incorrect lengths.

B.2. Knot Authoritative compression algorithm

The Knot Authoritative [9] name server uses different compression behavior, which is the result of internal optimization designed to balance runtime speed with compression size gains. In brief, and omitting complications, Knot Authoritative will only consider the QNAME and names in the immediately preceding RR section in an RRSET as compression targets.

A set of smart heuristics as described below can be implemented to mimic this and while not perfect it produces output nearly, but not quite, as good a match as with NSD. The heuristics are:

1. A match is only perfect if the name is completely compressed AND the TYPE of the section in which the name occurs matches the TYPE of the name used as the compression target.
2. If the name occurs in RDATA:
 - a If the compression target name is in a query, then only the first RR in an RRSET can use that name as a compression target.
 - b The compression target name MUST be in RDATA.
 - c The name section TYPE must match the compression target name section TYPE.

- d The compression target name MUST be in the immediately preceding RR in the RRSET.

Using this algorithm less than 0.1% of the reconstructed packets had incorrect lengths.

B.3. Observed differences

In sample traffic collected on a root name server around 2-4% of responses generated by Knot had different packet lengths to those produced by NSD.

Appendix C. Comparison of Binary Formats

Several binary representations were considered in particular CBOR, Apache Avro and Protocol Buffers.

Protocol Buffers and Avro both require a data schema, and validate data being stored against that schema.

[TODO: Finish pros and cons of CBOR vs Avro vs Protocol buffers - tools, schema, adoption, etc.]

The difference in file sizes were mostly minimal See Appendix D.3.

Appendix D. Sample data on the C-DNS format

This section presents some example figures for the output size of capture files when using different block sizes, data representations and binary formats. The data is sample data for a root instance.

[TODO: This section needs more work..]

D.1. Comparison to full PCAPS

As can be seen in more detail below for this sample data the compressed C-DNS files are around 30% the size of the full compressed PCAPS. It should also be noted that experiments showed that compression of the C-DNS format required very roughly an order of magnitude less CPU resources than compression of full PCAPSs when using one core from a 3.5GHz i7 processor.

D.2. Block size choices

[TODO: Discuss trade-off of file block size vs memory consumption.]

[TODO: Add graph that demonstrates block size of 5000 is optimal for the sample data used.]

D.3. Blocking vs more simple output

Some experiments were conducted producing output in a very simple format involving a single record per Q/R data item (akin to a .csv representation). The aim here was to examine whether the blocking mechanism (using a block size of 5000) was worth the complexity, particularly after compression of the output file using several general purpose compression tools. The original PCAP file was 325.79M and compressed using xz to 24.3Mb.

Format	Output size	lz4	gzip	xz
cbor-simple	44.23M	16.06M	11.50M	7.51M
cbor-block	22.44M	15.14M	10.70M	7.23M

It might be expected that blocking is exploiting commonality that a general purpose compression engine could also exploit, and the figures do indeed bear this out. The more powerful (and resource-consuming) the compression, the closer the compressed simple file size gets to the compressed chunk file size. With no compression, the blocked output size is typically half that of the simple output, but as greater degrees of compression are applied the gap shrinks. However, even with the stronger compressor, the chunked output remains roughly 5-10% smaller than the simple output. This, and the higher gains at lower compression, might be significant, depending on the target environment.

[TODO: Add data on reduction in CPU overhead of compressing blocked output vs simple output.]

This was repeated using some other binary representations:

Format	Output size	lz4	gzip	xz
json-simple	189.85M	25.59M	16.03M	9.74M
avro-simple	43.31M	16.07M	11.92M	7.99M
avro-block	17.44M	12.94M	10.08M	7.18M
protobuf-simple	46.02M	15.79M	11.59M	7.94M
protobuf-block	22.08M	15.43M	10.91M	7.40M

There's not a lot to choose between the three contenders with simple output. Avro produces the smaller output, CBOR the next and Protocol Buffers the largest, but the difference is under 10%. However, with blocking, while CBOR and Protocol Buffers are again within a few

percentage points of each other (though Protocol Buffers now has a slight advantage), Avro produces files in the region of 20% smaller, and holds a diminishing advantage through increased compression.

Authors' Addresses

John Dickinson
Sinodun IT
Magdalen Centre
Oxford Science Park
Oxford OX4 4GA

Email: jad@sinodun.com

Jim Hague
Sinodun IT
Magdalen Centre
Oxford Science Park
Oxford OX4 4GA

Email: jim@sinodun.com

Sara Dickinson
Sinodun IT
Magdalen Centre
Oxford Science Park
Oxford OX4 4GA

Email: sara@sinodun.com

Terry Manderson
ICANN

Email: terry.manderson@icann.org

John Bond
ICANN

Email: john.bond@icann.org

Network Working Group
Internet-Draft
Updates: 1034, 2181 (if approved)
Intended status: Standards Track
Expires: May 5, 2017

K. Fujiwara
JPRS
November 01, 2016

Updating Resolver Algorithm
draft-fujiwara-dnsop-resolver-update-00

Abstract

Parent side NS RRSets and glue records are all information to access servers for child zone. However, they may be overwritten by child zone data (zone apex NS RRSets and other A/AAAA RRSets). The overwrite makes name resolution unstable and induces vulnerabilities. RFC 2181 section 5.4.1 specifies trustworthiness of DNS data. And it is deemed that that all cached data (authoritative data, non-authoritative data, referrals and glue records) are merged into one. Resolvers may answer non-authoritative data, referrals and glue records that should not be returned. This document proposes updating resolver algorithm that separates the cache to "authoritative data cache" and "delegation cache". The former is used to answer stub resolvers, and the latter is used to iterate zones.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on May 5, 2017.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
2. Terminology	3
3. Traditional resolver algorithm	3
3.1. Importance of parent side NS RRSets	3
3.2. Recommendation of resolver's answer	4
3.3. Traditional resolver algorithm	5
4. Problem Statement	5
5. Updating of Resolver Algorithm	6
5.1. Recommendations to resolvers	6
5.2. Update to resolver algorithm	6
5.3. Characteristics of the update	7
5.4. Issues of the update	8
6. Implementation Ideas	8
7. IANA Considerations	8
8. Security Considerations	8
9. Acknowledgments	8
10. Change History	8
11. References	8
11.1. Normative References	8
11.2. Informative References	9
Author's Address	9

1. Introduction

Resolver algorithm is defined in [RFC1034] and updated by [RFC2181]. The resolver algorithm seems to assume single cache that holds all RRSets from received responses. [RFC2181] Section 5.4.1 Ranking data specifies the trustworthiness order of RRSets. When a resolver receives higher trustworthy data, the cached data is replaced by the received data.

Parent side NS RRSets is very important because it creates new zone and specifies how to access name servers for the created zone described in Section 3.1. However, parent side NS RRSets and glue records have least trustworthiness. The parent side NS RRSets and

glue records are replaced by authoritative data if resolvers receive authoritative data described in Section 3.3.

The overwrite makes name resolution unstable and some vulnerabilities described in Section 4. And it may break requirements of resolvers' answers described in Section 3.2.

This document proposes updated resolver algorithm that separate authoritative data cache that is answered to stub resolvers and delegation cache that is used to iterate zones. Details are described in Section 5

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

Many of the specialized terms used in this specification are defined in DNS Terminology [RFC7719].

3. Traditional resolver algorithm

[RFC1034] defines "zone cut", "delegation", "referral", "glue records", "authoritative", and "resolver algorithm".

[RFC2181] clarified the resolver algorithm defined in [RFC1034].

3.1. Importance of parent side NS RRSets

[RFC1034], [RFC2181] and [RFC7719] defines zone "cut", "delegation", "referral", and parent side NS RRSets functions as follows.

"'cuts' in the name space can be made between any two adjacent nodes. After all cuts are made, each group of connected name space is a separate zone. The zone is said to be authoritative for all names in the connected region." (Quoted from RFC 1034, Section 4.2)

"The RRs that describe cuts around the bottom of the zone are NS RRs that name the servers for the subzones. Since the cuts are between nodes, these RRs are NOT part of the authoritative data of the zone, and should be exactly the same as the corresponding RRs in the top node of the subzone." (Quoted from RFC 1034, section 4.2.1)

"That is, parent zones have all the information needed to access servers for their children zones." (Quoted from RFC 1034, section 4.2.1)

"Resolvers must be able to access at least one name server and use that name server's information to answer a query directly, or pursue the query using referrals to other name servers." (Quoted from RFC 1034, Section 2.4)

"Delegation is the process by which a separate zone is created in the name space beneath the apex of a given domain. Delegation happens when an NS RRset is added in the parent zone for the child origin." (Quoted from RFC 7719)

"This situation typically occurs when the glue address RRs have a smaller TTL than the NS RRs marking delegation," (Quoted from RFC 1035, Section 7.2)

"The existence of a zone cut is indicated in the parent zone by the existence of NS records specifying the origin of the child zone." (Quoted from RFC 2181, Section 6)

As described above, parent side NS RRSet makes a new zone. Parent side NS RRSet (referral) and glue records are all the information to access servers for the child zone.

That is, resolvers SHOULD NOT use child side NS RRSet to iterate zones.

3.2. Recommendation of resolver's answer

RFC 1034 describes resolver's answer as follows.

"The ideal answer is one from a server authoritative for the query which either gives the required data or a name error. The data is passed back to the user and entered in the cache for future use if its TTL is greater than zero." (Quoted from RFC 1034, Section 5.3.3)

"The simplest mode for the client is recursive, since in this mode the name server acts in the role of a resolver and returns either an error or the answer, but never referrals." (Quoted from RFC 1034, Section 4.3.1)

Recently, most of full-service resolver implementations answer only authoritative data to stub resolvers.

As described above, recommendation of resolver's answer is "answer only authoritative data." It does not break existing standards.

3.3. Traditional resolver algorithm

Resolver algorithm is defined in [RFC1034] Section 5.3.3. Resolvers cache all RRSets during the iterations in their cache. When resolvers receive new data, they will update their cache. The update is explained using an example resolution of "www.example.com/A" in "example.com" zone as follows.

When a resolver iterates "www.example.com/A" query, then one of root servers responds "com" NS RRSets (referral) with glue records, one of "com" servers responds "example.com" NS RRSets with glue records, and one of "example.com" servers responds "www.example.com" A RRSets.

As a result, the resolver caches all RRSets during the iterations in its cache.

After then, a resolver receives "example.com/NS" query, it retrieves "example.com" NS authoritative data defined in zone apex of "example.com" and it overwrites "example.com/NS" in the resolver's cache as "trustworthiness" rule of [RFC2181].

If the parent side "example.com" NS RRSets and the child side "example.com" NS RRSets are different, next resolution result will be changed because the resolver will send "www.example.com/A" query to name servers that are specified by "example.com" NS RRSets defined in zone apex. Glue records in the cache are also overwritten by authoritative data, and then, IP addresses of name servers that the resolver send to will be changed.

The other case, if one of "example.com" name servers responds "www.example.com" A RRSets with "example.com" NS RRSets in authority section (several existing authoritative server implementations perform this) , "example.com" NS RRSets from "com" TLD servers (referral) is overwritten by "example.com" NS data attached in the authoritative answer from child zone.

The overwrite is specified by [RFC2181] Section 5.4.1 Ranking data. "Referrals" is the ranking 7: "Data from the authority section of a non-authoritative answer". And "example.com" NS RRSets attached in the authoritative answer is the ranking 4: "Data from the authority section of an authoritative answer".

4. Problem Statement

[RFC1034] section 4.2.1 states that "the parent side NS RRSets should be exactly the same as the corresponding RRs in the top node of the subzone".

However, people sets different NS RRSets with mistakes, or intentionally. Name server configuration changes will make the differences because the changes take time.

If the zone data of name server(s) specified by referrals and specified by zone apex NS RRSets are different, name resolution becomes unstable. The cache overwrite of NS RRSets may break "Referrals and glue records are information to access servers for child zones" specified by [RFC1034] section 4.2.1.

The overwrite by zone apex NS RRSets induced security vulnerabilities. In 2012, "Ghost Domain Names: Revoked Yet Still Resolvable" [DUAN2012GHOST] was reported. The attack uses updates of NS RRSets attached in authoritative answer. Assume a resolver caches and uses zone apex NS RRSets, and the parent side NS RRSets is updated or removed. The resolver send queries to name servers specified by zone apex NS RRSets and update NS RRSets by the NS RRSets attached in the authority section of the answer. Parent side NS RRSets specifies the existence of delegation, however, resolvers may not check the existence of the parent side NS RRSets and the domain name will remain in the resolvers.

DNS software vendors fixed the problem to restrict the TTL of NS RRSets not to exceed the cached TTL value of old NS RRSets when replacing it.

5. Updating of Resolver Algorithm

5.1. Recommendations to resolvers

Resolvers MUST answer one of the following results: required data, name error, or empty (NODATA) from a server that is authoritative for the query, other name resolution errors (SERVFAIL, REFUSED), or no answer.

Resolvers iterate queries using referrals with corresponding glue records to other name servers. If referrals contain out-of-bailiwick name server names, resolvers need to resolve address records of out-of-bailiwick name servers.

Resolvers MUST NOT use glue records and referrals except iterating delegations. Resolvers MUST NOT use zone apex NS RRSets to iterate.

5.2. Update to resolver algorithm

This document update RFC 1034 Section 5.3.3. Algorithm as follows.

Separate the cache into "authoritative data cache" and "delegation cache". Pre-load root hint information (root NS RRSets and root server addresses) into the delegation cache.

"Step 4.a." is changed as "if the response answers the question or contains a name error, cache the data into authoritative cache as well as returning it back to the client".

"Step 4.b." is changed as if the response contains a better delegation to other servers, cache the delegation information into delegation cache, and go to step 2".

The cache in "Step 1" is the authoritative data cache.

The cache used in "step 2" is the delegation cache. "Set up their addresses using local data" is replaced as "Set up their addresses using the delegation cache".

As a result, RFC 2181 Section 5.4.1 Ranking data becomes useless because the overwrite will not happen. Pre-loaded zone files (or zones retrieved from zone transfer) are treated as answers from authoritative servers. They are treated as static authoritative data, referrals, and glue records. Referrals and glue records in pre-loaded zone files MUST NOT be answered to stub resolvers. They MUST be used to iterate name servers only.

Root zone is special because it is not delegated. Root hint and priming are exceptions because priming replaces pre-configured root hint by root zone apex NS RRSets (authoritative data).

5.3. Characteristics of the update

This update does not change resolver algorithm described in RFC 1034 section 5.3.3, except updates of referrals. It separates authoritative data (possible to answer) and referrals (used to iterate DNS tree). It does not require no special ordering (e.g. trustworthiness and ranking data). It offers more stability of name resolution because the results of traditional name resolution will flap if NS RRSets between the parent and the child are different.

This algorithm is similar to traditional algorithm when the cache is empty.

The update does not effect to DNSSEC [RFC4033] [RFC4034] [RFC4035] because DNSSEC validates authoritative data and does not validate referrals.

The update does not effect to DNS Query Name Minimisation [RFC7816] because answers from authoritative servers don't change. Delegation cache and authoritative data cache separation will need small implementation changes.

5.4. Issues of the update

This update makes impossible to control of TTL value of NS RRSset by the child zone owner. However, overwrite of the referral does not occur always and TTL control may increase queries to authoritative servers.

6. Implementation Ideas

Some implementers already implemented similar algorithm to their products.

7. IANA Considerations

{#:ianacons}

This document has no IANA actions.

8. Security Considerations

9. Acknowledgments

10. Change History

11. References

11.1. Normative References

- [RFC1034] Mockapetris, P., "Domain names - concepts and facilities", STD 13, RFC 1034, DOI 10.17487/RFC1034, November 1987, <<http://www.rfc-editor.org/info/rfc1034>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC2181] Elz, R. and R. Bush, "Clarifications to the DNS Specification", RFC 2181, DOI 10.17487/RFC2181, July 1997, <<http://www.rfc-editor.org/info/rfc2181>>.

Network Working Group
Internet-Draft
Intended status: Best Current Practice
Expires: October 18, 2020

M. Andrews
R. Bellis
ISC
April 16, 2020

A Common Operational Problem in DNS Servers - Failure To Communicate
draft-ietf-dnsop-no-response-issue-23

Abstract

The DNS is a query / response protocol. Failing to respond to queries, or responding incorrectly, causes both immediate operational problems and long term problems with protocol development.

This document identifies a number of common kinds of queries to which some servers either fail to respond or else respond incorrectly. This document also suggests procedures for zone operators to apply to identify and remediate the problem.

The document does not look at the DNS data itself, just the structure of the responses.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on October 18, 2020.

Copyright Notice

Copyright (c) 2020 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of

publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
2. Consequences	4
3. Common kinds of queries that result in no or bad responses.	5
3.1. Basic DNS Queries	5
3.1.1. Zone Existence	5
3.1.2. Unknown / Unsupported Type Queries	5
3.1.3. DNS Flags	6
3.1.4. Unknown DNS opcodes	6
3.1.5. TCP Queries	6
3.2. EDNS Queries	6
3.2.1. EDNS Queries - Version Independent	7
3.2.2. EDNS Queries - Version Specific	7
3.2.3. EDNS Options	7
3.2.4. EDNS Flags	7
3.2.5. Truncated EDNS Responses	8
3.2.6. DO=1 Handling	8
3.2.7. EDNS over TCP	8
4. Firewalls and Load Balancers	8
5. Packet Scrubbing Services	9
6. Whole Answer Caches	10
7. Response Code Selection	10
8. Testing	11
8.1. Testing - Basic DNS	12
8.1.1. Is The Server Configured For The Zone?	12
8.1.2. Testing Unknown Types	12
8.1.3. Testing Header Bits	13
8.1.4. Testing Unknown Opcodes	15
8.1.5. Testing TCP	15
8.2. Testing - Extended DNS	16
8.2.1. Testing Minimal EDNS	16
8.2.2. Testing EDNS Version Negotiation	17
8.2.3. Testing Unknown EDNS Options	17
8.2.4. Testing Unknown EDNS Flags	18
8.2.5. Testing EDNS Version Negotiation With Unknown EDNS Flags	19
8.2.6. Testing EDNS Version Negotiation With Unknown EDNS Options	20
8.2.7. Testing Truncated Responses	20
8.2.8. Testing DO=1 Handling	21

8.2.9. Testing EDNS Version Negotiation With DO=1	21
8.2.10. Testing With Multiple Defined EDNS Options	22
8.3. When EDNS Is Not Supported	22
9. Remediation	23
10. Security Considerations	24
11. IANA Considerations	24
12. Acknowledgements	24
13. References	24
13.1. Normative References	24
13.2. Informative References	25
Authors' Addresses	26

1. Introduction

The DNS [RFC1034], [RFC1035] is a query / response protocol. Failing to respond to queries, or responding incorrectly, causes both immediate operational problems and long term problems with protocol development.

Failure to respond to a query is indistinguishable from packet loss without doing an analysis of query-response patterns. Additionally failure to respond results in unnecessary queries being made by DNS clients, and introduces delays to the resolution process.

Due to the inability to distinguish between packet loss and nameservers or middle boxes dropping EDNS [RFC6891] queries, packet loss is sometimes misclassified as lack of EDNS support which can lead to DNSSEC validation failures.

The existence of servers which fail to respond to queries results in developers being hesitant to deploy new standards. Such servers need to be identified and remediated.

The DNS has response codes that cover almost any conceivable query response. A nameserver should be able to respond to any conceivable query using them. There should be no need to drop queries because a nameserver does not understand them.

Unless a nameserver is under attack, it should respond to all DNS requests directed to it. When a nameserver is under attack it may wish to drop packets. A common attack is to use a nameserver as an amplifier by sending spoofed packets. This is done because response packets are bigger than the queries and large amplification factors are available especially if EDNS is supported. Limiting the rate of responses is reasonable when this is occurring and the client should retry. This however only works if legitimate clients are not being forced to guess whether EDNS queries are accepted or not. As long as there are still a pool of servers that don't respond to EDNS

requests, clients have no way to know if the lack of response is due to packet loss, or EDNS packets not being supported, or rate limiting due to the server being under attack. Misclassification of server behaviour is unavoidable when rate limiting is used until the population of servers which fail to respond to well-formed queries drops to near zero.

Nameservers should respond to queries even if the queried name is not for any name the server is configured to answer for. Misconfigured nameservers are a common occurrence in the DNS and receiving queries for zones that the server is not configured for is not necessarily an indication that the server is under attack. Parent zone operators are advised to regularly check that the delegating NS records are consistent with those of the delegated zone and to correct them when they are not [RFC1034], Section 4.4.2, Paragraph 3. Doing this regularly should reduce the instances of broken delegations.

This document does not try to identify all possible errors nor does it supply an exhaustive list of tests.

2. Consequences

Failure to follow the relevant DNS RFCs has multiple adverse consequences. Some are caused directly by the non-compliant behaviour and others as a result of work-arounds forced on recursive servers. Addressing known issues now will reduce future interoperability issues as the DNS protocol continues to evolve and clients make use of newly-introduced DNS features. In particular the base DNS specification [RFC1034], [RFC1035] and the EDNS specification [RFC6891], when implemented, need to be followed.

Some examples of known consequences include:

- o The AD (Authenticated Data) bit in a response cannot be trusted to mean anything as some servers incorrectly copy the flag bit from the request to the response [RFC1035], [RFC4035]. The use of the AD bit in requests is defined in [RFC6840].
- o Widespread non-response to EDNS queries has led to recursive servers having to assume that EDNS is not supported and that fallback to plain DNS is required, potentially causing DNSSEC validation failures.
- o Widespread non-response to EDNS options requires recursive servers to decide whether to probe to see if it is the specific EDNS option or the use of EDNS in general that is causing the non response. In the limited amount of time required to resolve a query before the client times out this is not possible.

- o Incorrectly returning FORMERR to an EDNS option being present leads to the recursive server not being able to determine if the server is just broken in the handling of the EDNS option or doesn't support EDNS at all.
- o Mishandling of unknown query types has contributed to the abandonment of the transition of the SPF type.
- o Mishandling of unknown query types has slowed up the development of DANE and resulted in additional rules being specified to reduce the probability of interacting with a broken server when making TLSA queries.

The consequences of servers not following the RFCs will only grow if measures are not put in place to remove non compliant servers from the ecosystem. Working around issues due to non-compliance with RFCs is not sustainable.

Most (if not all) of these consequences could have been avoided if action had been taken to remove non-compliant servers as soon as people were aware of them, i.e. to actively seek out broken implementations and servers and inform their developers and operators that they need to fix their servers.

3. Common kinds of queries that result in no or bad responses.

This section is broken down into Basic DNS requests and EDNS requests.

3.1. Basic DNS Queries

3.1.1. Zone Existence

If a zone is delegated to a server, that server should respond to an SOA query for that zone with an SOA record. Failing to respond at all is always incorrect, regardless of the configuration of the server. Responding with anything other than an SOA record in the Answer section indicates a bad delegation.

3.1.2. Unknown / Unsupported Type Queries

Some servers fail to respond to unknown or unsupported types. If a server receives a query for a type that it doesn't recognise, or doesn't implement, it is expected to return the appropriate response as if it did recognise the type but does not have any data for that type: either NOERROR, or NXDOMAIN. The exceptions to this are queries for Meta-RR types which may return NOTIMP.

3.1.3. DNS Flags

Some servers fail to respond to DNS queries with various DNS flags set, regardless of whether they are defined or still reserved. At the time of writing there are servers that fail to respond to queries with the AD flag set to 1 and servers that fail to respond to queries with the last reserved flag set.

Servers should respond to such queries. If the server does not know the meaning of a flag it must not copy it to the response [RFC1035] Section 4.1.1. If the server does not understand the meaning of a request it should reply with a FORMERR response with unknown flags set to zero.

3.1.3.1. Recursive Queries

A non-recursive server is supposed to respond to recursive queries as if the RD bit is not set [RFC1034].

3.1.4. Unknown DNS opcodes

The use of previously undefined opcodes is to be expected. Since the DNS was first defined two new opcodes have been added, UPDATE and NOTIFY.

NOTIMP is the expected rcode to an unknown or unimplemented opcode.

Note: while new opcodes will most probably use the current layout structure for the rest of the message there is no requirement that anything other than the DNS header match.

3.1.5. TCP Queries

All DNS servers are supposed to respond to queries over TCP [RFC7766]. While firewalls should not block TCP connection attempts, those that do they should cleanly terminate the connection by sending TCP RESET or sending ICMP/ICMPv6 Administratively Prohibited messages. Dropping TCP connections introduces excessive delays to the resolution process.

3.2. EDNS Queries

EDNS queries are specified in [RFC6891].

3.2.1. EDNS Queries - Version Independent

Identifying servers that fail to respond to EDNS queries can be done by first confirming that the server responds to regular DNS queries, followed by a series of otherwise identical queries using EDNS, then making the original query again. A series of EDNS queries is needed as at least one DNS implementation responds to the first EDNS query with FORMERR but fails to respond to subsequent queries from the same address for a period until a regular DNS query is made. The EDNS query should specify a UDP buffer size of 512 bytes to avoid false classification of not supporting EDNS due to response packet size.

If the server responds to the first and last queries but fails to respond to most or all of the EDNS queries, it is probably faulty. The test should be repeated a number of times to eliminate the likelihood of a false positive due to packet loss.

Firewalls may also block larger EDNS responses but there is no easy way to check authoritative servers to see if the firewall is mis-configured.

3.2.2. EDNS Queries - Version Specific

Some servers respond correctly to EDNS version 0 queries but fail to respond to EDNS queries with version numbers that are higher than zero. Servers should respond with BADVERS to EDNS queries with version numbers that they do not support.

Some servers respond correctly to EDNS version 0 queries but fail to set QR=1 when responding to EDNS versions they do not support. Such responses may be discarded as invalid (as QR is not 1) or treated as requests (when the source port of the original request was port 53).

3.2.3. EDNS Options

Some servers fail to respond to EDNS queries with EDNS options set. The original EDNS specification left this behaviour undefined [RFC2671], but the correct behaviour was clarified in [RFC6891]. Unknown EDNS options are supposed to be ignored by the server.

3.2.4. EDNS Flags

Some servers fail to respond to EDNS queries with EDNS flags set. Servers should ignore EDNS flags they do not understand and must not add them to the response [RFC6891].

3.2.5. Truncated EDNS Responses

Some EDNS aware servers fail to include an OPT record when a truncated response is sent. An OPT record is supposed to be included in a truncated response [RFC6891].

Some EDNS aware servers fail to honour the advertised EDNS UDP buffer size and send over-sized responses [RFC6891]. Servers must send UDP responses no larger than the advertised EDNS UDP buffer size.

3.2.6. DO=1 Handling

Some nameservers incorrectly only return an EDNS response when the DO bit [RFC3225] is 1 in the query. Servers that support EDNS should always respond to EDNS requests with EDNS responses.

Some nameservers fail to copy the DO bit to the response despite clearly supporting DNSSEC by returning an RRSIG records to EDNS queries with DO=1. Nameservers that support DNSSEC are expected to copy the DO bit from the request to the response.

3.2.7. EDNS over TCP

Some EDNS aware servers incorrectly limit the TCP response sizes to the advertised UDP response size. This breaks DNS resolution to clients where the response sizes exceed the advertised UDP response size despite the server and the client being capable of sending and receiving larger TCP responses respectively. It effectively defeats setting TC=1 in UDP responses.

4. Firewalls and Load Balancers

Firewalls and load balancers can affect the externally visible behaviour of a nameserver. Tests for conformance should to be done from outside of any firewall so that the system is tested as a whole.

Firewalls and load balancers should not drop DNS packets that they don't understand. They should either pass the packets or generate an appropriate error response.

Requests for unknown query types are normal client behaviour and should not be construed as an attack. Nameservers have always been expected to be able to handle such queries.

Requests for unknown query classes are normal client behaviour and should not be construed as an attack. Nameservers have always been expected to be able to handle such queries.

Requests with unknown opcodes are normal client behaviour and should not be construed as an attack. Nameservers have always been expected to be able to handle such queries.

Requests with unassigned flags set (DNS or EDNS) are expected client behaviour and should not be construed as an attack. The behaviour for unassigned flags is to ignore them in the request and to not set them in the response. Dropping DNS / EDNS packets with unassigned flags makes it difficult to deploy extensions that make use of them due to the need to reconfigure and update firewalls.

Requests with unknown EDNS options are expected client behaviour and should not be construed as an attack. The correct behaviour for unknown EDNS options is to ignore their presence when constructing a reply.

Requests with unknown EDNS versions are expected client behaviour and should not be construed as an attack. The correct behaviour for unknown EDNS versions is to return BADVERS along with the highest EDNS version the server supports. Dropping EDNS packets breaks EDNS version negotiation.

Firewalls should not assume that there will only be a single response message to a request. There have been proposals to use EDNS to signal that multiple DNS messages be returned rather than a single UDP message that is fragmented at the IP layer.

DNS, and EDNS in particular, are designed to allow clients to be able to use new features against older servers without having to validate every option. Indiscriminate blocking of messages breaks that design.

However, there may be times when a nameserver mishandles messages with a particular flag, EDNS option, EDNS version field, opcode, type or class field or combination thereof to the point where the integrity of the nameserver is compromised. Firewalls should offer the ability to selectively reject messages using an appropriately constructed response based on all these fields while awaiting a fix from the nameserver vendor. Returning FORMERR or REFUSED are two potential error codes to return.

5. Packet Scrubbing Services

Packet scrubbing services are used to filter out undesired traffic, including but not limited to, denial of service traffic. This is often done using heuristic analysis of the traffic.

Packet scrubbing services can affect the externally visible behaviour of a nameserver in a similar way to firewalls. If an operator uses a packet scrubbing service, they should check that legitimate queries are not being blocked.

Packet scrubbing services, unlike firewalls, are also turned on and off in response to denial of service attacks. One needs to take care when choosing a scrubbing service.

Ideally, Operators should run these tests against a packet scrubbing service to ensure that these tests are not seen as attack vectors.

6. Whole Answer Caches

Whole answer caches take a previously constructed answer and return it to a subsequent query for the same question. However, they can return the wrong response if they do not take all of the relevant attributes of the query into account.

In addition to the standard tuple of <qname,qtype,qclass> a non-exhaustive set of attributes that must be considered include: RD, AD, CD, OPT record, DO, EDNS buffer size, EDNS version, EDNS options, and transport.

7. Response Code Selection

Choosing the correct response code when responding to DNS queries is important. Response codes should be chosen considering how clients will handle them.

For unimplemented opcodes NOTIMP is the expected response code. Note: Newly implemented opcodes may change the message format by extending the header, changing the structure of the records, etc. Servers are not expected to be able to parse these, and should respond with a response code of NOTIMP rather than FORMERR (which would be expected if there was a parse error with an known opcode).

For unimplemented type codes, and in the absence of other errors, the only valid response is NoError if the qname exists, and NameError (NXDOMAIN) otherwise. For Meta-RRs NOTIMP may be returned instead.

If a zone cannot be loaded because it contains unimplemented type codes that are not encoded as unknown record types according to [RFC3597] then the expected response is SERVFAIL as the whole zone should be rejected Section 5.2 [RFC1035]. If a zone loads then Section 4.3.2 [RFC1034] applies.

If the server supports EDNS and receives a query with an unsupported EDNS version, the correct response is BADVERS [RFC6891].

If the server does not support EDNS at all, FORMERR is the expected error code. That said a minimal EDNS server implementation requires parsing the OPT records and responding with an empty OPT record in the additional section in most cases. There is no need to interpret any EDNS options present in the request as unsupported EDNS options are expected to be ignored [RFC6891]. Additionally EDNS flags can be ignored. The only part of the OPT record that needs to be examined is the version field to determine if BADVERS needs to be sent or not.

8. Testing

Testing is divided into two sections: "Basic DNS", which all servers should meet, and "Extended DNS", which should be met by all servers that support EDNS (a server is deemed to support EDNS if it gives a valid EDNS response to any EDNS query). If a server does not support EDNS it should still respond to all the tests, albeit with error responses.

These tests query for records at the apex of a zone that the server is nominally configured to serve. All tests should use the same zone.

It is advisable to run all of the tests below in parallel so as to minimise the delays due to multiple timeouts when the servers do not respond. There are 16 queries directed to each nameserver (assuming no packet loss) testing different aspects of Basic DNS and Extended DNS.

The tests below use dig from BIND 9.11.0 [ISC]. Replace \$zone with the name of the zone being used for testing. Replace \$server with the name or address of the server being tested.

When testing recursive servers set RD=1 and choose a zone name that is known to exist and is not being served by the recursive server. The root zone (".") is often a good candidate as it is DNSSEC signed. RD=1, rather than RD=0, should be present in the responses for all test involving the opcode QUERY. Non-authoritative answers (AA=0) are expected when talking to a recursive server. AD=1 is only expected if the server is validating responses and one or both AD=1 or DO=1 is set in the request otherwise AD=0 is expected.

8.1. Testing - Basic DNS

This first set of tests cover basic DNS server behaviour and all servers should pass these tests.

8.1.1. Is The Server Configured For The Zone?

Ask for the SOA record of the configured zone. This query is made with no DNS flag bits set and without EDNS.

We expect the SOA record for the zone to be returned in the answer section, the rcode to be set to NOERROR, and the AA and QR bits to be set in the header; RA may also be set [RFC1034]. We do not expect an OPT record to be returned [RFC6891].

Verify the server is configured for the zone:

```
dig +noedns +noad +nored soa $zone @$server
```

```
expect: status: NOERROR
expect: the SOA record to be present in the answer section
expect: flag: aa to be present
expect: flag: rd to NOT be present
expect: flag: ad to NOT be present
expect: the OPT record to NOT be present
```

8.1.2. Testing Unknown Types

Identifying servers that fail to respond to unknown or unsupported types can be done by making an initial DNS query for an A record, making a number of queries for an unallocated type, then making a query for an A record again. IANA maintains a registry of allocated types.

If the server responds to the first and last queries but fails to respond to the queries for the unallocated type, it is probably faulty. The test should be repeated a number of times to eliminate the likelihood of a false positive due to packet loss.

Ask for the TYPE1000 RRset at the configured zone's name. This query is made with no DNS flag bits set and without EDNS. TYPE1000 has been chosen for this purpose as IANA is unlikely to allocate this type in the near future and it is not in a range reserved for private use [RFC6895]. Any unallocated type code could be chosen for this test.

We expect no records to be returned in the answer section, the rcode to be set to NOERROR, and the AA and QR bits to be set in the header;

RA may also be set [RFC1034]. We do not expect an OPT record to be returned [RFC6891].

Check that queries for an unknown type work:

```
dig +noedns +noad +noredc type1000 $zone @$server
```

```
expect: status: NOERROR
expect: an empty answer section.
expect: flag: aa to be present
expect: flag: rd to NOT be present
expect: flag: ad to NOT be present
expect: the OPT record to NOT be present
```

8.1.3. Testing Header Bits

8.1.3.1. Testing CD=1 Queries

Ask for the SOA record of the configured zone. This query is made with only the CD DNS flag bit set, all other DNS bits clear, and without EDNS.

We expect the SOA record for the zone to be returned in the answer section, the rcode to be set to NOERROR, and the AA and QR bits to be set in the header. We do not expect an OPT record to be returned.

If the server supports DNSSEC, CD should be set in the response [RFC4035] otherwise CD should be clear [RFC1034].

Check that queries with CD=1 work:

```
dig +noedns +noad +noredc +cd soa $zone @$server
```

```
expect: status: NOERROR
expect: the SOA record to be present in the answer section
expect: flag: aa to be present
expect: flag: rd to NOT be present
expect: flag: ad to NOT be present
expect: the OPT record to NOT be present
```

8.1.3.2. Testing AD=1 Queries

Ask for the SOA record of the configured zone. This query is made with only the AD DNS flag bit set and all other DNS bits clear and without EDNS.

We expect the SOA record for the zone to be returned in the answer section, the rcode to be set to NOERROR, and the AA and QR bits to be

set in the header. We do not expect an OPT record to be returned. The purpose of this query is to detect blocking of queries with the AD bit present, not the specific value of AD in the response.

Check that queries with AD=1 work:

```
dig +noedns +nored +ad soa $zone @$server
```

```
expect: status: NOERROR
expect: the SOA record to be present in the answer section
expect: flag: aa to be present
expect: flag: rd to NOT be present
expect: the OPT record to NOT be present
```

AD use in queries is defined in [RFC6840].

8.1.3.3. Testing Reserved Bit

Ask for the SOA record of the configured zone. This query is made with only the final reserved DNS flag bit set and all other DNS bits clear and without EDNS.

We expect the SOA record for the zone to be returned in the answer section, the rcode to be set to NOERROR, and the AA and QR bits to be set in the header; RA may be set. The final reserved bit must not be set [RFC1034]. We do not expect an OPT record to be returned [RFC6891].

Check that queries with the last unassigned DNS header flag work and that the flag bit is not copied to the response:

```
dig +noedns +noad +nored +zflag soa $zone @$server
```

```
expect: status: NOERROR
expect: the SOA record to be present in the answer section
expect: MBZ to NOT be in the response (see below)
expect: flag: aa to be present
expect: flag: rd to NOT be present
expect: flag: ad to NOT be present
expect: the OPT record to NOT be present
```

MBZ (Must Be Zero) is a dig-specific indication that the flag bit has been incorrectly copied. See Section 4.1.1, [RFC1035] "Z Reserved for future use. Must be zero in all queries and responses."

8.1.3.4. Testing Recursive Queries

Ask for the SOA record of the configured zone. This query is made with only the RD DNS flag bit set and without EDNS.

We expect the SOA record for the zone to be returned in the answer section, the rcode to be set to NOERROR, and the AA, QR and RD bits to be set in the header; RA may also be set [RFC1034]. We do not expect an OPT record to be returned [RFC6891].

Check that recursive queries work:

```
dig +noedns +noad +rec soa $zone @$server
```

```
expect: status: NOERROR
expect: the SOA record to be present in the answer section
expect: flag: aa to be present
expect: flag: rd to be present
expect: flag: ad to NOT be present
expect: the OPT record to NOT be present
```

8.1.4. Testing Unknown Opcodes

Construct a DNS message that consists of only a DNS header with opcode set to 15 (currently not allocated), no DNS header bits set and empty question, answer, authority and additional sections.

Check that new opcodes are handled:

```
dig +noedns +noad +opcode=15 +norec +header-only @$server
```

```
expect: status: NOTIMP
expect: opcode: 15
expect: all sections to be empty
expect: flag: aa to NOT be present
expect: flag: rd to NOT be present
expect: flag: ad to NOT be present
expect: the OPT record to NOT be present
```

8.1.5. Testing TCP

Whether a server accepts TCP connections can be tested by first checking that it responds to UDP queries to confirm that it is up and operating, then attempting the same query over TCP. An additional query should be made over UDP if the TCP connection attempt fails to confirm that the server under test is still operating.

Ask for the SOA record of the configured zone. This query is made with no DNS flag bits set and without EDNS. This query is to be sent using TCP.

We expect the SOA record for the zone to be returned in the answer section, the rcode to be set to NOERROR, and the AA and QR bits to be set in the header; RA may also be set [RFC1034]. We do not expect an OPT record to be returned [RFC6891].

Check that TCP queries work:

```
dig +noedns +noad +nored +tcp soa $zone @$server
```

```
expect: status: NOERROR
expect: the SOA record to be present in the answer section
expect: flag: aa to be present
expect: flag: rd to NOT be present
expect: flag: ad to NOT be present
expect: the OPT record to NOT be present
```

The requirement that TCP be supported is defined in [RFC7766].

8.2. Testing - Extended DNS

The next set of tests cover various aspects of EDNS behaviour. If any of these tests succeed (indicating at least some EDNS support) then all of them should succeed. There are servers that support EDNS but fail to handle plain EDNS queries correctly so a plain EDNS query is not a good indicator of lack of EDNS support.

8.2.1. Testing Minimal EDNS

Ask for the SOA record of the configured zone. This query is made with no DNS flag bits set. EDNS version 0 is used without any EDNS options or EDNS flags set.

We expect the SOA record for the zone to be returned in the answer section, the rcode to be set to NOERROR, and the AA and QR bits to be set in the header; RA may also be set [RFC1034]. We expect an OPT record to be returned. There should be no EDNS flags present in the response. The EDNS version field should be 0 and there should be no EDNS options present [RFC6891].

Check that plain EDNS queries work:

```
dig +nocookie +edns=0 +noad +nored soa $zone @$server
```

```
expect: status: NOERROR
expect: the SOA record to be present in the answer section
expect: an OPT record to be present in the additional section
expect: EDNS Version 0 in response
expect: flag: aa to be present
expect: flag: ad to NOT be present
```

+nocookie disables sending a EDNS COOKIE option which is otherwise enabled by default in BIND 9.11.0 (and later).

8.2.2. Testing EDNS Version Negotiation

Ask for the SOA record of a zone the server is nominally configured to serve. This query is made with no DNS flag bits set. EDNS version 1 is used without any EDNS options or EDNS flags set.

We expect the SOA record for the zone to NOT be returned in the answer section with the extended rcode set to BADVERS and the QR bit to be set in the header; RA may also be set [RFC1034]. We expect an OPT record to be returned. There should be no EDNS flags present in the response. The EDNS version field should be 0 in the response as no other EDNS version has as yet been specified [RFC6891].

Check that EDNS version 1 queries work (EDNS supported):

```
dig +nocookie +edns=1 +noednsneg +noad +nored soa $zone @$server
```

```
expect: status: BADVERS
expect: the SOA record to NOT be present in the answer section
expect: an OPT record to be present in the additional section
expect: EDNS Version 0 in response
expect: flag: aa to NOT be present
expect: flag: ad to NOT be present
```

+noednsneg has been set as dig supports EDNS version negotiation and we want to see only the response to the initial EDNS version 1 query.

8.2.3. Testing Unknown EDNS Options

Ask for the SOA record of the configured zone. This query is made with no DNS flag bits set. EDNS version 0 is used without any EDNS flags. An EDNS option is present with a value that has not yet been assigned by IANA. We have picked an unassigned code of 100 for the

example below. Any unassigned EDNS option code could have been chosen for this test.

We expect the SOA record for the zone to be returned in the answer section, the rcode to be set to NOERROR, and the AA and QR bits to be set in the header; RA may also be set [RFC1034]. We expect an OPT record to be returned. There should be no EDNS flags present in the response. The EDNS version field should be 0 as EDNS versions other than 0 are yet to be specified and there should be no EDNS options present as unknown EDNS options are supposed to be ignored by the server [RFC6891] Section 6.1.2.

Check that EDNS queries with an unknown option work (EDNS supported):

```
dig +nocookie +edns=0 +noad +norec +ednsopt=100 soa $zone @$server
```

```
expect: status: NOERROR
expect: the SOA record to be present in the answer section
expect: an OPT record to be present in the additional section
expect: OPT=100 to NOT be present
expect: EDNS Version 0 in response
expect: flag: aa to be present
expect: flag: ad to NOT be present
```

8.2.4. Testing Unknown EDNS Flags

Ask for the SOA record of the configured zone. This query is made with no DNS flag bits set. EDNS version 0 is used without any EDNS options. An unassigned EDNS flag bit is set (0x40 in this case).

We expect the SOA record for the zone to be returned in the answer section, the rcode to be set to NOERROR, and the AA and QR bits to be set in the header; RA may also be set [RFC1034]. We expect an OPT record to be returned. There should be no EDNS flags present in the response as unknown EDNS flags are supposed to be ignored. The EDNS version field should be 0 and there should be no EDNS options present [RFC6891].

Check that EDNS queries with unknown flags work (EDNS supported):

```
dig +nocookie +edns=0 +noad +norec +ednsflags=0x40 soa $zone @$server
```

```
expect: status: NOERROR
expect: the SOA record to be present in the answer section
expect: an OPT record to be present in the additional section
expect: MBZ not to be present
expect: EDNS Version 0 in response
expect: flag: aa to be present
expect: flag: ad to NOT be present
```

MBZ (Must Be Zero) is a dig-specific indication that a flag bit has been incorrectly copied as per Section 6.1.4, [RFC6891].

8.2.5. Testing EDNS Version Negotiation With Unknown EDNS Flags

Ask for the SOA record of the configured zone. This query is made with no DNS flag bits set. EDNS version 1 is used without any EDNS options. An unassigned EDNS flag bit is set (0x40 in this case).

We expect the SOA record for the zone to NOT be returned in the answer section with the extended rcode set to BADVERS and the QR bit to be set in the header; RA may also be set [RFC1034]. We expect an OPT record to be returned. There should be no EDNS flags present in the response as unknown EDNS flags are supposed to be ignored. The EDNS version field should be 0 as EDNS versions other than 0 are yet to be specified and there should be no EDNS options present [RFC6891].

Check that EDNS version 1 queries with unknown flags work (EDNS supported):

```
dig +nocookie +edns=1 +noednsneg +noad +norec +ednsflags=0x40 soa \
    $zone @$server
```

```
expect: status: BADVERS
expect: SOA record to NOT be present
expect: an OPT record to be present in the additional section
expect: MBZ not to be present
expect: EDNS Version 0 in response
expect: flag: aa to NOT be present
expect: flag: ad to NOT be present
```

8.2.6. Testing EDNS Version Negotiation With Unknown EDNS Options

Ask for the SOA record of the configured zone. This query is made with no DNS flag bits set. EDNS version 1 is used. An unknown EDNS option is present. We have picked an unassigned code of 100 for the example below. Any unassigned EDNS option code could have been chosen for this test.

We expect the SOA record for the zone to NOT be returned in the answer section with the extended rcode set to BADVERS and the QR bit to be set in the header; RA may also be set [RFC1034]. We expect an OPT record to be returned. There should be no EDNS flags present in the response. The EDNS version field should be 0 as EDNS versions other than 0 are yet to be specified and there should be no EDNS options present [RFC6891].

Check that EDNS version 1 queries with unknown options work (EDNS supported):

```
dig +nocookie +edns=1 +noednsneg +noad +nored +ednsopt=100 soa \
    $zone @$server
```

```
expect: status: BADVERS
expect: SOA record to NOT be present
expect: an OPT record to be present in the additional section
expect: OPT=100 to NOT be present
expect: EDNS Version 0 in response
expect: flag: aa to NOT be present
expect: flag: ad to NOT be present
```

8.2.7. Testing Truncated Responses

Ask for the DNSKEY records of the configured zone, which must be a DNSSEC signed zone. This query is made with no DNS flag bits set. EDNS version 0 is used without any EDNS options. The only EDNS flag set is DO. The EDNS UDP buffer size is set to 512. The intention of this query is to elicit a truncated response from the server. Most signed DNSKEY responses are bigger than 512 bytes. This test will not give a valid result if the zone is not signed.

We expect a response, the rcode to be set to NOERROR, and the AA and QR bits to be set, AD may be set in the response if the server supports DNSSEC otherwise it should be clear; TC and RA may also be set [RFC1035] [RFC4035]. We expect an OPT record to be present in the response. There should be no EDNS flags other than DO present in the response. The EDNS version field should be 0 and there should be no EDNS options present [RFC6891].

If TC is not set it is not possible to confirm that the server correctly adds the OPT record to the truncated responses or not.

```
dig +nored +dnssec +bufsize=512 +ignore dnskey $zone @$server
expect: NOERROR
expect: OPT record with version set to 0
```

8.2.8. Testing DO=1 Handling

Ask for the SOA record of the configured zone, which does not need to be DNSSEC signed. This query is made with no DNS flag bits set. EDNS version 0 is used without any EDNS options. The only EDNS flag set is DO.

We expect the SOA record for the zone to be returned in the answer section, the rcode to be set to NOERROR, and the AA and QR bits to be set in the response, AD may be set in the response if the server supports DNSSEC otherwise it should be clear; RA may also be set [RFC1034]. We expect an OPT record to be returned. There should be no EDNS flags other than DO present in the response which should be present if the server supports DNSSEC. The EDNS version field should be 0 and there should be no EDNS options present [RFC6891].

Check that DO=1 queries work (EDNS supported):

```
dig +nocookie +edns=0 +noad +nored +dnssec soa $zone @$server

expect: status: NOERROR
expect: the SOA record to be present in the answer section
expect: an OPT record to be present in the additional section
expect: DO=1 to be present if an RRSIG is in the response
expect: EDNS Version 0 in response
expect: flag: aa to be present
```

8.2.9. Testing EDNS Version Negotiation With DO=1

Ask for the SOA record of the configured zone, which does not need to be DNSSEC signed. This query is made with no DNS flag bits set. EDNS version 1 is used without any EDNS options. The only EDNS flag set is DO.

We expect the SOA record for the zone to NOT be returned in the answer section, the rcode to be set to NOERROR, ; the QR bit and possibly the RA bit to be set [RFC1034]. We expect an OPT record to be returned. There should be no EDNS flags other than DO present in the response which should be there if the server supports DNSSEC. The EDNS version field should be 0 and there should be no EDNS options present [RFC6891].

Check that EDNS version 1, DO=1 queries work (EDNS supported):

```
dig +nocookie +edns=1 +noednsneg +noad +norec +dnssec soa \
    $zone @$server

expect: status: BADVERS
expect: SOA record to NOT be present
expect: an OPT record to be present in the additional section
expect: DO=1 to be present if the EDNS version 0 DNSSEC query test
        returned DO=1
expect: EDNS Version 0 in response
expect: flag: aa to NOT be present
```

8.2.10. Testing With Multiple Defined EDNS Options

Ask for the SOA record of the configured zone. This query is made with no DNS flag bits set. EDNS version 0 is used. A number of defined EDNS options are present (NSID [RFC5001], DNS COOKIE [RFC7873], EDNS Client Subnet [RFC7871] and EDNS Expire [RFC7314]).

We expect the SOA record for the zone to be returned in the answer section, the rcode to be set to NOERROR, and the AA and QR bits to be set in the header; RA may also be set [RFC1034]. We expect an OPT record to be returned. There should be no EDNS flags present in the response. The EDNS version field should be 0. Any of the requested EDNS options supported by the server and permitted server configuration may be returned [RFC6891].

Check that EDNS queries with multiple defined EDNS options work:

```
dig +edns=0 +noad +norec +cookie +nsid +expire +subnet=0.0.0.0/0 \
    soa $zone @$server

expect: status: NOERROR
expect: the SOA record to be present in the answer section
expect: an OPT record to be present in the additional section
expect: EDNS Version 0 in response
expect: flag: aa to be present
expect: flag: ad to NOT be present
```

8.3. When EDNS Is Not Supported

If EDNS is not supported by the nameserver, we expect a response to each of the above queries. That response may be a FORMERR error response or the OPT record may just be ignored.

Some nameservers only return a EDNS response when a particular EDNS option or flag (e.g. DO=1) is present in the request. This

behaviour is not compliant behaviour and may hide other incorrect behaviour from the above tests. Re-testing with the triggering option / flag present will expose this misbehaviour.

9. Remediation

Nameserver operators are generally expected to test their own infrastructure for compliance to standards. The above tests should be run when new systems are brought online, and should be repeated periodically to ensure continued interoperability.

Domain registrants who do not maintain their own DNS infrastructure are entitled to a DNS service that conforms to standards and interoperates well. Registrants who become aware that their DNS operator does not have a well maintained or compliant infrastructure should insist that their service provider correct issues, and switch providers if they do not.

In the event that an operator experiences problems due to the behaviour of nameservers outside their control, the above tests will help in narrowing down the precise issue(s) which can then be reported to the relevant party.

If contact information for the operator of a misbehaving nameserver is not already known, the following methods of communication could be considered:

- o the RNAME of the zone authoritative for the name of the misbehaving server
- o the RNAME of zones for which the offending server is authoritative
- o administrative or technical contacts listed in the registration information for the parent domain of the name of the misbehaving server, or for zones for which the nameserver is authoritative
- o the registrar or registry for such zones
- o DNS-specific operational fora (e.g. mailing lists)

Operators of parent zones may wish to regularly test the authoritative nameservers of their child zones. However, parent operators can have widely varying capabilities in terms of notification or remediation depending on whether they have a direct relationship with the child operator. Many TLD registries, for example, cannot directly contact their registrants and may instead need to communicate through the relevant registrar. In such cases it may be most efficient for registrars to take on the responsibility

for testing the name servers of their registrants, since they have a direct relationship.

When notification is not effective at correcting problems with a misbehaving nameserver, parent operators can choose to remove NS record sets (and glue records below) that refer to the faulty server until the servers are fixed. This should only be done as a last resort and with due consideration, as removal of a delegation can have unanticipated side effects. For example, other parts of the DNS tree may depend on names below the removed zone cut, and the parent operator may find themselves responsible for causing new DNS failures to occur.

10. Security Considerations

Testing protocol compliance can potentially result in false reports of attempts to attack services from Intrusion Detection Services and firewalls. All of the tests are well-formed (though not necessarily common) DNS queries. None of the tests listed above should cause any harm to a protocol-compliant server.

Relaxing firewall settings to ensure EDNS compliance could potentially expose a critical implementation flaw in the nameserver. Nameservers should be tested for conformance before relaxing firewall settings.

When removing delegations for non-compliant servers there can be a knock on effect on other zones that require these zones to be operational for the nameservers addresses to be resolved.

11. IANA Considerations

There are no actions for IANA.

12. Acknowledgements

The contributions of the following are gratefully acknowledged:

Matthew Pounsett, Tim Wicinski.

13. References

13.1. Normative References

- [RFC1034] Mockapetris, P., "Domain names - concepts and facilities", STD 13, RFC 1034, DOI 10.17487/RFC1034, November 1987, <<https://www.rfc-editor.org/info/rfc1034>>.

- [RFC1035] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, DOI 10.17487/RFC1035, November 1987, <<https://www.rfc-editor.org/info/rfc1035>>.
- [RFC3225] Conrad, D., "Indicating Resolver Support of DNSSEC", RFC 3225, DOI 10.17487/RFC3225, December 2001, <<https://www.rfc-editor.org/info/rfc3225>>.
- [RFC4035] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "Protocol Modifications for the DNS Security Extensions", RFC 4035, DOI 10.17487/RFC4035, March 2005, <<https://www.rfc-editor.org/info/rfc4035>>.
- [RFC6840] Weiler, S., Ed. and D. Blacka, Ed., "Clarifications and Implementation Notes for DNS Security (DNSSEC)", RFC 6840, DOI 10.17487/RFC6840, February 2013, <<https://www.rfc-editor.org/info/rfc6840>>.
- [RFC6891] Damas, J., Graff, M., and P. Vixie, "Extension Mechanisms for DNS (EDNS(0))", STD 75, RFC 6891, DOI 10.17487/RFC6891, April 2013, <<https://www.rfc-editor.org/info/rfc6891>>.
- [RFC6895] Eastlake 3rd, D., "Domain Name System (DNS) IANA Considerations", BCP 42, RFC 6895, DOI 10.17487/RFC6895, April 2013, <<https://www.rfc-editor.org/info/rfc6895>>.
- [RFC7766] Dickinson, J., Dickinson, S., Bellis, R., Mankin, A., and D. Wessels, "DNS Transport over TCP - Implementation Requirements", RFC 7766, DOI 10.17487/RFC7766, March 2016, <<https://www.rfc-editor.org/info/rfc7766>>.

13.2. Informative References

- [ISC] "Internet Systems Consortium", <<https://www.isc.org/>>.
- [RFC2671] Vixie, P., "Extension Mechanisms for DNS (EDNS0)", RFC 2671, DOI 10.17487/RFC2671, August 1999, <<https://www.rfc-editor.org/info/rfc2671>>.
- [RFC3597] Gustafsson, A., "Handling of Unknown DNS Resource Record (RR) Types", RFC 3597, DOI 10.17487/RFC3597, September 2003, <<https://www.rfc-editor.org/info/rfc3597>>.
- [RFC5001] Austein, R., "DNS Name Server Identifier (NSID) Option", RFC 5001, DOI 10.17487/RFC5001, August 2007, <<https://www.rfc-editor.org/info/rfc5001>>.

- [RFC7314] Andrews, M., "Extension Mechanisms for DNS (EDNS) EXPIRE Option", RFC 7314, DOI 10.17487/RFC7314, July 2014, <<https://www.rfc-editor.org/info/rfc7314>>.
- [RFC7871] Contavalli, C., van der Gaast, W., Lawrence, D., and W. Kumari, "Client Subnet in DNS Queries", RFC 7871, DOI 10.17487/RFC7871, May 2016, <<https://www.rfc-editor.org/info/rfc7871>>.
- [RFC7873] Eastlake 3rd, D. and M. Andrews, "Domain Name System (DNS) Cookies", RFC 7873, DOI 10.17487/RFC7873, May 2016, <<https://www.rfc-editor.org/info/rfc7873>>.

Authors' Addresses

M. Andrews
Internet Systems Consortium
PO Box 360
Newmarket, NH 03857
US

Email: marka@isc.org

Ray Bellis
Internet Systems Consortium
PO Box 360
Newmarket, NH 03857
US

Email: ray@isc.org

DNSOP Working Group
Internet-Draft
Updates: 1035, 7766 (if approved)
Intended status: Standards Track
Expires: June 9, 2019

R. Bellis
ISC
S. Cheshire
Apple Inc.
J. Dickinson
S. Dickinson
Sinodun
T. Lemon
Nibbhaya Consulting
T. Pusateri
Unaffiliated
December 06, 2018

DNS Stateful Operations
draft-ietf-dnsop-session-signal-20

Abstract

This document defines a new DNS OPCODE for DNS Stateful Operations (DSO). DSO messages communicate operations within persistent stateful sessions, using type-length-value (TLV) syntax. Three TLVs are defined that manage session timeouts, termination, and encryption padding, and a framework is defined for extensions to enable new stateful operations. This document updates RFC 1035 by adding a new DNS header opcode which has different message semantics, and a new result code. This document updates RFC 7766 by redefining a session, providing new guidance on connection re-use, and providing a new mechanism for handling session idle timeouts.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on June 9, 2019.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
2. Requirements Language	5
3. Terminology	6
4. Applicability	9
4.1. Use Cases	9
4.1.1. Session Management	9
4.1.2. Long-lived Subscriptions	9
4.2. Applicable Transports	10
5. Protocol Details	11
5.1. DSO Session Establishment	12
5.1.1. Session Establishment Failure	13
5.1.2. Session Establishment Success	14
5.2. Operations After Session Establishment	14
5.3. Session Termination	15
5.3.1. Handling Protocol Errors	15
5.4. Message Format	16
5.4.1. DNS Header Fields in DSO Messages	17
5.4.2. DSO Data	19
5.4.3. TLV Syntax	21
5.4.4. EDNS(0) and TSIG	24
5.5. Message Handling	25
5.5.1. Delayed Acknowledgement Management	26
5.5.2. MESSAGE ID Namespaces	27
5.5.3. Error Responses	28
5.6. Responder-Initiated Operation Cancellation	29
6. DSO Session Lifecycle and Timers	30
6.1. DSO Session Initiation	30
6.2. DSO Session Timeouts	31
6.3. Inactive DSO Sessions	32
6.4. The Inactivity Timeout	33
6.4.1. Closing Inactive DSO Sessions	33

6.4.2.	Values for the Inactivity Timeout	34
6.5.	The Keepalive Interval	35
6.5.1.	Keepalive Interval Expiry	35
6.5.2.	Values for the Keepalive Interval	35
6.6.	Server-Initiated Session Termination	37
6.6.1.	Server-Initiated Retry Delay Message	38
6.6.2.	Misbehaving Clients	39
6.6.3.	Client Reconnection	39
7.	Base TLVs for DNS Stateful Operations	41
7.1.	Keepalive TLV	41
7.1.1.	Client handling of received Session Timeout values	43
7.1.2.	Relationship to edns-tcp-keepalive EDNS0 Option	44
7.2.	Retry Delay TLV	45
7.2.1.	Retry Delay TLV used as a Primary TLV	45
7.2.2.	Retry Delay TLV used as a Response Additional TLV	47
7.3.	Encryption Padding TLV	48
8.	Summary Highlights	49
8.1.	QR bit and MESSAGE ID	49
8.2.	TLV Usage	50
9.	Additional Considerations	52
9.1.	Service Instances	52
9.2.	Anycast Considerations	53
9.3.	Connection Sharing	54
9.4.	Operational Considerations for Middlebox	55
9.5.	TCP Delayed Acknowledgement Considerations	56
10.	IANA Considerations	59
10.1.	DSO OPCODE Registration	59
10.2.	DSO RCODE Registration	59
10.3.	DSO Type Code Registry	59
11.	Security Considerations	60
11.1.	TLS 0-RTT Considerations	61
12.	Acknowledgements	62
13.	References	62
13.1.	Normative References	62
13.2.	Informative References	63
	Authors' Addresses	65

1. Introduction

This document specifies a mechanism for managing stateful DNS connections. DNS most commonly operates over a UDP transport, but can also operate over streaming transports; the original DNS RFC specifies DNS over TCP [RFC1035] and a profile for DNS over TLS [RFC7858] has been specified. These transports can offer persistent, long-lived sessions and therefore when using them for transporting DNS messages it is of benefit to have a mechanism that can establish parameters associated with those sessions, such as timeouts. In such

situations it is also advantageous to support server-initiated messages (such as DNS Push Notifications [I-D.ietf-dnssd-push]).

The existing EDNS(0) Extension Mechanism for DNS [RFC6891] is explicitly defined to only have "per-message" semantics. While EDNS(0) has been used to signal at least one session-related parameter (edns-tcp-keepalive EDNS0 Option [RFC7828]) the result is less than optimal due to the restrictions imposed by the EDNS(0) semantics and the lack of server-initiated signalling. For example, a server cannot arbitrarily instruct a client to close a connection because the server can only send EDNS(0) options in responses to queries that contained EDNS(0) options.

This document defines a new DNS OPCODE, DSO ([TBA1], tentatively 6), for DNS Stateful Operations. DSO messages are used to communicate operations within persistent stateful sessions, expressed using type-length-value (TLV) syntax. This document defines an initial set of three TLVs, used to manage session timeouts, termination, and encryption padding.

All three TLVs defined here are mandatory for all implementations of DSO. Further TLVs may be defined in additional specifications.

DSO messages may or may not be acknowledged; this is signalled by providing a non-zero message ID for messages that must be acknowledged (DSO request messages) and a zero message ID for messages that are not to be acknowledged (DSO unidirectional messages), and is also specified in the definition of a particular DSO message type. Messages are pipelined; answers may appear out of order when more than one answer is pending.

The format for DSO messages (Section 5.4) differs somewhat from the traditional DNS message format used for standard queries and responses. The standard twelve-byte header is used, but the four count fields (QDCOUNT, ANCOUNT, NSCOUNT, ARCOUNT) are set to zero and accordingly their corresponding sections are not present.

The actual data pertaining to DNS Stateful Operations (expressed in TLV syntax) is appended to the end of the DNS message header. Just as in traditional DNS over TCP [RFC1035] [RFC7766] the stream protocol carrying DSO messages (which are just another kind of DNS message) frames them by putting a 16-bit message length at the start, so the length of the DSO message is determined from that length, rather than from any of the DNS header counts.

When displayed using packet analyzer tools that have not been updated to recognize the DSO format, this will result in the DSO data being

displayed as unknown additional data after the end of the DNS message.

This new format has distinct advantages over an RR-based format because it is more explicit and more compact. Each TLV definition is specific to its use case, and as a result contains no redundant or overloaded fields. Importantly, it completely avoids conflating DNS Stateful Operations in any way with normal DNS operations or with existing EDNS(0)-based functionality. A goal of this approach is to avoid the operational issues that have befallen EDNS(0), particularly relating to middlebox behaviour (see for example [I-D.ietf-dnsop-no-response-issue] sections 3.2 and 4).

With EDNS(0), multiple options may be packed into a single OPT pseudo-RR, and there is no generalized mechanism for a client to be able to tell whether a server has processed or otherwise acted upon each individual option within the combined OPT pseudo-RR. The specifications for each individual option need to define how each different option is to be acknowledged, if necessary.

In contrast to EDNS(0), with DSO there is no compelling motivation to pack multiple operations into a single message for efficiency reasons, because DSO always operates using a connection-oriented transport protocol. Each DSO operation is communicated in its own separate DNS message, and the transport protocol can take care of packing several DNS messages into a single IP packet if appropriate. For example, TCP can pack multiple small DNS messages into a single TCP segment. This simplification allows for clearer semantics. Each DSO request message communicates just one primary operation, and the RCODE in the corresponding response message indicates the success or failure of that operation.

2. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. Terminology

DSO: DNS Stateful Operations.

connection: a bidirectional byte (or message) stream, where the bytes (or messages) are delivered reliably and in-order, such as provided by using DNS over TCP [RFC1035] [RFC7766] or DNS over TLS [RFC7858].

session: The unqualified term "session" in the context of this document refers to a persistent network connection between two endpoints which allows for the exchange of DNS messages over a connection where either end of the connection can send messages to the other end. (The term has no relationship to the "session layer" of the OSI "seven-layer model".)

DSO Session: a session established between two endpoints that acknowledge persistent DNS state via the exchange of DSO messages over the connection. This is distinct from a DNS-over-TCP session as described in the previous specification for DNS over TCP [RFC7766].

close gracefully: a normal session shutdown, where the client closes the TCP connection to the server using a graceful close, such that no data is lost (e.g., using TCP FIN, see Section 5.3).

forcibly abort: a session shutdown as a result of a fatal error, where the TCP connection is unilaterally aborted without regard for data loss (e.g., using TCP RST, see Section 5.3).

server: the software with a listening socket, awaiting incoming connection requests, in the usual DNS sense.

client: the software which initiates a connection to the server's listening socket, in the usual DNS sense.

initiator: the software which sends a DSO request message or a DSO unidirectional message during a DSO session. Either a client or server can be an initiator

responder: the software which receives a DSO request message or a DSO unidirectional message during a DSO

session. Either a client or server can be a responder.

sender: the software which is sending a DNS message, a DSO message, a DNS response, or a DSO response.

receiver: the software which is receiving a DNS message, a DSO message, a DNS response, or a DSO response.

service instance: a specific instance of server software running on a specific host (Section 9.1).

long-lived operation: a long-lived operation is an outstanding operation on a DSO session where either the client or server, acting as initiator, has requested that the responder send new information regarding the request, as it becomes available.

Early Data: A TLS 1.3 handshake containing early data that begins a DSO session ([RFC8446] section 2.3). TCP Fast Open is only permitted when using TLS.

DNS message: any DNS message, including DNS queries, response, updates, DSO messages, etc.

DNS request message: any DNS message where the QR bit is 0.

DNS response message: any DNS message where the QR bit is 1.

DSO message: a DSO request message, DSO unidirectional message, or a DSO response to a DSO request message. If the QR bit is 1 in a DSO message, it is a DSO response message. If the QR bit is 0 in a DSO message, it is a DSO request message or DSO unidirectional message, as determined by the specification of its primary TLV.

DSO response message: a response to a DSO request message.

DSO request message: a DSO message that requires a response.

DSO unidirectional message: a DSO message that does not require and cannot induce a response.

Primary TLV: The first TLV in a DSO message or DSO response; in the DSO message this determines the nature of the operation being performed.

Additional TLV: Any TLVs in a DSO message response that follow the primary TLV.

Response Primary TLV: The (optional) first TLV in a DSO response.

Response Additional TLV: Any TLVs in a DSO response that follow the (optional) Response Primary TLV.

inactivity timer: the time since the most recent non-keepalive DNS message was sent or received. (see Section 6.4)

keepalive timer: the time since the most recent DNS message was sent or received. (see Section 6.5)

session timeouts: the inactivity timer and the keepalive timer.

inactivity timeout: the maximum value that the inactivity timer can have before the connection is gracefully closed.

keepalive interval: the maximum value that the keepalive timer can have before the client is required to send a keepalive. (see Section 7.1)

resetting a timer: setting the timer value to zero and restarting the timer.

clearing a timer: setting the timer value to zero but not restarting the timer.

4. Applicability

DNS Stateful Operations are applicable to several known use cases and are only applicable on transports that are capable of supporting a DSO Session.

4.1. Use Cases

There are several use cases for DNS Stateful operations that can be described here.

4.1.1. Session Management

Firstly, establishing session parameters such as server-defined timeouts is of great use in the general management of persistent connections. For example, using DSO sessions for stub-to-recursive DNS-over-TLS [RFC7858] is more flexible for both the client and the server than attempting to manage sessions using just the edns-tcp-keepalive EDNS0 Option [RFC7828]. The simple set of TLVs defined in this document is sufficient to greatly enhance connection management for this use case.

4.1.2. Long-lived Subscriptions

Secondly, DNS-SD [RFC6763] has evolved into a naturally session-based mechanism where, for example, long-lived subscriptions lend themselves to 'push' mechanisms as opposed to polling. Long-lived stateful connections and server-initiated messages align with this use case [I-D.ietf-dnssd-push].

A general use case is that DNS traffic is often bursty but session establishment can be expensive. One challenge with long-lived connections is to maintain sufficient traffic to maintain NAT and firewall state. To mitigate this issue this document introduces a new concept for the DNS, that is DSO "Keepalive traffic". This traffic carries no DNS data and is not considered 'activity' in the classic DNS sense, but serves to maintain state in middleboxes, and to assure client and server that they still have connectivity to each other.

4.2. Applicable Transports

DNS Stateful Operations are applicable in cases where it is useful to maintain an open session between a DNS client and server, where the transport allows such a session to be maintained, and where the transport guarantees in-order delivery of messages, on which DSO depends. Examples of transports that can support DNS Stateful Operations are DNS-over-TCP [RFC1035] [RFC7766] and DNS-over-TLS [RFC7858].

Note that in the case of DNS over TLS, there is no mechanism for upgrading from DNS-over-TCP to DNS-over-TLS mid-connection (see [RFC7858] section 7). A connection is either DNS-over-TCP from the start, or DNS-over-TLS from the start.

DNS Stateful Operations are not applicable for transports that cannot support clean session semantics, or that do not guarantee in-order delivery. While in principle such a transport could be constructed over UDP, the current DNS specification over UDP transport [RFC1035] does not provide in-order delivery or session semantics, and hence cannot be used. Similarly, DNS-over-HTTP [I-D.ietf-doh-dns-over-https] cannot be used because HTTP has its own mechanism for managing sessions, and this is incompatible with the mechanism specified here.

No other transports are currently defined for use with DNS Stateful Operations. Such transports can be added in the future, if they meet the requirements set out in the first paragraph of this section.

5. Protocol Details

The overall flow of DNS Stateful Operations goes through a series of phases:

Connection Establishment: A client establishes a connection to a server. (Section 4.2)

Connected but sessionless: A connection exists, but a DSO session has not been established. DNS messages can be sent from the client to server, and DNS responses can be sent from servers to clients. In this state a client that wishes to use DSO can attempt to establish a DSO session (Section 5.1). Standard DNS-over-TCP inactivity timeout handling is in effect [RFC7766] (see Section 7.1.2).

DSO Session Establishment in Progress: A client has sent a DSO request, but has not yet received a DSO response. In this phase, the client may send more DSO requests and more DNS requests, but **MUST NOT** send DSO unidirectional messages (Section 5.1).

DSO Session Establishment Failed: The attempt to establish the DSO session did not succeed. At this point, the client is permitted to continue operating without a DSO session (Connected but Sessionless) but does not send further DSO messages (Section 5.1).

DSO Session Established: Both client and server may send DSO messages and DNS messages; both may send replies in response to messages they receive (Section 5.2). The inactivity timer (Section 6.4) is active; the keepalive timer (Section 6.5) is active. Standard DNS-over-TCP inactivity timeout handling is no longer in effect [RFC7766] (see Section 7.1.2).

Server Shutdown: The server has decided to gracefully terminate the session, and has sent the client a Retry Delay message (Section 6.6.1). There may still be unprocessed messages from the client; the server will ignore these. The server will not send any further messages to the client (Section 6.6.1.1).

Client Shutdown: The client has decided to disconnect, either because it no longer needs service, the connection is inactive (Section 6.4.1), or because the server sent it a Retry Delay message (Section 6.6.1). The client closes the connection gracefully Section 5.3.

Reconnect: The client disconnected as a result of a server shutdown. The client either waits for the server-specified Retry Delay to expire (Section 6.6.3), or else contacts a different server

instance. If the client no longer needs service, it does not reconnect.

Forcibly Abort: The client or server detected a protocol error, and further communication would have undefined behavior. The client or server forcibly aborts the connection (Section 5.3).

Abort Reconnect Wait: The client has forcibly aborted the connection, but still needs service. Or, the server forcibly aborted the connection, but the client still needs service. The client either connects to a different service instance (Section 9.1) or waits to reconnect (Section 6.6.3.1).

5.1. DSO Session Establishment

In order for a session to be established between a client and a server, the client must first establish a connection to the server, using an applicable transport (see Section 4).

In some environments it may be known in advance by external means that both client and server support DSO, and in these cases either client or server may initiate DSO messages at any time. In this case, the session is established as soon as the connection is established; this is referred to as implicit session establishment.

However, in the typical case a server will not know in advance whether a client supports DSO, so in general, unless it is known in advance by other means that a client does support DSO, a server **MUST NOT** initiate DSO request messages or DSO unidirectional messages until a DSO Session has been mutually established by at least one successful DSO request/response exchange initiated by the client, as described below. This is referred to as explicit session establishment.

Until a DSO session has been implicitly or explicitly established, a client **MUST NOT** initiate DSO unidirectional messages.

A DSO Session is established over a connection by the client sending a DSO request message, such as a DSO Keepalive request message (Section 7.1), and receiving a response, with matching MESSAGE ID, and RCODE set to NOERROR (0), indicating that the DSO request was successful.

Some DSO messages are permitted as early data (Section 11.1). Others are not. Unidirectional messages are never permitted as early data unless an implicit session exists.

If a server receives a DSO message in early data whose primary TLV is not permitted to appear in early data, the server MUST forcibly abort the connection. If a client receives a DSO message in early data, and there is no implicit DSO session, the client MUST forcibly abort the connection. This can only be enforced on TLS connections; therefore, servers MUST NOT enable TFO when listening for a connection that does not require TLS.

5.1.1. Session Establishment Failure

If the response RCODE is set to NOTIMP (4), or in practise any value other than NOERROR (0) or DSOTYPENI (defined below), then the client MUST assume that the server does not implement DSO at all. In this case the client is permitted to continue sending DNS messages on that connection, but the client MUST NOT issue further DSO messages on that connection.

If the RCODE in the response is set to DSOTYPENI ("DSO-TYPE Not Implemented", [TBA2] tentatively RCODE 11) this indicates that the server does support DSO, but does not implement the DSO-TYPE of the primary TLV in this DSO request message. A server implementing DSO MUST NOT return DSOTYPENI for a DSO Keepalive request message, because the Keepalive TLV is mandatory to implement. But in the future, if a client attempts to establish a DSO Session using a response-requiring DSO request message using some newly-defined DSO-TYPE that the server does not understand, that would result in a DSOTYPENI response. If the server returns DSOTYPENI then a DSO Session is not considered established, but the client is permitted to continue sending DNS messages on the connection, including other DSO messages such as the DSO Keepalive, which may result in a successful NOERROR response, yielding the establishment of a DSO Session.

Two other possibilities exist: the server might drop the connection, or the server might send no response to the DSO message.

In the first case, the client SHOULD mark that service instance as not supporting DSO, and not attempt a DSO connection for some period of time (at least an hour) after the failed attempt. The client MAY reconnect but not use DSO, if appropriate (Section 6.6.3.2).

In the second case, the client SHOULD wait 30 seconds, after which time the server will be assumed not to support DSO. If the server doesn't respond within 30 seconds, the client MUST forcibly abort the connection to the server, since the server's behavior is out of spec, and hence its state is undefined. The client MAY reconnect, but not use DSO, if appropriate (Section 6.6.3.1).

5.1.2. Session Establishment Success

When the server receives a DSO request message from a client, and transmits a successful NOERROR response to that request, the server considers the DSO Session established.

When the client receives the server's NOERROR response to its DSO request message, the client considers the DSO Session established.

Once a DSO Session has been established, either end may unilaterally send appropriate DSO messages at any time, and therefore either client or server may be the initiator of a message.

5.2. Operations After Session Establishment

Once a DSO Session has been established, clients and servers should behave as described in this specification with regard to inactivity timeouts and session termination, not as previously prescribed in the earlier specification for DNS over TCP [RFC7766].

Because a server that supports DNS Stateful Operations MUST return an RCODE of NOERROR when it receives a Keepalive TLV DSO request message, the Keepalive TLV is an ideal candidate for use in establishing a DSO session. Any other option that can only succeed when sent to a server of the desired kind is also a good candidate for use in establishing a DSO session. For clients that implement only the DSO-TYPES defined in this base specification, sending a Keepalive TLV is the only DSO request message they have available to initiate a DSO Session. Even for clients that do implement other future DSO-TYPES, for simplicity they MAY elect to always send an initial DSO Keepalive request message as their way of initiating a DSO Session. A future definition of a new response-requiring DSO-TYPE gives implementers the option of using that new DSO-TYPE if they wish, but does not change the fact that sending a Keepalive TLV remains a valid way of initiating a DSO Session.

5.3. Session Termination

A "DSO Session" is terminated when the underlying connection is closed. Sessions are "closed gracefully" as a result of the server closing a session because it is overloaded, the client closing the session because it is done, or the client closing the session because it is inactive. Sessions are "forcibly aborted" when either the client or server closes the connection because of a protocol error.

- o Where this specification says, "close gracefully," that means sending a TLS close_notify (if TLS is in use) followed by a TCP FIN, or the equivalents for other protocols. Where this specification requires a connection to be closed gracefully, the requirement to initiate that graceful close is placed on the client, to place the burden of TCP's TIME-WAIT state on the client rather than the server.
- o Where this specification says, "forcibly abort," that means sending a TCP RST, or the equivalent for other protocols. In the BSD Sockets API this is achieved by setting the SO_LINGER option to zero before closing the socket.

5.3.1. Handling Protocol Errors

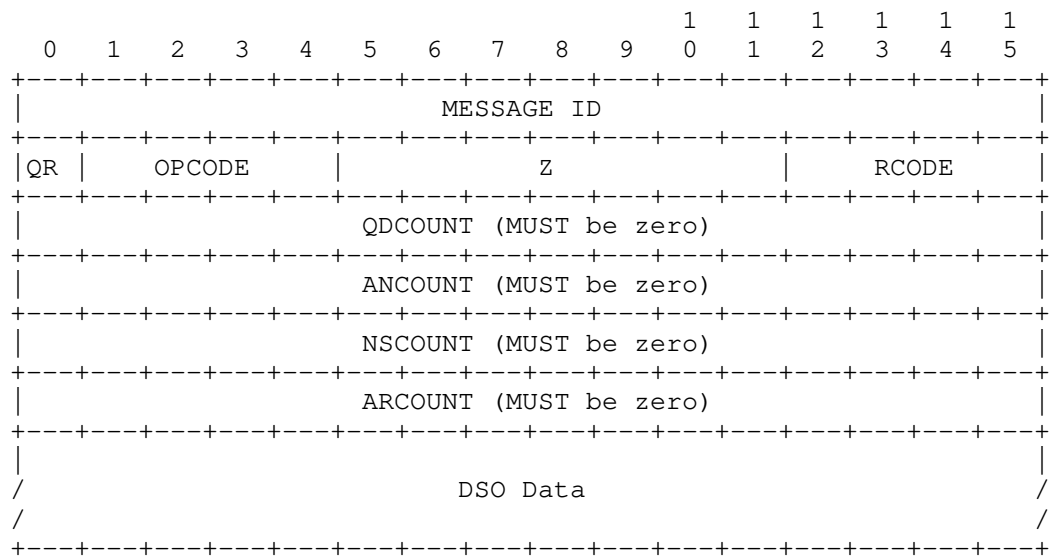
In protocol implementation there are generally two kinds of errors that software writers have to deal with. The first is situations that arise due to factors in the environment, such as temporary loss of connectivity. While undesirable, these situations do not indicate a flaw in the software, and they are situations that software should generally be able to recover from.

The second is situations that should never happen when communicating with a compliant DSO implementation. If they do happen, they indicate a serious flaw in the protocol implementation, beyond what it is reasonable to expect software to recover from. This document describes this latter form of error condition as a "fatal error" and specifies that an implementation encountering a fatal error condition "MUST forcibly abort the connection immediately".

5.4. Message Format

A DSO message begins with the standard twelve-byte DNS message header [RFC1035] with the OPCODE field set to the DSO OPCODE. However, unlike standard DNS messages, the question section, answer section, authority records section and additional records sections are not present. The corresponding count fields (QDCOUNT, ANCOUNT, NSCOUNT, ARCOUNT) MUST be set to zero on transmission.

If a DSO message is received where any of the count fields are not zero, then a FORMERR MUST be returned.



5.4.1.1. DNS Header Fields in DSO Messages

In a DSO unidirectional message the MESSAGE ID field MUST be set to zero. In a DSO request message the MESSAGE ID field MUST be set to a unique nonzero value, that the initiator is not currently using for any other active operation on this connection. For the purposes here, a MESSAGE ID is in use in this DSO Session if the initiator has used it in a DSO request message for which it is still awaiting a response, or if the client has used it to set up a long-lived operation that has not yet been cancelled. For example, a long-lived operation could be a Push Notification subscription [I-D.ietf-dnssd-push] or a Discovery Relay interface subscription [I-D.ietf-dnssd-mdns-relay].

Whether a message is a DSO request message or a DSO unidirectional message is determined only by the specification for the Primary TLV. An acknowledgment cannot be requested by including a nonzero message ID in a message that is required according to its primary TLV to be unidirectional. Nor can an acknowledgment be prevented by sending a message ID of zero in a message that is required to be a DSO request message according to its primary TLV. A responder that receives either such malformed message MUST treat it as a fatal error and forcibly abort the connection immediately.

In a DSO request message or DSO unidirectional message the DNS Header QR bit MUST be zero (QR=0). If the QR bit is not zero the message is not a DSO request or DSO unidirectional message.

In a DSO response message the DNS Header QR bit MUST be one (QR=1). If the QR bit is not one, the message is not a response message.

In a DSO response message (QR=1) the MESSAGE ID field MUST contain a copy of the value of the MESSAGE ID field in the DSO request message being responded to. In a DSO response message (QR=1) the MESSAGE ID field MUST NOT be zero. If a DSO response message (QR=1) is received where the MESSAGE ID is zero this is a fatal error and the recipient MUST forcibly abort the connection immediately.

The DNS Header OPCODE field holds the DSO OPCODE value.

The Z bits are currently unused in DSO messages, and in both DSO request messages and DSO responses the Z bits MUST be set to zero (0) on transmission and MUST be ignored on reception.

In a DSO request message (QR=0) the RCODE is set according to the definition of the request. For example, in a Retry Delay message (Section 6.6.1) the RCODE indicates the reason for termination. However, in most cases, except where clearly specified otherwise, in

a DSO request message (QR=0) the RCODE is set to zero on transmission, and silently ignored on reception.

The RCODE value in a response message (QR=1) may be one of the following values:

Code	Mnemonic	Description
0	NOERROR	Operation processed successfully
1	FORMERR	Format error
2	SERVFAIL	Server failed to process DSO request message due to a problem with the server
4	NOTIMP	DSO not supported
5	REFUSED	Operation declined for policy reasons
[TBA2] 11	DSOTYPENI	Primary TLV's DSO-Type is not implemented

Use of the above RCODEs is likely to be common in DSO but does not preclude the definition and use of other codes in future documents that make use of DSO.

If a document defining a new DSO-TYPE makes use of response codes not defined here, then that document MUST specify the specific interpretation of those RCODE values in the context of that new DSO TLV.

5.4.2. DSO Data

The standard twelve-byte DNS message header with its zero-valued count fields is followed by the DSO Data, expressed using TLV syntax, as described below in Section 5.4.3.

A DSO request message or DSO unidirectional message MUST contain at least one TLV. The first TLV in a DSO request message or DSO unidirectional message is referred to as the "Primary TLV" and determines the nature of the operation being performed, including whether it is a DSO request or a DSO unidirectional operation. In some cases it may be appropriate to include other TLVs in a DSO request message or DSO unidirectional message, such as the Encryption Padding TLV (Section 7.3), and these extra TLVs are referred to as the "Additional TLVs" and are not limited to what is defined in this document. New "Additional TLVs" may be defined in the future and those definitions will describe when their use is appropriate.

A DSO response message may contain no TLVs, or it may be specified to contain one or more TLVs appropriate to the information being communicated. This includes "Primary TLVs" and "Additional TLVs" defined in this document as well as in future TLV definitions. It may be permissible for an additional TLV to appear in a response to a primary TLV even though the specification of that primary TLV does not specify it explicitly. See Section 8.2 for more information.

A DSO response message may contain one or more TLVs with the Primary TLV DSO-TYPE the same as the Primary TLV from the corresponding DSO request message or it may contain zero or more Additional TLVs only. The MESSAGE ID field in the DNS message header is sufficient to identify the DSO request message to which this response message relates.

A DSO response message may contain one or more TLVs with DSO-TYPES different from the Primary TLV from the corresponding DSO request message, in which case those TLV(s) are referred to as "Response Additional TLVs".

Response Primary TLV(s), if present, MUST occur first in the response message, before any Response Additional TLVs.

It is anticipated that most DSO operations will be specified to use DSO request messages, which generate corresponding DSO responses. In some specialized high-traffic use cases, it may be appropriate to specify DSO unidirectional messages. DSO unidirectional messages can be more efficient on the network, because they don't generate a stream of corresponding reply messages. Using DSO unidirectional messages can also simplify software in some cases, by removing need

for an initiator to maintain state while it waits to receive replies it doesn't care about. When the specification for a particular TLV states that, when used as a Primary TLV (i.e., first) in an outgoing DSO request message (i.e., QR=0), that message is to be unidirectional, the MESSAGE ID field MUST be set to zero and the receiver MUST NOT generate any response message corresponding to this DSO unidirectional message.

The previous point, that the receiver MUST NOT generate responses to DSO unidirectional messages, applies even in the case of errors.

When a DSO message is received where both the QR bit and the MESSAGE ID field are zero, the receiver MUST NOT generate any response. For example, if the DSO-TYPE in the Primary TLV is unrecognized, then a DSOTYPENI error MUST NOT be returned; instead the receiver MUST forcibly abort the connection immediately.

DSO unidirectional messages MUST NOT be used "speculatively" in cases where the sender doesn't know if the receiver supports the Primary TLV in the message, because there is no way to receive any response to indicate success or failure. DSO unidirectional messages are only appropriate in cases where the sender already knows that the receiver supports, and wishes to receive, these messages.

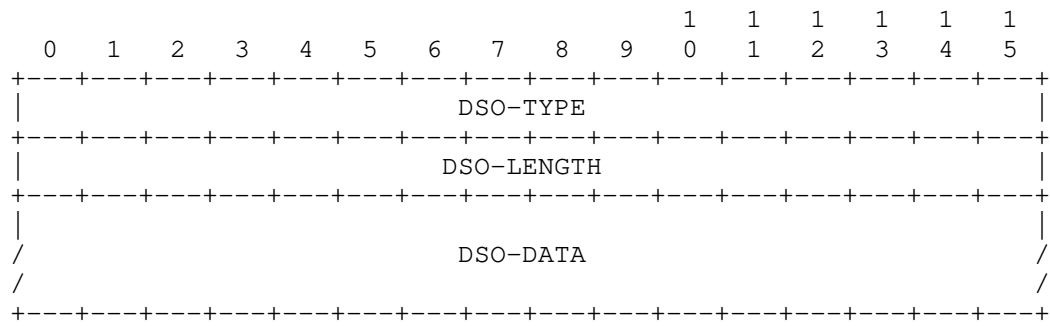
For example, after a client has subscribed for Push Notifications [I-D.ietf-dnssd-push], the subsequent event notifications are then sent as DSO unidirectional messages, and this is appropriate because the client initiated the message stream by virtue of its Push Notification subscription, thereby indicating its support of Push Notifications, and its desire to receive those notifications.

Similarly, after a Discovery Relay client has subscribed to receive inbound mDNS (multicast DNS, [RFC6762]) traffic from a Discovery Relay, the subsequent stream of received packets is then sent using DSO unidirectional messages, and this is appropriate because the client initiated the message stream by virtue of its Discovery Relay link subscription, thereby indicating its support of Discovery Relay, and its desire to receive inbound mDNS packets over that DSO session [I-D.ietf-dnssd-mdns-relay].

5.4.3. TLV Syntax

All TLVs, whether used as "Primary", "Additional", "Response Primary", or "Response Additional", use the same encoding syntax.

Specifications that define new TLVs must specify whether the DSO-TYPE can be used as the Primary TLV, used as an Additional TLV, or used in either context, both in the case of requests and of responses. The specification for a TLV must also state whether, when used as the Primary (i.e., first) TLV in a DSO message (i.e., QR=0), that DSO message is unidirectional or is a request message which requires a response. If the DSO message requires a response, the specification must also state which TLVs, if any, are to be included in the response. The Primary TLV may or may not be contained in the response, depending on what is specified for that TLV.



DSO-TYPE: A 16-bit unsigned integer, in network (big endian) byte order, giving the DSO-TYPE of the current DSO TLV per the IANA DSO Type Code Registry.

DSO-LENGTH: A 16-bit unsigned integer, in network (big endian) byte order, giving the size in bytes of the DSO-DATA.

DSO-DATA: Type-code specific format. The generic DSO machinery treats the DSO-DATA as an opaque "blob" without attempting to interpret it. Interpretation of the meaning of the DSO-DATA for a particular DSO-TYPE is the responsibility of the software that implements that DSO-TYPE.

5.4.3.1. Request TLVs

The first TLV in a DSO request message or DSO unidirectional message is the "Primary TLV" and indicates the operation to be performed. A DSO request message or DSO unidirectional message **MUST** contain at least one TLV—the Primary TLV.

Immediately following the Primary TLV, a DSO request message or DSO unidirectional message **MAY** contain one or more "Additional TLVs", which specify additional parameters relating to the operation.

5.4.3.2. Response TLVs

Depending on the operation, a DSO response message **MAY** contain no TLVs, because it is simply a response to a previous DSO request message, and the MESSAGE ID in the header is sufficient to identify the DSO request in question. Or it may contain a single response TLV, with the same DSO-TYPE as the Primary TLV in the request message. Alternatively it may contain one or more TLVs of other types, or a combination of the above, as appropriate for the information that needs to be communicated. The specification for each DSO TLV determines what TLVs are required in a response to a DSO request message using that TLV.

If a DSO response is received for an operation where the specification requires that the response carry a particular TLV or TLVs, and the required TLV(s) are not present, then this is a fatal error and the recipient of the defective response message **MUST** forcibly abort the connection immediately.

5.4.3.3. Unrecognized TLVs

If DSO request message is received containing an unrecognized Primary TLV, with a nonzero MESSAGE ID (indicating that a response is expected), then the receiver MUST send an error response with matching MESSAGE ID, and RCODE DSOTYPENI. The error response MUST NOT contain a copy of the unrecognized Primary TLV.

If DSO unidirectional message is received containing an unrecognized Primary TLV, with a zero MESSAGE ID (indicating that no response is expected), then this is a fatal error and the recipient MUST forcibly abort the connection immediately.

If a DSO request message or DSO unidirectional message is received where the Primary TLV is recognized, containing one or more unrecognized Additional TLVs, the unrecognized Additional TLVs MUST be silently ignored, and the remainder of the message is interpreted and handled as if the unrecognized parts were not present.

Similarly, if a DSO response message is received containing one or more unrecognized TLVs, the unrecognized TLVs MUST be silently ignored, and the remainder of the message is interpreted and handled as if the unrecognized parts were not present.

5.4.4. EDNS(0) and TSIG

Since the ARCOUNT field MUST be zero, a DSO message cannot contain a valid EDNS(0) option in the additional records section. If functionality provided by current or future EDNS(0) options is desired for DSO messages, one or more new DSO TLVs need to be defined to carry the necessary information.

For example, the EDNS(0) Padding Option [RFC7830] used for security purposes is not permitted in a DSO message, so if message padding is desired for DSO messages then the Encryption Padding TLV described in Section 7.3 MUST be used.

A DSO message can't contain a TSIG record, because a TSIG record is included in the additional section of the message, which would mean that ARCOUNT would be greater than zero. DSO messages are required to have an ARCOUNT of zero. Therefore, if use of signatures with DSO messages becomes necessary in the future, a new DSO TLV would have to be defined to perform this function.

Note however that, while DSO *messages* cannot include EDNS(0) or TSIG records, a DSO *session* is typically used to carry a whole series of DNS messages of different kinds, including DSO messages, and other DNS message types like Query [RFC1034] [RFC1035] and Update [RFC2136], and those messages can carry EDNS(0) and TSIG records.

Although messages may contain other EDNS(0) options as appropriate, this specification explicitly prohibits use of the edns-tcp-keepalive EDNS0 Option [RFC7828] in *any* messages sent on a DSO Session (because it is obsoleted by the functionality provided by the DSO Keepalive operation). If any message sent on a DSO Session contains an edns-tcp-keepalive EDNS0 Option this is a fatal error and the recipient of the defective message MUST forcibly abort the connection immediately.

5.5. Message Handling

As described above in Section 5.4.1, whether an outgoing DSO message with the QR bit in the DNS header set to zero is a DSO request or DSO unidirectional message is determined by the specification for the Primary TLV, which in turn determines whether the MESSAGE ID field in that outgoing message will be zero or nonzero.

Every DSO message with the QR bit in the DNS header set to zero and a nonzero MESSAGE ID field is a DSO request message, and MUST elicit a corresponding response, with the QR bit in the DNS header set to one and the MESSAGE ID field set to the value given in the corresponding DSO request message.

Valid DSO request messages sent by the client with a nonzero MESSAGE ID field elicit a response from the server, and valid DSO request messages sent by the server with a nonzero MESSAGE ID field elicit a response from the client.

Every DSO message with both the QR bit in the DNS header and the MESSAGE ID field set to zero is a DSO unidirectional message, and MUST NOT elicit a response.

5.5.1. Delayed Acknowledgement Management

Generally, most good TCP implementations employ a delayed acknowledgement timer to provide more efficient use of the network and better performance.

With a bidirectional exchange over TCP, as for example with a DSO request message, the operating system TCP implementation waits for the application-layer client software to generate the corresponding DSO response message. It can then send a single combined packet containing the TCP acknowledgement, the TCP window update, and the application-generated DSO response message. This is more efficient than sending three separate packets, as would occur if the TCP packet containing the DSO request were acknowledged immediately.

With a DSO unidirectional message or DSO response message, there is no corresponding application-generated DSO response message, and consequently, no hint to the transport protocol about when it should send its acknowledgement and window update.

Some networking APIs provide a mechanism that allows the application-layer client software to signal to the transport protocol that no response will be forthcoming (in effect it can be thought of as a zero-length "empty" write). Where available in the networking API being used, the recipient of a DSO unidirectional message or DSO response message, having parsed and interpreted the message, SHOULD then use this mechanism provided by the networking API to signal that no response for this message will be forthcoming, so that the TCP implementation can go ahead and send its acknowledgement and window update without further delay. See Section 9.5 for further discussion of why this is important.

5.5.2. MESSAGE ID Namespaces

The namespaces of 16-bit MESSAGE IDs are independent in each direction. This means it is **not** an error for both client and server to send DSO request messages at the same time as each other, using the same MESSAGE ID, in different directions. This simplification is necessary in order for the protocol to be implementable. It would be infeasible to require the client and server to coordinate with each other regarding allocation of new unique MESSAGE IDs. It is also not necessary to require the client and server to coordinate with each other regarding allocation of new unique MESSAGE IDs. The value of the 16-bit MESSAGE ID combined with the identity of the initiator (client or server) is sufficient to unambiguously identify the operation in question. This can be thought of as a 17-bit message identifier space, using message identifiers 0x00001-0x0FFFF for client-to-server DSO request messages, and message identifiers 0x10001-0x1FFFF for server-to-client DSO request messages. The least-significant 16 bits are stored explicitly in the MESSAGE ID field of the DSO message, and the most-significant bit is implicit from the direction of the message.

As described above in Section 5.4.1, an initiator **MUST NOT** reuse a MESSAGE ID that it already has in use for an outstanding DSO request message (unless specified otherwise by the relevant specification for the DSO-TYPE in question). At the very least, this means that a MESSAGE ID can't be reused in a particular direction on a particular DSO Session while the initiator is waiting for a response to a previous DSO request message using that MESSAGE ID on that DSO Session (unless specified otherwise by the relevant specification for the DSO-TYPE in question), and for a long-lived operation the MESSAGE ID for the operation can't be reused while that operation remains active.

If a client or server receives a response (QR=1) where the MESSAGE ID is zero, or is any other value that does not match the MESSAGE ID of any of its outstanding operations, this is a fatal error and the recipient **MUST** forcibly abort the connection immediately.

If a responder receives a DSO request message (QR=0) where the MESSAGE ID is not zero, and the responder tracks request MESSAGE IDs, and the MESSAGE ID matches the MESSAGE ID of a DSO request message it received for which a response has not yet been sent, it **MUST** forcibly abort the connection immediately. This behavior is required to prevent a hypothetical attack that takes advantage of undefined behavior in this case. However, if the responder does not track MESSAGE IDs in this way, no such risk exists, so tracking MESSAGE IDs just to implement this sanity check is not required.

5.5.3. Error Responses

When a DSO unidirectional message type is received (MESSAGE ID field is zero), the receiver should already be expecting this DSO message type. Section 5.4.3.3 describes the handling of unknown DSO message types. Parsing errors MUST also result in the receiver forcibly aborting the connection. When a DSO unidirectional message of an unexpected type is received, the receiver SHOULD forcibly abort the connection. Whether the connection should be forcibly aborted for other internal errors processing the DSO unidirectional message is implementation dependent, according to the severity of the error.

When a DSO request message is unsuccessful for some reason, the responder returns an error code to the initiator.

In the case of a server returning an error code to a client in response to an unsuccessful DSO request message, the server MAY choose to end the DSO Session, or MAY choose to allow the DSO Session to remain open. For error conditions that only affect the single operation in question, the server SHOULD return an error response to the client and leave the DSO Session open for further operations.

For error conditions that are likely to make all operations unsuccessful in the immediate future, the server SHOULD return an error response to the client and then end the DSO Session by sending a Retry Delay message, as described in Section 6.6.1.

Upon receiving an error response from the server, a client SHOULD NOT automatically close the DSO Session. An error relating to one particular operation on a DSO Session does not necessarily imply that all other operations on that DSO Session have also failed, or that future operations will fail. The client should assume that the server will make its own decision about whether or not to end the DSO Session, based on the server's determination of whether the error condition pertains to this particular operation, or would also apply to any subsequent operations. If the server does not end the DSO Session by sending the client a Retry Delay message (Section 6.6.1) then the client SHOULD continue to use that DSO Session for subsequent operations.

5.6. Responder-Initiated Operation Cancellation

This document, the base specification for DNS Stateful Operations, does not itself define any long-lived operations, but it defines a framework for supporting long-lived operations, such as Push Notification subscriptions [I-D.ietf-dnssd-push] and Discovery Relay interface subscriptions [I-D.ietf-dnssd-mdns-relay].

Long-lived operations, if successful, will remain active until the initiator terminates the operation.

However, it is possible that a long-lived operation may be valid at the time it was initiated, but then a later change of circumstances may render that operation invalid. For example, a long-lived client operation may pertain to a name that the server is authoritative for, but then the server configuration is changed such that it is no longer authoritative for that name.

In such cases, instead of terminating the entire session it may be desirable for the responder to be able to cancel selectively only those operations that have become invalid.

The responder performs this selective cancellation by sending a new response message, with the MESSAGE ID field containing the MESSAGE ID of the long-lived operation that is to be terminated (that it had previously acknowledged with a NOERROR RCODE), and the RCODE field of the new response message giving the reason for cancellation.

After a response message with nonzero RCODE has been sent, that operation has been terminated from the responder's point of view, and the responder sends no more messages relating to that operation.

After a response message with nonzero RCODE has been received by the initiator, that operation has been terminated from the initiator's point of view, and the cancelled operation's MESSAGE ID is now free for reuse.

6. DSO Session Lifecycle and Timers

6.1. DSO Session Initiation

A DSO Session begins as described in Section 5.1.

The client may perform as many DNS operations as it wishes using the newly created DSO Session. When the client has multiple messages to send, it SHOULD NOT wait for each response before sending the next message.

The server MUST act on messages in the order they are received, but SHOULD NOT delay sending responses to those messages as they become available in order to return them in the order the requests were received.

Section 6.2.1.1 of the DNS-over-TCP specification [RFC7766] specifies this in more detail.

6.2. DSO Session Timeouts

Two timeout values are associated with a DSO Session: the inactivity timeout, and the keepalive interval. Both values are communicated in the same TLV, the Keepalive TLV (Section 7.1).

The first timeout value, the inactivity timeout, is the maximum time for which a client may speculatively keep an inactive DSO Session open in the expectation that it may have future requests to send to that server.

The second timeout value, the keepalive interval, is the maximum permitted interval between messages if the client wishes to keep the DSO Session alive.

The two timeout values are independent. The inactivity timeout may be lower, the same, or higher than the keepalive interval, though in most cases the inactivity timeout is expected to be shorter than the keepalive interval.

A shorter inactivity timeout with a longer keepalive interval signals to the client that it should not speculatively keep an inactive DSO Session open for very long without reason, but when it does have an active reason to keep a DSO Session open, it doesn't need to be sending an aggressive level of DSO keepalive traffic to maintain that session. An example of this would be a client that has subscribed to DNS Push notifications: in this case, the client is not sending any traffic to the server, but the session is not inactive, because there is a active request to the server to receive push notifications.

A longer inactivity timeout with a shorter keepalive interval signals to the client that it may speculatively keep an inactive DSO Session open for a long time, but to maintain that inactive DSO Session it should be sending a lot of DSO keepalive traffic. This configuration is expected to be less common.

In the usual case where the inactivity timeout is shorter than the keepalive interval, it is only when a client has a long-lived, low-traffic, operation that the keepalive interval comes into play, to ensure that a sufficient residual amount of traffic is generated to maintain NAT and firewall state and to assure client and server that they still have connectivity to each other.

On a new DSO Session, if no explicit DSO Keepalive message exchange has taken place, the default value for both timeouts is 15 seconds.

For both timeouts, lower values of the timeout result in higher network traffic, and higher CPU load on the server.

6.3. Inactive DSO Sessions

At both servers and clients, the generation or reception of any complete DNS message (including DNS requests, responses, updates, DSO messages, etc.) resets both timers for that DSO Session, with the one exception that a DSO Keepalive message resets only the keepalive timer, not the inactivity timeout timer.

In addition, for as long as the client has an outstanding operation in progress, the inactivity timer remains cleared, and an inactivity timeout cannot occur.

For short-lived DNS operations like traditional queries and updates, an operation is considered in progress for the time between request and response, typically a period of a few hundred milliseconds at most. At the client, the inactivity timer is cleared upon transmission of a request and remains cleared until reception of the corresponding response. At the server, the inactivity timer is cleared upon reception of a request and remains cleared until transmission of the corresponding response.

For long-lived DNS Stateful operations (such as a Push Notification subscription [I-D.ietf-dnssd-push] or a Discovery Relay interface subscription [I-D.ietf-dnssd-mdns-relay]), an operation is considered in progress for as long as the operation is active, i.e. until it is cancelled. This means that a DSO Session can exist, with active operations, with no messages flowing in either direction, for far longer than the inactivity timeout, and this is not an error. This is why there are two separate timers: the inactivity timeout, and the keepalive interval. Just because a DSO Session has no traffic for an extended period of time does not automatically make that DSO Session "inactive", if it has an active operation that is awaiting events.

6.4. The Inactivity Timeout

The purpose of the inactivity timeout is for the server to balance the trade off between the costs of setting up new DSO Sessions and the costs of maintaining inactive DSO Sessions. A server with abundant DSO Session capacity can offer a high inactivity timeout, to permit clients to keep a speculative DSO Session open for a long time, to save the cost of establishing a new DSO Session for future communications with that server. A server with scarce memory resources can offer a low inactivity timeout, to cause clients to promptly close DSO Sessions whenever they have no outstanding operations with that server, and then create a new DSO Session later when needed.

6.4.1. Closing Inactive DSO Sessions

When a connection's inactivity timeout is reached the client **MUST** begin closing the idle connection, but a client is not required to keep an idle connection open until the inactivity timeout is reached. A client **MAY** close a DSO Session at any time, at the client's discretion. If a client determines that it has no current or reasonably anticipated future need for a currently inactive DSO Session, then the client **SHOULD** gracefully close that connection.

If, at any time during the life of the DSO Session, the inactivity timeout value (i.e., 15 seconds by default) elapses without there being any operation active on the DSO Session, the client **MUST** close the connection gracefully.

If, at any time during the life of the DSO Session, twice the inactivity timeout value (i.e., 30 seconds by default), or five seconds, if twice the inactivity timeout value is less than five seconds, elapses without there being any operation active on the DSO Session, the server **MUST** consider the client delinquent, and **MUST** forcibly abort the DSO Session.

In this context, an operation being active on a DSO Session includes a query waiting for a response, an update waiting for a response, or an active long-lived operation, but not a DSO Keepalive message exchange itself. A DSO Keepalive message exchange resets only the keepalive interval timer, not the inactivity timeout timer.

If the client wishes to keep an inactive DSO Session open for longer than the default duration then it uses the DSO Keepalive message to request longer timeout values, as described in Section 7.1.

6.4.2. Values for the Inactivity Timeout

For the inactivity timeout value, lower values result in more frequent DSO Session teardown and re-establishment. Higher values result in lower traffic and lower CPU load on the server, but higher memory burden to maintain state for inactive DSO Sessions.

A server may dictate any value it chooses for the inactivity timeout (either in a response to a client-initiated request, or in a server-initiated message) including values under one second, or even zero.

An inactivity timeout of zero informs the client that it should not speculatively maintain idle connections at all, and as soon as the client has completed the operation or operations relating to this server, the client should immediately begin closing this session.

A server will forcibly abort an idle client session after twice the inactivity timeout value, or five seconds, whichever is greater. In the case of a zero inactivity timeout value, this means that if a client fails to close an idle client session then the server will forcibly abort the idle session after five seconds.

An inactivity timeout of 0xFFFFFFFF represents "infinity" and informs the client that it may keep an idle connection open as long as it wishes. Note that after granting an unlimited inactivity timeout in this way, at any point the server may revise that inactivity timeout by sending a new DSO Keepalive message dictating new Session Timeout values to the client.

The largest *finite* inactivity timeout supported by the current Keepalive TLV is 0xFFFFFFFFE ($2^{32}-2$ milliseconds, approximately 49.7 days).

6.5. The Keepalive Interval

The purpose of the keepalive interval is to manage the generation of sufficient messages to maintain state in middleboxes (such as NAT gateways or firewalls) and for the client and server to periodically verify that they still have connectivity to each other. This allows them to clean up state when connectivity is lost, and to establish a new session if appropriate.

6.5.1. Keepalive Interval Expiry

If, at any time during the life of the DSO Session, the keepalive interval value (i.e., 15 seconds by default) elapses without any DNS messages being sent or received on a DSO Session, the client **MUST** take action to keep the DSO Session alive, by sending a DSO Keepalive message (Section 7.1). A DSO Keepalive message exchange resets only the keepalive timer, not the inactivity timer.

If a client disconnects from the network abruptly, without cleanly closing its DSO Session, perhaps leaving a long-lived operation uncanceled, the server learns of this after failing to receive the required DSO keepalive traffic from that client. If, at any time during the life of the DSO Session, twice the keepalive interval value (i.e., 30 seconds by default) elapses without any DNS messages being sent or received on a DSO Session, the server **SHOULD** consider the client delinquent, and **SHOULD** forcibly abort the DSO Session.

6.5.2. Values for the Keepalive Interval

For the keepalive interval value, lower values result in a higher volume of DSO keepalive traffic. Higher values of the keepalive interval reduce traffic and CPU load, but have minimal effect on the memory burden at the server, because clients keep a DSO Session open for the same length of time (determined by the inactivity timeout) regardless of the level of DSO keepalive traffic required.

It may be appropriate for clients and servers to select different keepalive interval values depending on the nature of the network they are on.

A corporate DNS server that knows it is serving only clients on the internal network, with no intervening NAT gateways or firewalls, can impose a higher keepalive interval, because frequent DSO keepalive traffic is not required.

A public DNS server that is serving primarily residential consumer clients, where it is likely there will be a NAT gateway on the path,

may impose a lower keepalive interval, to generate more frequent DSO keepalive traffic.

A smart client may be adaptive to its environment. A client using a private IPv4 address [RFC1918] to communicate with a DNS server at an address outside that IPv4 private address block, may conclude that there is likely to be a NAT gateway on the path, and accordingly request a lower keepalive interval.

By default it is RECOMMENDED that clients request, and servers grant, a keepalive interval of 60 minutes. This keepalive interval provides for reasonably timely detection if a client abruptly disconnects without cleanly closing the session, and is sufficient to maintain state in firewalls and NAT gateways that follow the IETF recommended Best Current Practice that the "established connection idle-timeout" used by middleboxes be at least 2 hours 4 minutes [RFC5382] [RFC7857].

Note that the lower the keepalive interval value, the higher the load on client and server. Moreover for a keep-alive value that is smaller than the time needed for the transport to retransmit, a single packet loss would cause a server to overzealously abort the connect. For example, a (hypothetical and unrealistic) keepalive interval value of 100 ms would result in a continuous stream of ten messages per second or more (if allowed by the current congestion control window), in both directions, to keep the DSO Session alive. And, in this extreme example, a single retransmission over a path with, e.g., 100ms RTT would introduce a momentary pause in the stream of messages, long enough to cause the server to abort the connection.

Because of this concern, the server MUST NOT send a DSO Keepalive message (either a response to a client-initiated request, or a server-initiated message) with a keepalive interval value less than ten seconds. If a client receives a DSO Keepalive message specifying a keepalive interval value less than ten seconds this is a fatal error and the client MUST forcibly abort the connection immediately.

A keepalive interval value of 0xFFFFFFFF represents "infinity" and informs the client that it should generate no DSO keepalive traffic. Note that after signaling that the client should generate no DSO keepalive traffic in this way, at any point the server may revise that DSO keepalive traffic requirement by sending a new DSO Keepalive message dictating new Session Timeout values to the client.

The largest *finite* keepalive interval supported by the current Keepalive TLV is 0xFFFFFFF (2³²-2 milliseconds, approximately 49.7 days).

6.6. Server-Initiated Session Termination

In addition to cancelling individual long-lived operations selectively (Section 5.6) there are also occasions where a server may need to terminate one or more entire sessions. An entire session may need to be terminated if the client is defective in some way, or departs from the network without closing its session. Sessions may also need to be terminated if the server becomes overloaded, or if the server is reconfigured and lacks the ability to be selective about which operations need to be cancelled.

This section discusses various reasons a session may be terminated, and the mechanisms for doing so.

In normal operation, closing a DSO Session is the client's responsibility. The client makes the determination of when to close a DSO Session based on an evaluation of both its own needs, and the inactivity timeout value dictated by the server. A server only causes a DSO Session to be ended in the exceptional circumstances outlined below. Some of the exceptional situations in which a server may terminate a DSO Session include:

- o The server application software or underlying operating system is shutting down or restarting.
- o The server application software terminates unexpectedly (perhaps due to a bug that makes it crash, causing the underlying operating system to send a TCP RST).
- o The server is undergoing a reconfiguration or maintenance procedure, that, due to the way the server software is implemented, requires clients to be disconnected. For example, some software is implemented such that it reads a configuration file at startup, and changing the server's configuration entails modifying the configuration file and then killing and restarting the server software, which generally entails a loss of network connections.
- o The client fails to meet its obligation to generate the required DSO keepalive traffic, or to close an inactive session by the prescribed time (twice the time interval dictated by the server, or five seconds, whichever is greater, as described in Section 6.2).
- o The client sends a grossly invalid or malformed request that is indicative of a seriously defective client implementation.
- o The server is over capacity and needs to shed some load.

6.6.1. Server-Initiated Retry Delay Message

In the cases described above where a server elects to terminate a DSO Session, it could do so simply by forcibly aborting the connection. However, if it did this the likely behavior of the client might be simply to treat this as a network failure and reconnect immediately, putting more burden on the server.

Therefore, to avoid this reconnection implosion, a server SHOULD instead choose to shed client load by sending a Retry Delay message, with an appropriate RCODE value informing the client of the reason the DSO Session needs to be terminated. The format of the Retry Delay TLV, and the interpretations of the various RCODE values, are described in Section 7.2. After sending a Retry Delay message, the server MUST NOT send any further messages on that DSO Session.

The server MAY randomize retry delays in situations where many retry delays are sent in quick succession, so as to avoid all the clients attempting to reconnect at once. In general, implementations should avoid using the Retry Delay message in a way that would result in many clients reconnecting at the same time, if every client attempts to reconnect at the exact time specified.

Upon receipt of a Retry Delay message from the server, the client MUST make note of the reconnect delay for this server, and then immediately close the connection gracefully.

After sending a Retry Delay message the server SHOULD allow the client five seconds to close the connection, and if the client has not closed the connection after five seconds then the server SHOULD forcibly abort the connection.

A Retry Delay message MUST NOT be initiated by a client. If a server receives a Retry Delay message this is a fatal error and the server MUST forcibly abort the connection immediately.

6.6.1.1. Outstanding Operations

At the instant a server chooses to initiate a Retry Delay message there may be DNS requests already in flight from client to server on this DSO Session, which will arrive at the server after its Retry Delay message has been sent. The server MUST silently ignore such incoming requests, and MUST NOT generate any response messages for them. When the Retry Delay message from the server arrives at the client, the client will determine that any DNS requests it previously sent on this DSO Session, that have not yet received a response, now will certainly not be receiving any response. Such requests should

be considered failed, and should be retried at a later time, as appropriate.

In the case where some, but not all, of the existing operations on a DSO Session have become invalid (perhaps because the server has been reconfigured and is no longer authoritative for some of the names), but the server is terminating all affected DSO Sessions en masse by sending them all a Retry Delay message, the reconnect delay MAY be zero, indicating that the clients SHOULD immediately attempt to re-establish operations.

It is likely that some of the attempts will be successful and some will not, depending on the nature of the reconfiguration.

In the case where a server is terminating a large number of DSO Sessions at once (e.g., if the system is restarting) and the server doesn't want to be inundated with a flood of simultaneous retries, it SHOULD send different reconnect delay values to each client. These adjustments MAY be selected randomly, pseudorandomly, or deterministically (e.g., incrementing the time value by one tenth of a second for each successive client, yielding a post-restart reconnection rate of ten clients per second).

6.6.2. Misbehaving Clients

A server may determine that a client is not following the protocol correctly. There may be no way for the server to recover the session, in which case the server forcibly terminates the connection. Since the client doesn't know why the connection dropped, it may reconnect immediately. If the server has determined that a client is not following the protocol correctly, it may terminate the DSO session as soon as it is established, specifying a long retry-delay to prevent the client from immediately reconnecting.

6.6.3. Client Reconnection

After a DSO Session is ended by the server (either by sending the client a Retry Delay message, or by forcibly aborting the underlying transport connection) the client SHOULD try to reconnect, to that service instance, or to another suitable service instance, if more than one is available. If reconnecting to the same service instance, the client MUST respect the indicated delay, if available, before attempting to reconnect. Clients should not attempt to randomize the delay; the server will randomly jitter the retry delay values it sends to each client if this behavior is desired.

If the service instance will only be out of service for a short maintenance period, it should use a value a little longer than the

expected maintenance window. It should not default to a very large delay value, or clients may not attempt to reconnect after it resumes service.

If a particular service instance does not want a client to reconnect ever (perhaps the service instance is being de-commissioned), it SHOULD set the retry delay to the maximum value 0xFFFFFFFF (2³²-1 milliseconds, approximately 49.7 days). It is not possible to instruct a client to stay away for longer than 49.7 days. If, after 49.7 days, the DNS or other configuration information still indicates that this is the valid service instance for a particular service, then clients MAY attempt to reconnect. In reality, if a client is rebooted or otherwise lose state, it may well attempt to reconnect before 49.7 days elapses, for as long as the DNS or other configuration information continues to indicate that this is the service instance the client should use.

6.6.3.1. Reconnecting After a Forcible Abort

If a connection was forcibly aborted by the client, the client SHOULD mark that service instance as not supporting DSO. The client MAY reconnect but not attempt to use DSO, or may connect to a different service instance, if applicable.

6.6.3.2. Reconnecting After an Unexplained Connection Drop

It is also possible for a server to forcibly terminate the connection; in this case the client doesn't know whether the termination was the result of a protocol error or a network outage. When the client notices that the connection has been dropped, it can attempt to reconnect immediately. However, if the connection is dropped again without the client being able to successfully do whatever it is trying to do, it should mark the server as not supporting DSO.

6.6.3.3. Probing for Working DSO Support

Once a server has been marked by the client as not supporting DSO, the client SHOULD NOT attempt DSO operations on that server until some time has elapsed. A reasonable minimum would be an hour. Since forcibly aborted connections are the result of a software failure, it's not likely that the problem will be solved in the first hour after it's first encountered. However, by restricting the retry interval to an hour, the client will be able to notice when the problem has been fixed without placing an undue burden on the server.

7. Base TLVs for DNS Stateful Operations

This section describes the three base TLVs for DNS Stateful Operations: Keepalive, Retry Delay, and Encryption Padding.

7.1. Keepalive TLV

The Keepalive TLV (DSO-TYPE=1) performs two functions. Primarily it establishes the values for the Session Timeouts. Incidentally, it also resets the keepalive timer for the DSO Session, meaning that it can be used as a kind of "no-op" message for the purpose of keeping a session alive. The client will request the desired session timeout values and the server will acknowledge with the response values that it requires the client to use.

DSO messages with the Keepalive TLV as the primary TLV may appear in early data.

The DSO-DATA for the Keepalive TLV is as follows:

```

      1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 3 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+-----+-----+-----+
|                                     |
|      INACTIVITY TIMEOUT (32 bits)  |
|                                     |
+-----+-----+-----+-----+-----+-----+
|                                     |
|      KEEPALIVE INTERVAL (32 bits)  |
|                                     |
+-----+-----+-----+-----+-----+-----+

```

INACTIVITY TIMEOUT: The inactivity timeout for the current DSO Session, specified as a 32-bit unsigned integer, in network (big endian) byte order, in units of milliseconds. This is the timeout at which the client **MUST** begin closing an inactive DSO Session. The inactivity timeout can be any value of the server's choosing. If the client does not gracefully close an inactive DSO Session, then after twice this interval, or five seconds, whichever is greater, the server will forcibly abort the connection.

KEEPALIVE INTERVAL: The keepalive interval for the current DSO Session, specified as a 32-bit unsigned integer, in network (big endian) byte order, in units of milliseconds. This is the interval at which a client **MUST** generate DSO keepalive traffic to maintain connection state. The keepalive interval **MUST NOT** be less than ten seconds. If the client does not generate the mandated DSO keepalive traffic, then after twice this interval the server will forcibly abort the connection. Since the minimum allowed keepalive interval is ten seconds, the minimum time at which a server will forcibly disconnect a client for failing to generate the mandated DSO keepalive traffic is twenty seconds.

The transmission or reception of DSO Keepalive messages (i.e., messages where the Keepalive TLV is the first TLV) reset only the keepalive timer, not the inactivity timer. The reason for this is that periodic DSO Keepalive messages are sent for the sole purpose of keeping a DSO Session alive, when that DSO Session has current or recent non-maintenance activity that warrants keeping that DSO Session alive. Sending DSO keepalive traffic itself is not considered a client activity; it is considered a maintenance activity that is performed in service of other client activities. If DSO keepalive traffic itself were to reset the inactivity timer, then that would create a circular livelock where keepalive traffic would be sent indefinitely to keep a DSO Session alive, where the only activity on that DSO Session would be the keepalive traffic keeping the DSO Session alive so that further keepalive traffic can be sent. For a DSO Session to be considered active, it must be carrying something more than just keepalive traffic. This is why merely sending or receiving a DSO Keepalive message does not reset the inactivity timer.

When sent by a client, the DSO Keepalive request message MUST be sent as an DSO request message, with a nonzero MESSAGE ID. If a server receives a DSO Keepalive message with a zero MESSAGE ID then this is a fatal error and the server MUST forcibly abort the connection immediately. The DSO Keepalive request message resets a DSO Session's keepalive timer, and at the same time communicates to the server the client's requested Session Timeout values. In a server response to a client-initiated DSO Keepalive request message, the Session Timeouts contain the server's chosen values from this point forward in the DSO Session, which the client MUST respect. This is modeled after the DHCP protocol, where the client requests a certain lease lifetime using DHCP option 51 [RFC2132], but the server is the ultimate authority for deciding what lease lifetime is actually granted.

When a client is sending its second and subsequent DSO Keepalive request messages to the server, the client SHOULD continue to request its preferred values each time. This allows flexibility, so that if conditions change during the lifetime of a DSO Session, the server can adapt its responses to better fit the client's needs.

Once a DSO Session is in progress (Section 5.1) a DSO Keepalive message MAY be initiated by a server. When sent by a server, the DSO Keepalive message MUST be sent as a DSO unidirectional message, with the MESSAGE ID set to zero. The client MUST NOT generate a response to a server-initiated DSO Keepalive message. If a client receives a DSO Keepalive request message with a nonzero MESSAGE ID then this is a fatal error and the client MUST forcibly abort the connection immediately. The DSO Keepalive unidirectional message from the

server resets a DSO Session's keepalive timer, and at the same time unilaterally informs the client of the new Session Timeout values to use from this point forward in this DSO Session. No client DSO response to this unilateral declaration is required or allowed.

In DSO Keepalive response messages, the Keepalive TLV is REQUIRED and is used only as a Response Primary TLV sent as a reply to a DSO Keepalive request message from the client. A Keepalive TLV MUST NOT be added to other responses as a Response Additional TLV. If the server wishes to update a client's Session Timeout values other than in response to a DSO Keepalive request message from the client, then it does so by sending an DSO Keepalive unidirectional message of its own, as described above.

It is not required that the Keepalive TLV be used in every DSO Session. While many DNS Stateful operations will be used in conjunction with a long-lived session state, not all DNS Stateful operations require long-lived session state, and in some cases the default 15-second value for both the inactivity timeout and keepalive interval may be perfectly appropriate. However, note that for clients that implement only the DSO-TYPEs defined in this document, a DSO Keepalive request message is the only way for a client to initiate a DSO Session.

7.1.1. Client handling of received Session Timeout values

When a client receives a response to its client-initiated DSO Keepalive message, or receives a server-initiated DSO Keepalive message, the client has then received Session Timeout values dictated by the server. The two timeout values contained in the Keepalive TLV from the server may each be higher, lower, or the same as the respective Session Timeout values the client previously had for this DSO Session.

In the case of the keepalive timer, the handling of the received value is straightforward. The act of receiving the message containing the DSO Keepalive TLV itself resets the keepalive timer, and updates the keepalive interval for the DSO Session. The new keepalive interval indicates the maximum time that may elapse before another message must be sent or received on this DSO Session, if the DSO Session is to remain alive.

In the case of the inactivity timeout, the handling of the received value is a little more subtle, though the meaning of the inactivity timeout remains as specified -- it still indicates the maximum permissible time allowed without useful activity on a DSO Session. The act of receiving the message containing the Keepalive TLV does not itself reset the inactivity timer. The time elapsed since the

last useful activity on this DSO Session is unaffected by exchange of DSO Keepalive messages. The new inactivity timeout value in the Keepalive TLV in the received message does update the timeout associated with the running inactivity timer; that becomes the new maximum permissible time without activity on a DSO Session.

- o If the current inactivity timer value is less than the new inactivity timeout, then the DSO Session may remain open for now. When the inactivity timer value reaches the new inactivity timeout, the client **MUST** then begin closing the DSO Session, as described above.
- o If the current inactivity timer value is equal to the new inactivity timeout, then this DSO Session has been inactive for exactly as long as the server will permit, and now the client **MUST** immediately begin closing this DSO Session.
- o If the current inactivity timer value is already greater than the new inactivity timeout, then this DSO Session has already been inactive for longer than the server permits, and the client **MUST** immediately begin closing this DSO Session.
- o If the current inactivity timer value is already more than twice the new inactivity timeout, then the client is immediately considered delinquent (this DSO Session is immediately eligible to be forcibly terminated by the server) and the client **MUST** immediately begin closing this DSO Session. However if a server abruptly reduces the inactivity timeout in this way, then, to give the client time to close the connection gracefully before the server resorts to forcibly aborting it, the server **SHOULD** give the client an additional grace period of one quarter of the new inactivity timeout, or five seconds, whichever is greater.

7.1.2. Relationship to edns-tcp-keepalive EDNS0 Option

The inactivity timeout value in the Keepalive TLV (DSO-TYPE=1) has similar intent to the edns-tcp-keepalive EDNS0 Option [RFC7828]. A client/server pair that supports DSO **MUST NOT** use the edns-tcp-keepalive EDNS0 Option within any message after a DSO Session has been established. A client that has sent a DSO message to establish a session **MUST NOT** send an edns-tcp-keepalive EDNS0 Option from this point on. Once a DSO Session has been established, if either client or server receives a DNS message over the DSO Session that contains an edns-tcp-keepalive EDNS0 Option, this is a fatal error and the receiver of the edns-tcp-keepalive EDNS0 Option **MUST** forcibly abort the connection immediately.

7.2. Retry Delay TLV

The Retry Delay TLV (DSO-TYPE=2) can be used as a Primary TLV (unidirectional) in a server-to-client message, or as a Response Additional TLV in either direction. DSO messages with a Relay Delay TLV as their primary TLV are not permitted in early data.

The DSO-DATA for the Retry Delay TLV is as follows:

```

          1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 3 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+-----+-----+-----+-----+-----+
|                                RETRY DELAY (32 bits)                                |
+-----+-----+-----+-----+-----+-----+-----+-----+

```

RETRY DELAY: A time value, specified as a 32-bit unsigned integer, in network (big endian) byte order, in units of milliseconds, within which the initiator MUST NOT retry this operation, or retry connecting to this server. Recommendations for the RETRY DELAY value are given in Section 6.6.1.

7.2.1. Retry Delay TLV used as a Primary TLV

When sent from server to client, the Retry Delay TLV is used as the Primary TLV in a DSO unidirectional message. It is used by a server to instruct a client to close the DSO Session and underlying connection, and not to reconnect for the indicated time interval.

In this case it applies to the DSO Session as a whole, and the client MUST begin closing the DSO Session, as described in Section 6.6.1. The RCODE in the message header SHOULD indicate the principal reason for the termination:

- o NOERROR indicates a routine shutdown or restart.
- o FORMERR indicates that a client request was too badly malformed for the session to continue.
- o SERVFAIL indicates that the server is overloaded due to resource exhaustion and needs to shed load.
- o REFUSED indicates that the server has been reconfigured, and at this time it is now unable to perform one or more of the long-lived client operations that were previously being performed on this DSO Session.
- o NOTAUTH indicates that the server has been reconfigured and at this time it is now unable to perform one or more of the long-

lived client operations that were previously being performed on this DSO Session because it does not have authority over the names in question (for example, a DNS Push Notification server could be reconfigured such that it is no longer accepting DNS Push Notification requests for one or more of the currently subscribed names).

This document specifies only these RCODE values for the Retry Delay message. Servers sending Retry Delay messages SHOULD use one of these values. However, future circumstances may create situations where other RCODE values are appropriate in Retry Delay messages, so clients MUST be prepared to accept Retry Delay messages with any RCODE value.

In some cases, when a server sends a Retry Delay message to a client, there may be more than one reason for the server wanting to end the session. Possibly the configuration could have been changed such that some long-lived client operations can no longer be continued due to policy (REFUSED), and other long-lived client operations can no longer be performed due to the server no longer being authoritative for those names (NOTAUTH). In such cases the server MAY use any of the applicable RCODE values, or RCODE=NOERROR (routine shutdown or restart).

Note that the selection of RCODE value in a Retry Delay message is not critical, since the RCODE value is generally used only for information purposes, such as writing to a log file for future human analysis regarding the nature of the disconnection. Generally clients do not modify their behavior depending on the RCODE value. The RETRY DELAY in the message tells the client how long it should wait before attempting a new connection to this service instance.

For clients that do in some way modify their behavior depending on the RCODE value, they should treat unknown RCODE values the same as RCODE=NOERROR (routine shutdown or restart).

A Retry Delay message from server to client is a DSO unidirectional message; the MESSAGE ID MUST be set to zero in the outgoing message and the client MUST NOT send a response.

A client MUST NOT send a Retry Delay DSO message to a server. If a server receives a DSO message where the Primary TLV is the Retry Delay TLV, this is a fatal error and the server MUST forcibly abort the connection immediately.

7.2.2. Retry Delay TLV used as a Response Additional TLV

In the case of a DSO request message that results in a nonzero RCODE value, the responder MAY append a Retry Delay TLV to the response, indicating the time interval during which the initiator SHOULD NOT attempt this operation again.

The indicated time interval during which the initiator SHOULD NOT retry applies only to the failed operation, not to the DSO Session as a whole.

7.3. Encryption Padding TLV

The Encryption Padding TLV (DSO-TYPE=3) can only be used as an Additional or Response Additional TLV. It is only applicable when the DSO Transport layer uses encryption such as TLS.

The DSO-DATA for the Padding TLV is optional and is a variable length field containing non-specified values. A DSO-LENGTH of 0 essentially provides for 4 bytes of padding (the minimum amount).

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
PADDING -- VARIABLE NUMBER OF BYTES															

As specified for the EDNS(0) Padding Option [RFC7830] the PADDING bytes SHOULD be set to 0x00. Other values MAY be used, for example, in cases where there is a concern that the padded message could be subject to compression before encryption. PADDING bytes of any value MUST be accepted in the messages received.

The Encryption Padding TLV may be included in either a DSO request message, response, or both. As specified for the EDNS(0) Padding Option [RFC7830] if a DSO request message is received with an Encryption Padding TLV, then the DSO response MUST also include an Encryption Padding TLV.

The length of padding is intentionally not specified in this document and is a function of current best practices with respect to the type and length of data in the preceding TLVs [I-D.ietf-dprive-padding-policy].

8. Summary Highlights

This section summarizes some noteworthy highlights about various aspects of the DSO protocol.

8.1. QR bit and MESSAGE ID

In DSO Request Messages the QR bit is 0 and the MESSAGE ID is nonzero.

In DSO Response Messages the QR bit is 1 and the MESSAGE ID is nonzero.

In DSO Unidirectional Messages the QR bit is 0 and the MESSAGE ID is zero.

The table below illustrates which combinations are legal and how they are interpreted:

	MESSAGE ID zero	MESSAGE ID nonzero
QR=0	DSO unidirectional Message	DSO Request Message
QR=1	Invalid - Fatal Error	DSO Response Message

8.2. TLV Usage

The table below indicates, for each of the three TLVs defined in this document, whether they are valid in each of ten different contexts.

The first five contexts are DSO requests or DSO unidirectional messages from client to server, and the corresponding responses from server back to client:

- o C-P - Primary TLV, sent in DSO Request message, from client to server, with nonzero MESSAGE ID indicating that this request MUST generate response message.
- o C-U - Primary TLV, sent in DSO Unidirectional message, from client to server, with zero MESSAGE ID indicating that this request MUST NOT generate response message.
- o C-A - Additional TLV, optionally added to a DSO request message or DSO unidirectional message from client to server.
- o CRP - Response Primary TLV, included in response message sent back to the client (in response to a client "C-P" request with nonzero MESSAGE ID indicating that a response is required) where the DSO-TYPE of the Response TLV matches the DSO-TYPE of the Primary TLV in the request.
- o CRA - Response Additional TLV, included in response message sent back to the client (in response to a client "C-P" request with nonzero MESSAGE ID indicating that a response is required) where the DSO-TYPE of the Response TLV does not match the DSO-TYPE of the Primary TLV in the request.

The second five contexts are their counterparts in the opposite direction: DSO requests or DSO unidirectional messages from server to client, and the corresponding responses from client back to server.

- o S-P - Primary TLV, sent in DSO Request message, from server to client, with nonzero MESSAGE ID indicating that this request MUST generate response message.
- o S-U - Primary TLV, sent in DSO Unidirectional message, from server to client, with zero MESSAGE ID indicating that this request MUST NOT generate response message.
- o S-A - Additional TLV, optionally added to a DSO request message or DSO unidirectional message from server to client.

- o SRP - Response Primary TLV, included in response message sent back to the server (in response to a server "S-P" request with nonzero MESSAGE ID indicating that a response is required) where the DSO-TYPE of the Response TLV matches the DSO-TYPE of the Primary TLV in the request.
- o SRA - Response Additional TLV, included in response message sent back to the server (in response to a server "S-P" request with nonzero MESSAGE ID indicating that a response is required) where the DSO-TYPE of the Response TLV does not match the DSO-TYPE of the Primary TLV in the request.

	C-P	C-U	C-A	CRP	CRA	S-P	S-U	S-A	SRP	SRA
KeepAlive	X			X			X			
RetryDelay					X		X			X
Padding			X		X			X		X

Note that some of the columns in this table are currently empty. The table provides a template for future TLV definitions to follow. It is recommended that definitions of future TLVs include a similar table summarizing the contexts where the new TLV is valid.

9. Additional Considerations

9.1. Service Instances

We use the term service instance to refer to software running on a host which can receive connections on some set of IP address and port tuples. What makes the software an instance is that regardless of which of these tuples the client uses to connect to it, the client is connected to the same software, running on the same node (but see Section 9.2), and will receive the same answers and the same keying information.

Service instances are identified from the perspective of the client. If the client is configured with IP addresses and port number tuples, it has no way to tell if the service offered at one tuple is the same server that is listening on a different tuple. So in this case, the client treats each such tuple as if it references a separate service instance.

In some cases a client is configured with a hostname and a port number (either implicitly, where the port number is omitted and assumed, or explicitly, as in the case of DNS SRV records). In these cases, the (hostname, port) tuple uniquely identifies the service instance (hostname comparisons are case-insensitive [RFC1034]).

It is possible that two hostnames might point to some common IP addresses; this is a configuration error which the client is not obliged to detect. The effect of this could be that after being told to disconnect, the client might reconnect to the same server because it is represented as a different service instance.

Implementations SHOULD NOT resolve hostnames and then perform matching of IP address(es) in order to evaluate whether two entities should be determined to be the "same service instance".

9.2. Anycast Considerations

When an anycast service is configured on a particular IP address and port, it must be the case that although there is more than one physical server responding on that IP address, each such server can be treated as equivalent. What we mean by "equivalent" here is that both servers can provide the same service and, where appropriate, the same authentication information, such as PKI certificates, when establishing connections.

If a change in network topology causes packets in a particular TCP connection to be sent to an anycast server instance that does not know about the connection, the new server will automatically terminate the connection with a TCP reset, since it will have no record of the connection, and then the client can reconnect or stop using the connection, as appropriate.

If after the connection is re-established, the client's assumption that it is connected to the same service is violated in some way, that would be considered to be incorrect behavior in this context. It is however out of the possible scope for this specification to make specific recommendations in this regard; that would be up to follow-on documents that describe specific uses of DNS stateful operations.

9.3. Connection Sharing

As previously specified for DNS over TCP [RFC7766]:

To mitigate the risk of unintentional server overload, DNS clients **MUST** take care to minimize the number of concurrent TCP connections made to any individual server. It is **RECOMMENDED** that for any given client/server interaction there **SHOULD** be no more than one connection for regular queries, one for zone transfers, and one for each protocol that is being used on top of TCP (for example, if the resolver was using TLS). However, it is noted that certain primary/secondary configurations with many busy zones might need to use more than one TCP connection for zone transfers for operational reasons (for example, to support concurrent transfers of multiple zones).

A single server may support multiple services, including DNS Updates [RFC2136], DNS Push Notifications [I-D.ietf-dnssd-push], and other services, for one or more DNS zones. When a client discovers that the target server for several different operations is the same service instance (see Section 9.1), the client **SHOULD** use a single shared DSO Session for all those operations.

This requirement has two benefits. First, it reduces unnecessary connection load on the DNS server. Second, it avoids paying the TCP slow start penalty when making subsequent connections to the same server.

However, server implementers and operators should be aware that connection sharing may not be possible in all cases. A single host device may be home to multiple independent client software instances that don't coordinate with each other. Similarly, multiple independent client devices behind the same NAT gateway will also typically appear to the DNS server as different source ports on the same client IP address. Because of these constraints, a DNS server **MUST** be prepared to accept multiple connections from different source ports on the same client IP address.

9.4. Operational Considerations for Middlebox

Where an application-layer middlebox (e.g., a DNS proxy, forwarder, or session multiplexer) is in the path, care must be taken to avoid a configuration in which DSO traffic is mis-handled. The simplest way to avoid such problems is to avoid using middleboxes. When this is not possible, middleboxes should be evaluated to make sure that they behave correctly.

Correct behavior for middleboxes consists of one of:

- o The middlebox does not forward DSO messages, and responds to DSO messages with a response code other than NOERROR or DSOTYPENI.
- o The middlebox acts as a DSO server and follows this specification in establishing connections.
- o There is a 1:1 correspondence between incoming and outgoing connections, such that when a connection is established to the middlebox, it is guaranteed that exactly one corresponding connection will be established from the middlebox to some DNS resolver, and all incoming messages will be forwarded without modification or reordering. An example of this would be a NAT forwarder or TCP connection optimizer (e.g. for a high-latency connection such as a geosynchronous satellite link).

Middleboxes that do not meet one of the above criteria are very likely to fail in unexpected and difficult-to-diagnose ways. For example, a DNS load balancer might unbundle DNS messages from the incoming TCP stream and forward each message from the stream to a different DNS server. If such a load balancer is in use, and the DNS servers it points implement DSO and are configured to enable DSO, DSO session establishment will succeed, but no coherent session will exist between the client and the server. If such a load balancer is pointed at a DNS server that does not implement DSO or is configured not to allow DSO, no such problem will exist, but such a configuration risks unexpected failure if new server software is installed which does implement DSO.

It is of course possible to implement a middlebox that properly supports DSO. It is even possible to implement one that implements DSO with long-lived operations. This can be done either by maintaining a 1:1 correspondence between incoming and outgoing connections, as mentioned above, or by terminating incoming sessions at the middlebox, but maintaining state in the middlebox about any long-lived that are requested. Specifying this in detail is beyond the scope of this document.

9.5. TCP Delayed Acknowledgement Considerations

Most modern implementations of the Transmission Control Protocol (TCP) include a feature called "Delayed Acknowledgement" [RFC1122].

Without this feature, TCP can be very wasteful on the network. For illustration, consider a simple example like remote login, using a very simple TCP implementation that lacks delayed acks. When the user types a keystroke, a data packet is sent. When the data packet arrives at the server, the simple TCP implementation sends an immediate acknowledgement. Mere milliseconds later, the server process reads the one byte of keystroke data, and consequently the simple TCP implementation sends an immediate window update. Mere milliseconds later, the server process generates the character echo, and sends this data back in reply. The simple TCP implementation then sends this data packet immediately too. In this case, this simple TCP implementation sends a burst of three packets almost instantaneously (ack, window update, data).

Clearly it would be more efficient if the TCP implementation were to combine the three separate packets into one, and this is what the delayed ack feature enables.

With delayed ack, the TCP implementation waits after receiving a data packet, typically for 200 ms, and then send its ack if (a) more data packet(s) arrive (b) the receiving process generates some reply data, or (c) 200 ms elapses without either of the above occurring.

With delayed ack, remote login becomes much more efficient, generating just one packet instead of three for each character echo.

The logic of delayed ack is that the 200 ms delay cannot do any significant harm. If something at the other end were waiting for something, then the receiving process should generate the reply that the thing at the end is waiting for, and TCP will then immediately send that reply (and the ack and window update). And if the receiving process does not in fact generate any reply for this particular message, then by definition the thing at the other end cannot be waiting for anything, so the 200 ms delay is harmless.

This assumption may be true, unless the sender is using Nagle's algorithm, a similar efficiency feature, created to protect the network from poorly written client software that performs many rapid small writes in succession. Nagle's algorithm allows these small writes to be combined into larger, less wasteful packets.

Unfortunately, Nagle's algorithm and delayed ack, two valuable efficiency features, can interact badly with each other when used together [NagleDA].

DSO request messages elicit responses; DSO unidirectional messages and DSO response messages do not.

For DSO request messages, which do elicit responses, Nagle's algorithm and delayed ack work as intended.

For DSO messages that do not elicit responses, the delayed ack mechanism causes the ack to be delayed by 200 ms. The 200 ms delay on the ack can in turn cause Nagle's algorithm to prevent the sender from sending any more data for 200 ms until the awaited ack arrives. On an enterprise GigE backbone with sub-millisecond round-trip times, a 200 ms delay is enormous in comparison.

When this issues is raised, there are two solutions that are often offered, neither of them ideal:

1. Disable delayed ack. For DSO messages that elicit no response, removing delayed ack avoids the needless 200 ms delay, and sends back an immediate ack, which tells Nagle's algorithm that it should immediately grant the sender permission to send its next packet. Unfortunately, for DSO messages that *do* elicit a response, removing delayed ack removes the efficiency gains of combining acks with data, and the responder will now send two or three packets instead of one.
2. Disable Nagle's algorithm. When acks are delayed by the delayed ack algorithm, removing Nagle's algorithm prevents the sender from being blocked from sending its next small packet immediately. Unfortunately, on a network with a higher round-trip time, removing Nagle's algorithm removes the efficiency gains of combining multiple small packets into fewer larger ones, with the goal of limiting the number of small packets in flight at any one time.

For DSO messages that elicit a response, delayed ack and Nagle's algorithm do the right thing.

The problem here is that with DSO messages that elicit no response, the TCP implementation is stuck waiting, unsure if a response is about to be generated, or whether the TCP implementation should go ahead and send an ack and window update.

The solution is networking APIs that allow the receiver to inform the TCP implementation that a received message has been read, processed,

and no response for this message will be generated. TCP can then stop waiting for a response that will never come, and immediately go ahead and send an ack and window update.

For implementations of DSO, disabling delayed ack is NOT RECOMMENDED, because of the harm this can do to the network.

For implementations of DSO, disabling Nagle's algorithm is NOT RECOMMENDED, because of the harm this can do to the network.

At the time that this document is being prepared for publication, it is known that at least one TCP implementation provides the ability for the recipient of a TCP message to signal that it is not going to send a response, and hence the delayed ack mechanism can stop waiting. Implementations on operating systems where this feature is available SHOULD make use of it.

10. IANA Considerations

10.1. DSO OPCODE Registration

The IANA is requested to record the value [TBA1] (tentatively 6) for the DSO OPCODE in the DNS OPCODE Registry. DSO stands for DNS Stateful Operations.

10.2. DSO RCODE Registration

The IANA is requested to record the value [TBA2] (tentatively 11) for the DSOTYPENI error code in the DNS RCODE Registry. The DSOTYPENI error code ("DSO-TYPE Not Implemented") indicates that the receiver does implement DNS Stateful Operations, but does not implement the specific DSO-TYPE of the primary TLV in the DSO request message.

10.3. DSO Type Code Registry

The IANA is requested to create the 16-bit DSO Type Code Registry, with initial (hexadecimal) values as shown below:

Type	Name	Early Data	Status	Reference
0000	Reserved	NO	Standard	RFC-TBD
0001	KeepAlive	OK	Standard	RFC-TBD
0002	RetryDelay	NO	Standard	RFC-TBD
0003	EncryptionPadding	NA	Standard	RFC-TBD
0004-003F	Unassigned, reserved for DSO session-management TLVs	NO		
0040-F7FF	Unassigned	NO		
F800-FBFF	Experimental/local use	NO		
FC00-FFFF	Reserved for future expansion	NO		

The meanings of the fields are as follows:

Type: the 16-bit DSO type code

Name: the human-readable name of the TLV

Early Data: If OK, this TLV may be sent as early data in a TLS 0-RTT ([RFC8446] Section 2.3) initial handshake. If NA, the TLV may appear as a secondary TLV in a DSO message that is sent as early data.

Status: IETF Document status (or "External" if not documented in an IETF document).

Reference: A stable reference to the document in which this TLV is defined.

DSO Type Code zero is reserved and is not currently intended for allocation.

Registrations of new DSO Type Codes in the "Reserved for DSO session-management" range 0004-003F and the "Reserved for future expansion" range FC00-FFFF require publication of an IETF Standards Action document [RFC8126].

Any document defining a new TLV which lists a value of "OK" in the 0-RTT column must include a threat analysis for the use of the TLV in the case of TLS 0-RTT. See Section 11.1 for details.

Requests to register additional new DSO Type Codes in the "Unassigned" range 0040-F7FF are to be recorded by IANA after Expert Review [RFC8126]. The expert review should validate that the requested type code is specified in a way that conforms to this specification, and that the intended use for the code would not be addressed with an experimental/local assignment.

DSO Type Codes in the "experimental/local" range F800-FBFF may be used as Experimental Use or Private Use values [RFC8126] and may be used freely for development purposes, or for other purposes within a single site. No attempt is made to prevent multiple sites from using the same value in different (and incompatible) ways. There is no need for IANA to review such assignments (since IANA does not record them) and assignments are not generally useful for broad interoperability. It is the responsibility of the sites making use of "experimental/local" values to ensure that no conflicts occur within the intended scope of use.

11. Security Considerations

If this mechanism is to be used with DNS over TLS, then these messages are subject to the same constraints as any other DNS-over-

TLS messages and MUST NOT be sent in the clear before the TLS session is established.

The data field of the "Encryption Padding" TLV could be used as a covert channel.

When designing new DSO TLVs, the potential for data in the TLV to be used as a tracking identifier should be taken into consideration, and should be avoided when not required.

When used without TLS or similar cryptographic protection, a malicious entity maybe able to inject a malicious unidirectional DSO Retry Delay Message into the data stream, specifying an unreasonably large RETRY DELAY, causing a denial-of-service attack against the client.

The establishment of DSO sessions has an impact on the number of open TCP connections on a DNS server. Additional resources may be used on the server as a result. However, because the server can limit the number of DSO sessions established and can also close existing DSO sessions as needed, denial of service or resource exhaustion should not be a concern.

11.1. TLS 0-RTT Considerations

DSO permits zero round-trip operation using TCP Fast Open [RFC7413] with TLS 1.3 [RFC8446] 0-RTT to reduce or eliminate round trips in session establishment. TCP Fast Open is only permitted in combination with TLS 0-RTT. In the rest of this section we refer to TLS 1.3 early data in a TLS 0-RTT initial handshake message, whether or not it is included in a TCP SYN packet with early data using the TCP Fast Open option, as "early data."

A DSO message may or may not be permitted to be sent as early data. The definition for each TLV that can be used as a primary TLV is required to state whether or not that TLV is permitted as early data. Only response-requiring messages are ever permitted as early data, and only clients are permitted to send any DSO message as early data, unless there is an implicit session (see Section 5.1).

For DSO messages that are permitted as early data, a client MAY include one or more such messages as early data without having to wait for a DSO response to the first DSO request message to confirm successful establishment of a DSO session.

However, unless there is an implicit session, a client MUST NOT send DSO unidirectional messages until after a DSO Session has been mutually established.

Similarly, unless there is an implicit session, a server MUST NOT send DSO request messages until it has received a response-requiring DSO request message from a client and transmitted a successful NOERROR response for that request.

Caution must be taken to ensure that DSO messages sent as early data are idempotent, or are otherwise immune to any problems that could be result from the inadvertent replay that can occur with zero round-trip operation.

It would be possible to add a TLV that requires the server to do some significant work, and send that to the server as initial data in a TCP SYN packet. A flood of such packets could be used as a DoS attack on the server. None of the TLVs defined here have this property.

If a new TLV is specified that does have this property, that TLV must be specified as not permitted in 0-RTT messages. This prevents work from being done until a round-trip has occurred from the server to the client to verify that the source address of the packet is reachable.

Documents that define new TLVs must state whether each new TLV may be sent as early data. Such documents must include a threat analysis in the security considerations section for each TLV defined in the document that may be sent as early data. This threat analysis should be done based on the advice given in [RFC8446] Section 2.3, 8 and Appendix E.5.

12. Acknowledgements

Thanks to Stephane Bortzmeyer, Tim Chown, Ralph Droms, Paul Hoffman, Jan Komissar, Edward Lewis, Allison Mankin, Rui Paulo, David Schinazi, Manju Shankar Rao, Bernie Volz and Bob Harold for their helpful contributions to this document.

13. References

13.1. Normative References

- [RFC1034] Mockapetris, P., "Domain names - concepts and facilities", STD 13, RFC 1034, DOI 10.17487/RFC1034, November 1987, <<https://www.rfc-editor.org/info/rfc1034>>.
- [RFC1035] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, DOI 10.17487/RFC1035, November 1987, <<https://www.rfc-editor.org/info/rfc1035>>.

- [RFC1918] Rekhter, Y., Moskowitz, B., Karrenberg, D., de Groot, G., and E. Lear, "Address Allocation for Private Internets", BCP 5, RFC 1918, DOI 10.17487/RFC1918, February 1996, <<https://www.rfc-editor.org/info/rfc1918>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC2136] Vixie, P., Ed., Thomson, S., Rekhter, Y., and J. Bound, "Dynamic Updates in the Domain Name System (DNS UPDATE)", RFC 2136, DOI 10.17487/RFC2136, April 1997, <<https://www.rfc-editor.org/info/rfc2136>>.
- [RFC6891] Damas, J., Graff, M., and P. Vixie, "Extension Mechanisms for DNS (EDNS(0))", STD 75, RFC 6891, DOI 10.17487/RFC6891, April 2013, <<https://www.rfc-editor.org/info/rfc6891>>.
- [RFC7766] Dickinson, J., Dickinson, S., Bellis, R., Mankin, A., and D. Wessels, "DNS Transport over TCP - Implementation Requirements", RFC 7766, DOI 10.17487/RFC7766, March 2016, <<https://www.rfc-editor.org/info/rfc7766>>.
- [RFC7830] Mayrhofer, A., "The EDNS(0) Padding Option", RFC 7830, DOI 10.17487/RFC7830, May 2016, <<https://www.rfc-editor.org/info/rfc7830>>.
- [RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <<https://www.rfc-editor.org/info/rfc8126>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.

13.2. Informative References

- [I-D.ietf-dnsop-no-response-issue]
Andrews, M. and R. Bellis, "A Common Operational Problem in DNS Servers - Failure To Respond.", draft-ietf-dnsop-no-response-issue-12 (work in progress), November 2018.

- [I-D.ietf-dnssd-mdns-relay]
Lemon, T. and S. Cheshire, "Multicast DNS Discovery Relay", draft-ietf-dnssd-mdns-relay-01 (work in progress), July 2018.
- [I-D.ietf-dnssd-push]
Pusateri, T. and S. Cheshire, "DNS Push Notifications", draft-ietf-dnssd-push-16 (work in progress), November 2018.
- [I-D.ietf-doh-dns-over-https]
Hoffman, P. and P. McManus, "DNS Queries over HTTPS (DoH)", draft-ietf-doh-dns-over-https-14 (work in progress), August 2018.
- [I-D.ietf-dprive-padding-policy]
Mayrhofer, A., "Padding Policy for EDNS(0)", draft-ietf-dprive-padding-policy-06 (work in progress), July 2018.
- [NagleDA] Cheshire, S., "TCP Performance problems caused by interaction between Nagle's Algorithm and Delayed ACK", May 2005,
<<http://www.stuartcheshire.org/papers/nagledelayedack/>>.
- [RFC1122] Braden, R., Ed., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, DOI 10.17487/RFC1122, October 1989,
<<https://www.rfc-editor.org/info/rfc1122>>.
- [RFC2132] Alexander, S. and R. Droms, "DHCP Options and BOOTP Vendor Extensions", RFC 2132, DOI 10.17487/RFC2132, March 1997,
<<https://www.rfc-editor.org/info/rfc2132>>.
- [RFC5382] Guha, S., Ed., Biswas, K., Ford, B., Sivakumar, S., and P. Srisuresh, "NAT Behavioral Requirements for TCP", BCP 142, RFC 5382, DOI 10.17487/RFC5382, October 2008,
<<https://www.rfc-editor.org/info/rfc5382>>.
- [RFC6762] Cheshire, S. and M. Krochmal, "Multicast DNS", RFC 6762, DOI 10.17487/RFC6762, February 2013,
<<https://www.rfc-editor.org/info/rfc6762>>.
- [RFC6763] Cheshire, S. and M. Krochmal, "DNS-Based Service Discovery", RFC 6763, DOI 10.17487/RFC6763, February 2013,
<<https://www.rfc-editor.org/info/rfc6763>>.

- [RFC7413] Cheng, Y., Chu, J., Radhakrishnan, S., and A. Jain, "TCP Fast Open", RFC 7413, DOI 10.17487/RFC7413, December 2014, <<https://www.rfc-editor.org/info/rfc7413>>.
- [RFC7828] Wouters, P., Abley, J., Dickinson, S., and R. Bellis, "The edns-tcp-keepalive EDNS0 Option", RFC 7828, DOI 10.17487/RFC7828, April 2016, <<https://www.rfc-editor.org/info/rfc7828>>.
- [RFC7857] Penno, R., Perreault, S., Boucadair, M., Ed., Sivakumar, S., and K. Naito, "Updates to Network Address Translation (NAT) Behavioral Requirements", BCP 127, RFC 7857, DOI 10.17487/RFC7857, April 2016, <<https://www.rfc-editor.org/info/rfc7857>>.
- [RFC7858] Hu, Z., Zhu, L., Heidemann, J., Mankin, A., Wessels, D., and P. Hoffman, "Specification for DNS over Transport Layer Security (TLS)", RFC 7858, DOI 10.17487/RFC7858, May 2016, <<https://www.rfc-editor.org/info/rfc7858>>.
- [RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <<https://www.rfc-editor.org/info/rfc8446>>.

Authors' Addresses

Ray Bellis
Internet Systems Consortium, Inc.
950 Charter Street
Redwood City CA 94063
USA

Phone: +1 (650) 423-1200
Email: ray@isc.org

Stuart Cheshire
Apple Inc.
One Apple Park Way
Cupertino CA 95014
USA

Phone: +1 (408) 996-1010
Email: cheshire@apple.com

John Dickinson
Sinodun Internet Technologies
Magadalen Centre
Oxford Science Park
Oxford OX4 4GA
United Kingdom

Email: jad@sinodun.com

Sara Dickinson
Sinodun Internet Technologies
Magadalen Centre
Oxford Science Park
Oxford OX4 4GA
United Kingdom

Email: sara@sinodun.com

Ted Lemon
Nibbhaya Consulting
P.O. Box 958
Brattleboro VT 05302-0958
USA

Email: mellon@fugue.com

Tom Pusateri
Unaffiliated
Raleigh NC 27608
USA

Phone: +1 (919) 867-1330
Email: pusateri@bangj.com

Network Working Group
Internet-Draft
Obsoletes: 7719 (if approved)
Updates: 2308 (if approved)
Intended status: Best Current Practice
Expires: March 17, 2019

P. Hoffman
ICANN
A. Sullivan
K. Fujiwara
JPRS
September 13, 2018

DNS Terminology
draft-ietf-dnsop-terminology-bis-14

Abstract

The domain name system (DNS) is defined in literally dozens of different RFCs. The terminology used by implementers and developers of DNS protocols, and by operators of DNS systems, has sometimes changed in the decades since the DNS was first defined. This document gives current definitions for many of the terms used in the DNS in a single document.

This document obsoletes RFC 7719 and updates RFC 2308.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on March 17, 2019.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of

publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
2. Names	4
3. DNS Response Codes	9
4. DNS Transactions	10
5. Resource Records	13
6. DNS Servers and Clients	15
7. Zones	21
8. Wildcards	26
9. Registration Model	27
10. General DNSSEC	29
11. DNSSEC States	33
12. Security Considerations	35
13. IANA Considerations	35
14. References	35
14.1. Normative References	35
14.2. Informative References	38
Appendix A. Definitions Updated by this Document	42
Appendix B. Definitions First Defined in this Document	43
Index	45
Acknowledgements	48
Authors' Addresses	49

1. Introduction

The Domain Name System (DNS) is a simple query-response protocol whose messages in both directions have the same format. (Section 2 gives a definition of "public DNS", which is often what people mean when they say "the DNS".) The protocol and message format are defined in [RFC1034] and [RFC1035]. These RFCs defined some terms, but later documents defined others. Some of the terms from [RFC1034] and [RFC1035] now have somewhat different meanings than they did in 1987.

This document collects a wide variety of DNS-related terms. Some of them have been precisely defined in earlier RFCs, some have been loosely defined in earlier RFCs, and some are not defined in any earlier RFC at all.

Most of the definitions here are the consensus definition of the DNS community -- both protocol developers and operators. Some of the definitions differ from earlier RFCs, and those differences are noted. In this document, where the consensus definition is the same as the one in an RFC, that RFC is quoted. Where the consensus definition has changed somewhat, the RFC is mentioned but the new stand-alone definition is given. See Appendix A for a list of the definitions that this document updates.

It is important to note that, during the development of this document, it became clear that some DNS-related terms are interpreted quite differently by different DNS experts. Further, some terms that are defined in early DNS RFCs now have definitions that are generally agreed to, but that are different from the original definitions. Therefore, this document is a substantial revision to [RFC7719].

The terms are organized loosely by topic. Some definitions are for new terms for things that are commonly talked about in the DNS community but that never had terms defined for them.

Other organizations sometimes define DNS-related terms their own way. For example, the WHATWG defines "domain" at <https://url.spec.whatwg.org/>. The Root Server System Advisory Committee (RSSAC) has a good lexicon [RSSAC026].

Note that there is no single consistent definition of "the DNS". It can be considered to be some combination of the following: a commonly used naming scheme for objects on the Internet; a distributed database representing the names and certain properties of these objects; an architecture providing distributed maintenance, resilience, and loose coherency for this database; and a simple query-response protocol (as mentioned below) implementing this architecture. Section 2 defines "global DNS" and "private DNS" as a way to deal with these differing definitions.

Capitalization in DNS terms is often inconsistent among RFCs and various DNS practitioners. The capitalization used in this document is a best guess at current practices, and is not meant to indicate that other capitalization styles are wrong or archaic. In some cases, multiple styles of capitalization are used for the same term due to quoting from different RFCs.

Readers should note that the terms in this document are grouped by topic. Someone who is not already familiar with the DNS can probably not learn about the DNS from scratch by reading this document from front to back. Instead, skipping around may be the only way to get enough context to understand some of the definitions. This document

has an index that might be useful for readers who are attempting to learn the DNS by reading this document.

2. Names

Naming system: A naming system associates names with data. Naming systems have many significant facets that help differentiate them from each other. Some commonly-identified facets include:

- * Composition of names
- * Format of names
- * Administration of names
- * Types of data that can be associated with names
- * Types of metadata for names
- * Protocol for getting data from a name
- * Context for resolving a name

Note that this list is a small subset of facets that people have identified over time for naming systems, and the IETF has yet to agree on a good set of facets that can be used to compare naming systems. For example, other facets might include "protocol to update data in a name", "privacy of names", and "privacy of data associated with names", but those are not as well-defined as the ones listed above. The list here is chosen because it helps describe the DNS and naming systems similar to the DNS.

Domain name: An ordered list of one or more labels.

Note that this is a definition independent of the DNS RFCs, and the definition here also applies to systems other than the DNS. [RFC1034] defines the "domain name space" using mathematical trees and their nodes in graph theory, and this definition has the same practical result as the definition here. Any path of a directed acyclic graph can be represented by a domain name consisting of the labels of its nodes, ordered by decreasing distance from the root(s) (which is the normal convention within the DNS, including this document). A domain name whose last label identifies a root of the graph is fully qualified; other domain names whose labels form a strict prefix of a fully qualified domain name are relative to its first omitted node.

Also note that different IETF and non-IETF documents have used the term "domain name" in many different ways. It is common for earlier documents to use "domain name" to mean "names that match the syntax in [RFC1035]", but possibly with additional rules such as "and are, or will be, resolvable in the global DNS" or "but only using the presentation format".

Label: An ordered list of zero or more octets that makes up a portion of a domain name. Using graph theory, a label identifies one node in a portion of the graph of all possible domain names.

Global DNS: Using the short set of facets listed in "Naming system", the global DNS can be defined as follows. Most of the rules here come from [RFC1034] and [RFC1035], although the term "global DNS" has not been defined before now.

Composition of names -- A name in the global DNS has one or more labels. The length of each label is between 0 and 63 octets inclusive. In a fully-qualified domain name, the last label in the ordered list is 0 octets long; it is the only label whose length may be 0 octets, and it is called the "root" or "root label". A domain name in the global DNS has a maximum total length of 255 octets in the wire format; the root represents one octet for this calculation. (Multicast DNS [RFC6762] allows names up to 255 bytes plus a terminating zero byte based on a different interpretation of RFC 1035 and what is included in the 255 octets.)

Format of names -- Names in the global DNS are domain names. There are three formats: wire format, presentation format, and common display.

The basic wire format for names in the global DNS is a list of labels ordered by decreasing distance from the root, with the root label last. Each label is preceded by a length octet. [RFC1035] also defines a compression scheme that modifies this format.

The presentation format for names in the global DNS is a list of labels ordered by decreasing distance from the root, encoded as ASCII, with a "." character between each label. In presentation format, a fully-qualified domain name includes the root label and the associated separator dot. For example, in presentation format, a fully-qualified domain name with two non-root labels is always shown as "example.tld." instead of "example.tld". [RFC1035] defines a method for showing octets that do not display in ASCII.

The common display format is used in applications and free text. It is the same as the presentation format, but showing the root label and the "." before it is optional and is rarely done. For example, in common display format, a fully-qualified domain name with two non-root labels is usually shown as "example.tld" instead of "example.tld.". Names in the common display format are normally written such that the directionality of the writing system presents labels by decreasing distance from the root (so, in both English and the C programming language the root or TLD label in the ordered list is right-most; but in Arabic it may be left-most, depending on local conventions).

Administration of names -- Administration is specified by delegation (see the definition of "delegation" in Section 7). Policies for administration of the root zone in the global DNS are determined by the names operational community, which convenes itself in the Internet Corporation for Assigned Names and Numbers (ICANN). The names operational community selects the IANA Functions Operator for the global DNS root zone. At the time this document is published, that operator is Public Technical Identifiers (PTI). (See <<https://pti.icann.org/>> for more information about PTI operating the IANA Functions.) The name servers that serve the root zone are provided by independent root operators. Other zones in the global DNS have their own policies for administration.

Types of data that can be associated with names -- A name can have zero or more resource records associated with it. There are numerous types of resource records with unique data structures defined in many different RFCs and in the IANA registry at [IANA_Resource_Registry].

Types of metadata for names -- Any name that is published in the DNS appears as a set of resource records (see the definition of "RRset" in Section 5). Some names do not themselves have data associated with them in the DNS, but "appear" in the DNS anyway because they form part of a longer name that does have data associated with it (see the definition of "empty non-terminals" in Section 7).

Protocol for getting data from a name -- The protocol described in [RFC1035].

Context for resolving a name -- The global DNS root zone distributed by PTI.

Private DNS: Names that use the protocol described in [RFC1035] but that do not rely on the global DNS root zone, or names that are

otherwise not generally available on the Internet but are using the protocol described in [RFC1035]. A system can use both the global DNS and one or more private DNS systems; for example, see "Split DNS" in Section 6.

Note that domain names that do not appear in the DNS, and that are intended never to be looked up using the DNS protocol, are not part of the global DNS or a private DNS even though they are domain names.

Multicast DNS: "Multicast DNS (mDNS) provides the ability to perform DNS-like operations on the local link in the absence of any conventional Unicast DNS server. In addition, Multicast DNS designates a portion of the DNS namespace to be free for local use, without the need to pay any annual fee, and without the need to set up delegations or otherwise configure a conventional DNS server to answer for those names." (Quoted from [RFC6762], Abstract) Although it uses a compatible wire format, mDNS is strictly speaking a different protocol than DNS. Also, where the above quote says "a portion of the DNS namespace", it would be clearer to say "a portion of the domain name space" The names in mDNS are not intended to be looked up in the DNS.

Locally served DNS zone: A locally served DNS zone is a special case of private DNS. Names are resolved using the DNS protocol in a local context. [RFC6303] defines subdomains of IN-ADDR.ARPA that are locally served zones. Resolution of names through locally served zones may result in ambiguous results. For example, the same name may resolve to different results in different locally served DNS zone contexts. The context for a locally served DNS zone may be explicit, for example, as defined in [RFC6303], or implicit, as defined by local DNS administration and not known to the resolution client.

Fully qualified domain name (FQDN): This is often just a clear way of saying the same thing as "domain name of a node", as outlined above. However, the term is ambiguous. Strictly speaking, a fully qualified domain name would include every label, including the zero-length label of the root: such a name would be written "www.example.net." (note the terminating dot). But because every name eventually shares the common root, names are often written relative to the root (such as "www.example.net") and are still called "fully qualified". This term first appeared in [RFC0819]. In this document, names are often written relative to the root.

The need for the term "fully qualified domain name" comes from the existence of partially qualified domain names, which are names where one or more of the last labels in the ordered list are

omitted (for example, a domain name of "www" relative to "example.net" identifies "www.example.net"). Such relative names are understood only by context.

Host name: This term and its equivalent, "hostname", have been widely used but are not defined in [RFC1034], [RFC1035], [RFC1123], or [RFC2181]. The DNS was originally deployed into the Host Tables environment as outlined in [RFC0952], and it is likely that the term followed informally from the definition there. Over time, the definition seems to have shifted. "Host name" is often meant to be a domain name that follows the rules in Section 3.5 of [RFC1034], the "preferred name syntax" (that is, every character in each label is a letter, a digit, or a hyphen). Note that any label in a domain name can contain any octet value; hostnames are generally considered to be domain names where every label follows the rules in the "preferred name syntax", with the amendment that labels can start with ASCII digits (this amendment comes from Section 2.1 of [RFC1123]).

People also sometimes use the term hostname to refer to just the first label of an FQDN, such as "printer" in "printer.admin.example.com". (Sometimes this is formalized in configuration in operating systems.) In addition, people sometimes use this term to describe any name that refers to a machine, and those might include labels that do not conform to the "preferred name syntax".

TLD: A Top-Level Domain, meaning a zone that is one layer below the root, such as "com" or "jp". There is nothing special, from the point of view of the DNS, about TLDs. Most of them are also delegation-centric zones (defined in Section 7, and there are significant policy issues around their operation. TLDs are often divided into sub-groups such as Country Code Top-Level Domains (ccTLDs), Generic Top-Level Domains (gTLDs), and others; the division is a matter of policy, and beyond the scope of this document.

IDN: The common abbreviation for "Internationalized Domain Name". The IDNA protocol is the standard mechanism for handling domain names with non-ASCII characters in applications in the DNS. The current standard at the time of this writing, normally called "IDNA2008", is defined in [RFC5890], [RFC5891], [RFC5892], [RFC5893], and [RFC5894]. These documents define many IDN-specific terms such as "LDH label", "A-label", and "U-label". [RFC6365] defines more terms that relate to internationalization (some of which relate to IDNs), and [RFC6055] has a much more extensive discussion of IDNs, including some new terminology.

Subdomain: "A domain is a subdomain of another domain if it is contained within that domain. This relationship can be tested by seeing if the subdomain's name ends with the containing domain's name." (Quoted from [RFC1034], Section 3.1). For example, in the host name "nnn.mmm.example.com", both "mmm.example.com" and "nnn.mmm.example.com" are subdomains of "example.com". Note that the comparisons here are done on whole labels; that is, "ooo.example.com" is not a subdomain of "oo.example.com".

Alias: The owner of a CNAME resource record, or a subdomain of the owner of a DNAME resource record (DNAME records are defined in [RFC6672]). See also "canonical name".

Canonical name: A CNAME resource record "identifies its owner name as an alias, and specifies the corresponding canonical name in the RDATA section of the RR." (Quoted from [RFC1034], Section 3.6.2) This usage of the word "canonical" is related to the mathematical concept of "canonical form".

CNAME: "It is traditional to refer to the owner of a CNAME record as 'a CNAME'. This is unfortunate, as 'CNAME' is an abbreviation of 'canonical name', and the owner of a CNAME record is an alias, not a canonical name." (Quoted from [RFC2181], Section 10.1.1)

3. DNS Response Codes

Some of response codes that are defined in [RFC1035] have acquired their own shorthand names. All of the RCODEs are listed at [IANA_Resource_Registry], although that site uses mixed-case capitalization, while most documents use all-caps. Some of the common names are described here, but the official list is in the IANA registry.

NOERROR: "No error condition" (Quoted from [RFC1035], Section 4.1.1.)

FORMERR: "Format error - The name server was unable to interpret the query." (Quoted from [RFC1035], Section 4.1.1.)

SERVFAIL: "Server failure - The name server was unable to process this query due to a problem with the name server." (Quoted from [RFC1035], Section 4.1.1.)

NXDOMAIN: "Name Error - This code signifies that the domain name referenced in the query does not exist." (Quoted from [RFC1035], Section 4.1.1.) [RFC2308] established NXDOMAIN as a synonym for Name Error.

NOTIMP: "Not Implemented - The name server does not support the requested kind of query." (Quoted from [RFC1035], Section 4.1.1.)

REFUSED: "Refused - The name server refuses to perform the specified operation for policy reasons. For example, a name server may not wish to provide the information to the particular requester, or a name server may not wish to perform a particular operation (e.g., zone transfer) for particular data." (Quoted from [RFC1035], Section 4.1.1.)

NODATA: "A pseudo RCODE which indicates that the name is valid for the given class, but there are no records of the given type. A NODATA response has to be inferred from the answer." (Quoted from [RFC2308], Section 1.) "NODATA is indicated by an answer with the RCODE set to NOERROR and no relevant answers in the answer section. The authority section will contain an SOA record, or there will be no NS records there." (Quoted from [RFC2308], Section 2.2.) Note that referrals have a similar format to NODATA replies; [RFC2308] explains how to distinguish them.

The term "NXRRSET" is sometimes used as a synonym for NODATA. However, this is a mistake, given that NXRRSET is a specific error code defined in [RFC2136].

Negative response: A response that indicates that a particular RRset does not exist, or whose RCODE indicates the nameserver cannot answer. Sections 2 and 7 of [RFC2308] describe the types of negative responses in detail.

4. DNS Transactions

The header of a DNS message is its first 12 octets. Many of the fields and flags in the header diagram in Sections 4.1.1 through 4.1.3 of [RFC1035] are referred to by their names in that diagram. For example, the response codes are called "RCODEs", the data for a record is called the "RDATA", and the authoritative answer bit is often called "the AA flag" or "the AA bit".

Class: A class "identifies a protocol family or instance of a protocol" (Quoted from [RFC1034], Section 3.6). "The DNS tags all data with a class as well as the type, so that we can allow parallel use of different formats for data of type address." (Quoted from [RFC1034], Section 2.2). In practice, the class for nearly every query is "IN". There are some queries for "CH", but they are usually for the purposes of information about the server itself rather than for a different type of address.

QNAME: The most commonly-used rough definition is that the QNAME is a field in the Question section of a query. "A standard query specifies a target domain name (QNAME), query type (QTYPE), and query class (QCLASS) and asks for RRs which match." (Quoted from [RFC1034], Section 3.7.1.). Strictly speaking, the definition comes from [RFC1035], Section 4.1.2, where the QNAME is defined in respect of the Question Section. This definition appears to be applied consistently: the discussion of inverse queries in section 6.4 refers to the "owner name of the query RR and its TTL", because inverse queries populate the Answer Section and leave the Question Section empty. (Inverse queries are deprecated in [RFC3425], and so relevant definitions do not appear in this document.)

[RFC2308], however, has an alternate definition that puts the QNAME in the answer (or series of answers) instead of the query. It defines QNAME as: "...the name in the query section of an answer, or where this resolves to a CNAME, or CNAME chain, the data field of the last CNAME. The last CNAME in this sense is that which contains a value which does not resolve to another CNAME." This definition has a certain internal logic, because of the way CNAME substitution works and the definition of CNAME. If a name server does not find an RRset that matches a query, but it finds the same name in the same class with a CNAME record, then the name server "includes the CNAME record in the response and restarts the query at the domain name specified in the data field of the CNAME record." (Quoted from [RFC1034] Section 3.6.2). This is made explicit in the resolution algorithm outlined in Section 4.3.2 of [RFC1034], which says to "change QNAME to the canonical name in the CNAME RR, and go back to step 1" in the case of a CNAME RR. Since a CNAME record explicitly declares that the owner name is canonically named what is in the RDATA, then there is a way to view the new name (i.e. the name that was in the RDATA of the CNAME RR) as also being the QNAME.

This creates a kind of confusion, however, because the response to a query that results in CNAME processing contains in the echoed Question Section one QNAME (the name in the original query), and a second QNAME that is in the data field of the last CNAME. The confusion comes from the iterative/recursive mode of resolution, which finally returns an answer that need not actually have the same owner name as the QNAME contained in the original query.

To address this potential confusion, it is helpful to distinguish between three meanings:

- * QNAME (original): The name actually sent in the Question Section in the original query, which is always echoed in the

(final) reply in the Question Section when the QR bit is set to 1.

- * QNAME (effective): A name actually resolved, which is either the name originally queried, or a name received in a CNAME chain response.
- * QNAME (final): The name actually resolved, which is either the name actually queried or else the last name in a CNAME chain response.

Note that, because the definition in [RFC2308] is actually for a different concept than what was in [RFC1034], it would have been better if [RFC2308] had used a different name for that concept. In general use today, QNAME almost always means what is defined above as "QNAME (original)".

Referrals: A type of response in which a server, signaling that it is not (completely) authoritative for an answer, provides the querying resolver with an alternative place to send its query. Referrals can be partial.

A referral arises when a server is not performing recursive service while answering a query. It appears in step 3(b) of the algorithm in [RFC1034], Section 4.3.2.

There are two types of referral response. The first is a downward referral (sometimes described as "delegation response"), where the server is authoritative for some portion of the QNAME. The authority section RRset's RDATA contains the name servers specified at the referred-to zone cut. In normal DNS operation, this kind of response is required in order to find names beneath a delegation. The bare use of "referral" means this kind of referral, and many people believe that this is the only legitimate kind of referral in the DNS.

The second is an upward referral (sometimes described as "root referral"), where the server is not authoritative for any portion of the QNAME. When this happens, the referred-to zone in the authority section is usually the root zone (.). In normal DNS operation, this kind of response is not required for resolution or for correctly answering any query. There is no requirement that any server send upward referrals. Some people regard upward referrals as a sign of a misconfiguration or error. Upward referrals always need some sort of qualifier (such as "upward" or "root"), and are never identified by the bare word "referral".

A response that has only a referral contains an empty answer section. It contains the NS RRset for the referred-to zone in the authority section. It may contain RRs that provide addresses in the additional section. The AA bit is clear.

In the case where the query matches an alias, and the server is not authoritative for the target of the alias but it is authoritative for some name above the target of the alias, the resolution algorithm will produce a response that contains both the authoritative answer for the alias, and also a referral. Such a partial answer and referral response has data in the answer section. It has the NS RRset for the referred-to zone in the authority section. It may contain RRs that provide addresses in the additional section. The AA bit is set, because the first name in the answer section matches the QNAME and the server is authoritative for that answer (see [RFC1035], Section 4.1.1).

5. Resource Records

RR: An acronym for resource record. ([RFC1034], Section 3.6.)

RRset: A set of resource records "with the same label, class and type, but with different data". (Definition from [RFC2181], Section 5) Also spelled RRSet in some documents. As a clarification, "same label" in this definition means "same owner name". In addition, [RFC2181] states that "the TTLs of all RRs in an RRSet must be the same".

Note that RRSIG resource records do not match this definition. [RFC4035] says: "An RRset MAY have multiple RRSIG RRs associated with it. Note that as RRSIG RRs are closely tied to the RRsets whose signatures they contain, RRSIG RRs, unlike all other DNS RR types, do not form RRsets. In particular, the TTL values among RRSIG RRs with a common owner name do not follow the RRset rules described in [RFC2181]."

Master file: "Master files are text files that contain RRs in text form. Since the contents of a zone can be expressed in the form of a list of RRs a master file is most often used to define a zone, though it can be used to list a cache's contents." (Quoted from [RFC1035], Section 5.) Master files are sometimes called "zone files".

Presentation format: The text format used in master files. This format is shown but not formally defined in [RFC1034] and [RFC1035]. The term "presentation format" first appears in [RFC4034].

EDNS: The extension mechanisms for DNS, defined in [RFC6891]. Sometimes called "EDNS0" or "EDNS(0)" to indicate the version number. EDNS allows DNS clients and servers to specify message sizes larger than the original 512 octet limit, to expand the response code space, and to carry additional options that affect the handling of a DNS query.

OPT: A pseudo-RR (sometimes called a "meta-RR") that is used only to contain control information pertaining to the question-and-answer sequence of a specific transaction. (Definition from [RFC6891], Section 6.1.1) It is used by EDNS.

Owner: "The domain name where a RR is found" (Quoted from [RFC1034], Section 3.6). Often appears in the term "owner name".

SOA field names: DNS documents, including the definitions here, often refer to the fields in the RDATA of an SOA resource record by field name. "SOA" stands for "start of a zone of authority". Those fields are defined in Section 3.3.13 of [RFC1035]. The names (in the order they appear in the SOA RDATA) are MNAME, RNAME, SERIAL, REFRESH, RETRY, EXPIRE, and MINIMUM. Note that the meaning of MINIMUM field is updated in Section 4 of [RFC2308]; the new definition is that the MINIMUM field is only "the TTL to be used for negative responses". This document tends to use field names instead of terms that describe the fields.

TTL: The maximum "time to live" of a resource record. "A TTL value is an unsigned number, with a minimum value of 0, and a maximum value of 2147483647. That is, a maximum of $2^{31} - 1$. When transmitted, the TTL is encoded in the less significant 31 bits of the 32 bit TTL field, with the most significant, or sign, bit set to zero." (Quoted from [RFC2181], Section 8) (Note that [RFC1035] erroneously stated that this is a signed integer; that was fixed by [RFC2181].)

The TTL "specifies the time interval that the resource record may be cached before the source of the information should again be consulted". (Quoted from [RFC1035], Section 3.2.1) Also: "the time interval (in seconds) that the resource record may be cached before it should be discarded". (Quoted from [RFC1035], Section 4.1.3). Despite being defined for a resource record, the TTL of every resource record in an RRset is required to be the same ([RFC2181], Section 5.2).

The reason that the TTL is the maximum time to live is that a cache operator might decide to shorten the time to live for operational purposes, such as if there is a policy to disallow TTL values over a certain number. Some servers are known to ignore

the TTL on some RRsets (such as when the authoritative data has a very short TTL) even though this is against the advice in RFC 1035. An RRset can be flushed from the cache before the end of the TTL interval, at which point the value of the TTL becomes unknown because the RRset with which it was associated no longer exists.

There is also the concept of a "default TTL" for a zone, which can be a configuration parameter in the server software. This is often expressed by a default for the entire server, and a default for a zone using the \$TTL directive in a zone file. The \$TTL directive was added to the master file format by [RFC2308].

Class independent: A resource record type whose syntax and semantics are the same for every DNS class. A resource record type that is not class independent has different meanings depending on the DNS class of the record, or the meaning is undefined for some class. Most resource record types are defined for class 1 (IN, the Internet), but many are undefined for other classes.

Address records: Records whose type is A or AAAA. [RFC2181] informally defines these as "(A, AAAA, etc)". Note that new types of address records could be defined in the future.

6. DNS Servers and Clients

This section defines the terms used for the systems that act as DNS clients, DNS servers, or both. In the RFCs, DNS servers are sometimes called "name servers", "nameservers", or just "servers". There is no formal definition of DNS server, but the RFCs generally assume that it is an Internet server that listens for queries and sends responses using the DNS protocol defined in [RFC1035] and its successors.

It is important to note that the terms "DNS server" and "name server" require context in order to understand the services being provided. Both authoritative servers and recursive resolvers are often called "DNS servers" and "name servers" even though they serve different roles (but may be part of the same software package).

For terminology specific to the public DNS root server system, see [RSSAC026]. That document defines terms such as "root server", "root server operator", and terms that are specific to the way that the root zone of the public DNS is served.

Resolver: A program "that extract[s] information from name servers in response to client requests." (Quoted from [RFC1034], Section 2.4) A resolver performs queries for a name, type, and

class, and receives responses. The logical function is called "resolution". In practice, the term is usually referring to some specific type of resolver (some of which are defined below), and understanding the use of the term depends on understanding the context.

A related term is "resolve", which is not formally defined in [RFC1034] or [RFC1035]. An imputed definition might be "asking a question that consists of a domain name, class, and type, and receiving some sort of response". Similarly, an imputed definition of "resolution" might be "the response received from resolving".

Stub resolver: A resolver that cannot perform all resolution itself. Stub resolvers generally depend on a recursive resolver to undertake the actual resolution function. Stub resolvers are discussed but never fully defined in Section 5.3.1 of [RFC1034]. They are fully defined in Section 6.1.3.1 of [RFC1123].

Iterative mode: A resolution mode of a server that receives DNS queries and responds with a referral to another server. Section 2.3 of [RFC1034] describes this as "The server refers the client to another server and lets the client pursue the query". A resolver that works in iterative mode is sometimes called an "iterative resolver". See also "iterative resolution" later in this section.

Recursive mode: A resolution mode of a server that receives DNS queries and either responds to those queries from a local cache or sends queries to other servers in order to get the final answers to the original queries. Section 2.3 of [RFC1034] describes this as "The first server pursues the query for the client at another server". Section 4.3.1 of [RFC1034] says "in [recursive] mode the name server acts in the role of a resolver and returns either an error or the answer, but never referrals." That same section also says "The recursive mode occurs when a query with RD set arrives at a server which is willing to provide recursive service; the client can verify that recursive mode was used by checking that both RA and RD are set in the reply."

A server operating in recursive mode may be thought of as having a name server side (which is what answers the query) and a resolver side (which performs the resolution function). Systems operating in this mode are commonly called "recursive servers". Sometimes they are called "recursive resolvers". In practice it is not possible to know in advance whether the server that one is querying will also perform recursion; both terms can be observed in use interchangeably.

Recursive resolver: A resolver that acts in recursive mode. In general, a recursive resolver is expected to cache the answers it receives (which would make it a full-service resolver), but some recursive resolvers might not cache.

[RFC4697] tried to differentiate between a recursive resolver and an iterative resolver.

Recursive query: A query with the Recursion Desired (RD) bit set to 1 in the header. (See Section 4.1.1 of [RFC1035].) If recursive service is available and is requested by the RD bit in the query, the server uses its resolver to answer the query. (See Section 4.3.2 of [RFC1035].)

Non-recursive query: A query with the Recursion Desired (RD) bit set to 0 in the header. A server can answer non-recursive queries using only local information: the response contains either an error, the answer, or a referral to some other server "closer" to the answer. (See Section 4.3.1 of [RFC1035].)

Iterative resolution: A name server may be presented with a query that can only be answered by some other server. The two general approaches to dealing with this problem are "recursive", in which the first server pursues the query on behalf of the client at another server, and "iterative", in which the server refers the client to another server and lets the client pursue the query there. (See Section 2.3 of [RFC1034].)

In iterative resolution, the client repeatedly makes non-recursive queries and follows referrals and/or aliases. The iterative resolution algorithm is described in Section 5.3.3 of [RFC1034].

Full resolver: This term is used in [RFC1035], but it is not defined there. RFC 1123 defines a "full-service resolver" that may or may not be what was intended by "full resolver" in [RFC1035]. This term is not properly defined in any RFC.

Full-service resolver: Section 6.1.3.1 of [RFC1123] defines this term to mean a resolver that acts in recursive mode with a cache (and meets other requirements).

Priming: "The act of finding the list of root servers from a configuration that lists some or all of the purported IP addresses of some or all of those root servers." (Quoted from [RFC8109], Section 2.) In order to operate in recursive mode, a resolver needs to know the address of at least one root server. Priming is most often done from a configuration setting that contains a list of authoritative servers for the root zone.

Root hints: "Operators who manage a DNS recursive resolver typically need to configure a 'root hints file'. This file contains the names and IP addresses of the authoritative name servers for the root zone, so the software can bootstrap the DNS resolution process. For many pieces of software, this list comes built into the software." (Quoted from [IANA_RootFiles]) This file is often used in priming.

Negative caching: "The storage of knowledge that something does not exist, cannot give an answer, or does not give an answer." (Quoted from [RFC2308], Section 1)

Authoritative server: "A server that knows the content of a DNS zone from local knowledge, and thus can answer queries about that zone without needing to query other servers." (Quoted from [RFC2182], Section 2.) An authoritative server is named in the NS ("name server") record in a zone. It is a system that responds to DNS queries with information about zones for which it has been configured to answer with the AA flag in the response header set to 1. It is a server that has authority over one or more DNS zones. Note that it is possible for an authoritative server to respond to a query without the parent zone delegating authority to that server. Authoritative servers also provide "referrals", usually to child zones delegated from them; these referrals have the AA bit set to 0 and come with referral data in the Authority and (if needed) the Additional sections.

Authoritative-only server: A name server that only serves authoritative data and ignores requests for recursion. It will "not normally generate any queries of its own. Instead, it answers non-recursive queries from iterative resolvers looking for information in zones it serves." (Quoted from [RFC4697], Section 2.4) In this case, "ignores requests for recursion" means "responds to requests for recursion with responses indicating that recursion was not performed".

Zone transfer: The act of a client requesting a copy of a zone and an authoritative server sending the needed information. (See Section 7 for a description of zones.) There are two common standard ways to do zone transfers: the AXFR ("Authoritative Transfer") mechanism to copy the full zone (described in [RFC5936], and the IXFR ("Incremental Transfer") mechanism to copy only parts of the zone that have changed (described in [RFC1995]). Many systems use non-standard methods for zone transfer outside the DNS protocol.

Slave server: See "Secondary server".

Secondary server: "An authoritative server which uses zone transfer to retrieve the zone" (Quoted from [RFC1996], Section 2.1).

Secondary servers are also discussed in [RFC1034]. [RFC2182] describes secondary servers in more detail. Although early DNS RFCs such as [RFC1996] referred to this as a "slave", the current common usage has shifted to calling it a "secondary".

Master server: See "Primary server".

Primary server: "Any authoritative server configured to be the source of zone transfer for one or more [secondary] servers" (Quoted from [RFC1996], Section 2.1) or, more specifically, "an authoritative server configured to be the source of AXFR or IXFR data for one or more [secondary] servers" (Quoted from [RFC2136]). Primary servers are also discussed in [RFC1034]. Although early DNS RFCs such as [RFC1996] referred to this as a "master", the current common usage has shifted to "primary".

Primary master: "The primary master is named in the zone's SOA MNAME field and optionally by an NS RR". (Quoted from [RFC1996], Section 2.1). [RFC2136] defines "primary master" as "Master server at the root of the AXFR/IXFR dependency graph. The primary master is named in the zone's SOA MNAME field and optionally by an NS RR. There is by definition only one primary master server per zone."

The idea of a primary master is only used in [RFC1996] and [RFC2136]. A modern interpretation of the term "primary master" is a server that is both authoritative for a zone and that gets its updates to the zone from configuration (such as a master file) or from UPDATE transactions.

Stealth server: This is "like a slave server except not listed in an NS RR for the zone." (Quoted from [RFC1996], Section 2.1)

Hidden master: A stealth server that is a primary server for zone transfers. "In this arrangement, the master name server that processes the updates is unavailable to general hosts on the Internet; it is not listed in the NS RRset." (Quoted from [RFC6781], Section 3.4.3). An earlier RFC, [RFC4641], said that the hidden master's name "appears in the SOA RRs MNAME field", although in some setups, the name does not appear at all in the public DNS. A hidden master can also be a secondary server for the zone itself.

Forwarding: The process of one server sending a DNS query with the RD bit set to 1 to another server to resolve that query.

Forwarding is a function of a DNS resolver; it is different than simply blindly relaying queries.

[RFC5625] does not give a specific definition for forwarding, but describes in detail what features a system that forwards needs to support. Systems that forward are sometimes called "DNS proxies", but that term has not yet been defined (even in [RFC5625]).

Forwarder: Section 1 of [RFC2308] describes a forwarder as "a nameserver used to resolve queries instead of directly using the authoritative nameserver chain". [RFC2308] further says "The forwarder typically either has better access to the internet, or maintains a bigger cache which may be shared amongst many resolvers." That definition appears to suggest that forwarders normally only query authoritative servers. In current use, however, forwarders often stand between stub resolvers and recursive servers. [RFC2308] is silent on whether a forwarder is iterative-only or can be a full-service resolver.

Policy-implementing resolver: A resolver acting in recursive mode that changes some of the answers that it returns based on policy criteria, such as to prevent access to malware sites or objectionable content. In general, a stub resolver has no idea whether upstream resolvers implement such policy or, if they do, the exact policy about what changes will be made. In some cases, the user of the stub resolver has selected the policy-implementing resolver with the explicit intention of using it to implement the policies. In other cases, policies are imposed without the user of the stub resolver being informed.

Open resolver: A full-service resolver that accepts and processes queries from any (or nearly any) client. This is sometimes also called a "public resolver", although the term "public resolver" is used more with open resolvers that are meant to be open, as compared to the vast majority of open resolvers that are probably misconfigured to be open. Open resolvers are discussed in [RFC5358]

Split DNS: The terms "split DNS" and "split-horizon DNS" have long been used in the DNS community without formal definition. In general, they refer to situations in which DNS servers that are authoritative for a particular set of domains provide partly or completely different answers in those domains depending on the source of the query. The effect of this is that a domain name that is notionally globally unique nevertheless has different meanings for different network users. This can sometimes be the result of a "view" configuration, described below.

[RFC2775], Section 3.8 gives a related definition that is too specific to be generally useful.

View: A configuration for a DNS server that allows it to provide different responses depending on attributes of the query, such as for "split DNS". Typically, views differ by the source IP address of a query, but can also be based on the destination IP address, the type of query (such as AXFR), whether it is recursive, and so on. Views are often used to provide more names or different addresses to queries from "inside" a protected network than to those "outside" that network. Views are not a standardized part of the DNS, but they are widely implemented in server software.

Passive DNS: A mechanism to collect DNS data by storing DNS responses from name servers. Some of these systems also collect the DNS queries associated with the responses, although doing so raises some privacy concerns. Passive DNS databases can be used to answer historical questions about DNS zones such as which values were present at a given time in the past, or when a name was spotted first. Passive DNS databases allow searching of the stored records on keys other than just the name and type, such as "find all names which have A records of a particular value".

Anycast: "The practice of making a particular service address available in multiple, discrete, autonomous locations, such that datagrams sent are routed to one of several available locations." (Quoted from [RFC4786], Section 2) See [RFC4786] for more detail on Anycast and other terms that are specific to its use.

Instance: "When anycast routing is used to allow more than one server to have the same IP address, each one of those servers is commonly referred to as an 'instance'." "An instance of a server, such as a root server, is often referred to as an 'Anycast instance'." (Quoted from [RSSAC026])

Privacy-enabling DNS server: "A DNS server that implements DNS over TLS [RFC7858] and may optionally implement DNS over DTLS [RFC8094]." (Quoted from [RFC8310], Section 2) Other types of DNS servers might also be considered privacy-enabling, such as those running DNS over HTTPS [I-D.ietf-doh-dns-over-https].

7. Zones

This section defines terms that are used when discussing zones that are being served or retrieved.

Zone: "Authoritative information is organized into units called 'zones', and these zones can be automatically distributed to the

name servers which provide redundant service for the data in a zone." (Quoted from [RFC1034], Section 2.4)

Child: "The entity on record that has the delegation of the domain from the Parent." (Quoted from [RFC7344], Section 1.1)

Parent: "The domain in which the Child is registered." (Quoted from [RFC7344], Section 1.1) Earlier, "parent name server" was defined in [RFC0882] as "the name server that has authority over the place in the domain name space that will hold the new domain". (Note that [RFC0882] was obsoleted by [RFC1034] and [RFC1035].) [RFC0819] also has some description of the relationship between parents and children.

Origin:

There are two different uses for this term:

(a) "The domain name that appears at the top of a zone (just below the cut that separates the zone from its parent). The name of the zone is the same as the name of the domain at the zone's origin." (Quoted from [RFC2181], Section 6.) These days, this sense of "origin" and "apex" (defined below) are often used interchangeably.

(b) The domain name within which a given relative domain name appears in zone files. Generally seen in the context of "\$ORIGIN", which is a control entry defined in [RFC1035], Section 5.1, as part of the master file format. For example, if the \$ORIGIN is set to "example.org.", then a master file line for "www" is in fact an entry for "www.example.org.".

Apex: The point in the tree at an owner of an SOA and corresponding authoritative NS RRset. This is also called the "zone apex". [RFC4033] defines it as "the name at the child's side of a zone cut". The "apex" can usefully be thought of as a data-theoretic description of a tree structure, and "origin" is the name of the same concept when it is implemented in zone files. The distinction is not always maintained in use, however, and one can find uses that conflict subtly with this definition. [RFC1034] uses the term "top node of the zone" as a synonym of "apex", but that term is not widely used. These days, the first sense of "origin" (above) and "apex" are often used interchangeably.

Zone cut: The delimitation point between two zones where the origin of one of the zones is the child of the other zone.

"Zones are delimited by 'zone cuts'. Each zone cut separates a 'child' zone (below the cut) from a 'parent' zone (above the cut)." (Quoted from [RFC2181], Section 6; note that this is barely an ostensive definition.) Section 4.2 of [RFC1034] uses "cuts" instead of "zone cut".

Delegation: The process by which a separate zone is created in the name space beneath the apex of a given domain. Delegation happens when an NS RRset is added in the parent zone for the child origin. Delegation inherently happens at a zone cut. The term is also commonly a noun: the new zone that is created by the act of delegating.

Authoritative data: "All of the RRs attached to all of the nodes from the top node of the zone down to leaf nodes or nodes above cuts around the bottom edge of the zone." (Quoted from [RFC1034], Section 4.2.1) Note that this definition might inadvertently also cause any NS records that appear in the zone to be included, even those that might not truly be authoritative because there are identical NS RRs below the zone cut. This reveals the ambiguity in the notion of authoritative data, because the parent-side NS records authoritatively indicate the delegation, even though they are not themselves authoritative data.

[RFC4033], Section 2, defines "Authoritative RRset" which is related to authoritative data but has a more precise definition.

Lame delegation: "A lame delegations exists when a nameserver is delegated responsibility for providing nameservice for a zone (via NS records) but is not performing nameservice for that zone (usually because it is not set up as a primary or secondary for the zone)." (Quoted from [RFC1912], Section 2.8)

Another definition is that a lame delegation "happens when a name server is listed in the NS records for some domain and in fact it is not a server for that domain. Queries are thus sent to the wrong servers, who don't know nothing (at least not as expected) about the queried domain. Furthermore, sometimes these hosts (if they exist!) don't even run name servers." (Quoted from [RFC1713], Section 2.3)

Glue records: "[Resource records] which are not part of the authoritative data [of the zone], and are address resource records for the [name servers in subzones]. These RRs are only necessary if the name server's name is 'below' the cut, and are only used as part of a referral response." Without glue "we could be faced with the situation where the NS RRs tell us that in order to learn a name server's address, we should contact the server using the

address we wish to learn." (Definition from [RFC1034], Section 4.2.1)

A later definition is that glue "includes any record in a zone file that is not properly part of that zone, including nameserver records of delegated sub-zones (NS records), address records that accompany those NS records (A, AAAA, etc), and any other stray data that might appear" (Quoted from [RFC2181], Section 5.4.1). Although glue is sometimes used today with this wider definition in mind, the context surrounding the [RFC2181] definition suggests it is intended to apply to the use of glue within the document itself and not necessarily beyond.

Bailiwick: "In-bailiwick" is an adjective to describe a name server whose name is either a subdomain of or (rarely) the same as the origin of the zone that contains the delegation to the name server. In-bailiwick name servers may have glue records in their parent zone (using the first of the definitions of "glue records" in the definition above). (The term "bailiwick" means the district or territory where a bailiff or policeman has jurisdiction.)

"In-bailiwick" names are divided into two type of name server names: "in-domain" names and "sibling domain" names.

- * In-domain: an adjective to describe a name server whose name is either subordinate to or (rarely) the same as the owner name of the NS resource records. An in-domain name server name **MUST** have glue records or name resolution fails. For example, a delegation for "child.example.com" may have "in-domain" name server name "ns.child.example.com".
- * Sibling domain: a name server's name that is either subordinate to or (rarely) the same as the zone origin and not subordinate to or the same as the owner name of the NS resource records. Glue records for sibling domains are allowed, but not necessary. For example, a delegation for "child.example.com" in "example.com" zone may have "sibling" name server name "ns.another.example.com".

"Out-of-bailiwick" is the antonym of in-bailiwick. An adjective to describe a name server whose name is not subordinate to or the same as the zone origin. Glue records for out-of-bailiwick name servers are useless. Following table shows examples of delegation types.

Delegation	Parent	Name Server Name	Type
com	.	a.gtld-servers.net	in-bailiwick / sibling domain
net	.	a.gtld-servers.net	in-bailiwick / in-domain
example.org	org	ns.example.org	in-bailiwick / in-domain
example.org	org	ns.ietf.org	in-bailiwick / sibling domain
example.org	org	ns.example.com	out-of-bailiwick
example.jp	jp	ns.example.jp	in-bailiwick / in-domain
example.jp	jp	ns.example.ne.jp	in-bailiwick / sibling domain
example.jp	jp	ns.example.com	out-of-bailiwick

Root zone: The zone of a DNS-based tree whose apex is the zero-length label. Also sometimes called "the DNS root".

Empty non-terminals (ENT): "Domain names that own no resource records but have subdomains that do." (Quoted from [RFC4592], Section 2.2.2.) A typical example is in SRV records: in the name "_sip._tcp.example.com", it is likely that "_tcp.example.com" has no RRsets, but that "_sip._tcp.example.com" has (at least) an SRV RRset.

Delegation-centric zone: A zone that consists mostly of delegations to child zones. This term is used in contrast to a zone that might have some delegations to child zones, but also has many data resource records for the zone itself and/or for child zones. The term is used in [RFC4956] and [RFC5155], but is not defined there.

Occluded name: "The addition of a delegation point via dynamic update will render all subordinate domain names to be in a limbo, still part of the zone, but not available to the lookup process. The addition of a DNAME resource record has the same impact. The subordinate names are said to be 'occluded'." (Quoted from [RFC5936], Section 3.5)

Fast flux DNS: This "occurs when a domain is found in DNS using A records to multiple IP addresses, each of which has a very short Time-to-Live (TTL) value associated with it. This means that the domain resolves to varying IP addresses over a short period of time." (Quoted from [RFC6561], Section 1.1.5, with typo corrected) In addition to having legitimate uses, fast flux DNS can be used to deliver malware. Because the addresses change so rapidly, it is difficult to ascertain all the hosts. It should be noted that the technique also works with AAAA records, but such use is not frequently observed on the Internet as of this writing.

Reverse DNS, reverse lookup: "The process of mapping an address to a name is generally known as a 'reverse lookup', and the IN-

ADDR.ARPA and IP6.ARPA zones are said to support the 'reverse DNS'." (Quoted from [RFC5855], Section 1)

Forward lookup: "Hostname-to-address translation". (Quoted from [RFC2133], Section 6)

arpa: Address and Routing Parameter Area Domain: "The 'arpa' domain was originally established as part of the initial deployment of the DNS, to provide a transition mechanism from the Host Tables that were common in the ARPANET, as well as a home for the IPv4 reverse mapping domain. During 2000, the abbreviation was redesignated to 'Address and Routing Parameter Area' in the hope of reducing confusion with the earlier network name." (Quoted from [RFC3172], Section 2.) .arpa is an "infrastructure domain", a domain whose "role is to support the operating infrastructure of the Internet". (Quoted from [RFC3172], Section 2.) See [RFC3172] for more history of this name.

Service name: "Service names are the unique key in the Service Name and Transport Protocol Port Number registry. This unique symbolic name for a service may also be used for other purposes, such as in DNS SRV records." (Quoted from [RFC6335], Section 5.)

8. Wildcards

Wildcard: [RFC1034] defined "wildcard", but in a way that turned out to be confusing to implementers. For an extended discussion of wildcards, including clearer definitions, see [RFC4592]. Special treatment is given to RRs with owner names starting with the label "*". "Such RRs are called 'wildcards'. Wildcard RRs can be thought of as instructions for synthesizing RRs." (Quoted from [RFC1034], Section 4.3.3)

Asterisk label: "The first octet is the normal label type and length for a 1-octet-long label, and the second octet is the ASCII representation for the '*' character. A descriptive name of a label equaling that value is an 'asterisk label'." (Quoted from [RFC4592], Section 2.1.1)

Wildcard domain name: "A 'wildcard domain name' is defined by having its initial (i.e., leftmost or least significant) label be asterisk label." (Quoted from [RFC4592], Section 2.1.1)

Closest encloser: "The longest existing ancestor of a name." (Quoted from [RFC5155], Section 1.3) An earlier definition is "The node in the zone's tree of existing domain names that has the most labels matching the query name (consecutively, counting from the root label downward). Each match is a 'label match' and the order

of the labels is the same." (Quoted from [RFC4592], Section 3.3.1)

Closest provable enclosure: "The longest ancestor of a name that can be proven to exist. Note that this is only different from the closest enclosure in an Opt-Out zone." (Quoted from [RFC5155], Section 1.3) See Section 10 for more on "opt-out".

Next closer name: "The name one label longer than the closest provable enclosure of a name." (Quoted from [RFC5155], Section 1.3)

Source of Synthesis: "The source of synthesis is defined in the context of a query process as that wildcard domain name immediately descending from the closest enclosure, provided that this wildcard domain name exists. 'Immediately descending' means that the source of synthesis has a name of the form: <asterisk label>.<closest enclosure>." (Quoted from [RFC4592], Section 3.3.1)

9. Registration Model

Registry: The administrative operation of a zone that allows registration of names within that zone. People often use this term to refer only to those organizations that perform registration in large delegation-centric zones (such as TLDs); but formally, whoever decides what data goes into a zone is the registry for that zone. This definition of "registry" is from a DNS point of view; for some zones, the policies that determine what can go in the zone are decided by zones that are superordinate and not the registry operator.

Registrant: An individual or organization on whose behalf a name in a zone is registered by the registry. In many zones, the registry and the registrant may be the same entity, but in TLDs they often are not.

Registrar: A service provider that acts as a go-between for registrants and registries. Not all registrations require a registrar, though it is common to have registrars involved in registrations in TLDs.

EPP: The Extensible Provisioning Protocol (EPP), which is commonly used for communication of registration information between registries and registrars. EPP is defined in [RFC5730].

WHOIS: A protocol specified in [RFC3912], often used for querying registry databases. WHOIS data is frequently used to associate

registration data (such as zone management contacts) with domain names. The term "WHOIS data" is often used as a synonym for the registry database, even though that database may be served by different protocols, particularly RDAP. The WHOIS protocol is also used with IP address registry data.

RDAP: The Registration Data Access Protocol, defined in [RFC7480], [RFC7481], [RFC7482], [RFC7483], [RFC7484], and [RFC7485]. The RDAP protocol and data format are meant as a replacement for WHOIS.

DNS operator: An entity responsible for running DNS servers. For a zone's authoritative servers, the registrant may act as their own DNS operator, or their registrar may do it on their behalf, or they may use a third-party operator. For some zones, the registry function is performed by the DNS operator plus other entities who decide about the allowed contents of the zone.

Public suffix: "A domain that is controlled by a public registry." (Quoted from [RFC6265], Section 5.3) A common definition for this term is a domain under which subdomains can be registered by third parties, and on which HTTP cookies (which are described in detail in [RFC6265]) should not be set. There is no indication in a domain name whether it is a public suffix; that can only be determined by outside means. In fact, both a domain and a subdomain of that domain can be public suffixes.

There is nothing inherent in a domain name to indicate whether it is a public suffix. One resource for identifying public suffixes is the Public Suffix List (PSL) maintained by Mozilla (<http://publicsuffix.org/>).

For example, at the time this document is published, the "com.au" domain is listed as a public suffix in the PSL. (Note that this example might change in the future.)

Note that the term "public suffix" is controversial in the DNS community for many reasons, and may be significantly changed in the future. One example of the difficulty of calling a domain a public suffix is that designation can change over time as the registration policy for the zone changes, such as was the case with the "uk" TLD in 2014.

Subordinate and Superordinate: These terms are introduced in [RFC3731] for use in the registration model, but not defined there. Instead, they are given in examples. "For example, domain name 'example.com' has a superordinate relationship to host name ns1.example.com'." "For example, host ns1.example1.com is a

subordinate host of domain example1.com, but it is not a subordinate host of domain example2.com." (Quoted from [RFC3731], Section 1.1.) These terms are strictly ways of referring to the relationship standing of two domains where one is a subdomain of the other.

10. General DNSSEC

Most DNSSEC terms are defined in [RFC4033], [RFC4034], [RFC4035], and [RFC5155]. The terms that have caused confusion in the DNS community are highlighted here.

DNSSEC-aware and DNSSEC-unaware: These two terms, which are used in some RFCs, have not been formally defined. However, Section 2 of [RFC4033] defines many types of resolvers and validators, including "non-validating security-aware stub resolver", "non-validating stub resolver", "security-aware name server", "security-aware recursive name server", "security-aware resolver", "security-aware stub resolver", and "security-oblivious 'anything'". (Note that the term "validating resolver", which is used in some places in DNSSEC-related documents, is also not defined in those RFCs, but is defined below.)

Signed zone: "A zone whose RRsets are signed and that contains properly constructed DNSKEY, Resource Record Signature (RRSIG), Next Secure (NSEC), and (optionally) DS records." (Quoted from [RFC4033], Section 2.) It has been noted in other contexts that the zone itself is not really signed, but all the relevant RRsets in the zone are signed. Nevertheless, if a zone that should be signed contains any RRsets that are not signed (or opted out), those RRsets will be treated as bogus, so the whole zone needs to be handled in some way.

It should also be noted that, since the publication of [RFC6840], NSEC records are no longer required for signed zones: a signed zone might include NSEC3 records instead. [RFC7129] provides additional background commentary and some context for the NSEC and NSEC3 mechanisms used by DNSSEC to provide authenticated denial-of-existence responses. NSEC and NSEC3 are described below.

Unsigned zone: Section 2 of [RFC4033] defines this as "a zone that is not signed". Section 2 of [RFC4035] defines this as "A zone that does not include these records [properly constructed DNSKEY, Resource Record Signature (RRSIG), Next Secure (NSEC), and (optionally) DS records] according to the rules in this section". There is an important note at the end of Section 5.2 of [RFC4035] that defines an additional situation in which a zone is considered unsigned: "If the resolver does not support any of the algorithms

listed in an authenticated DS RRset, then the resolver will not be able to verify the authentication path to the child zone. In this case, the resolver SHOULD treat the child zone as if it were unsigned."

NSEC: "The NSEC record allows a security-aware resolver to authenticate a negative reply for either name or type non-existence with the same mechanisms used to authenticate other DNS replies." (Quoted from [RFC4033], Section 3.2.) In short, an NSEC record provides authenticated denial of existence.

"The NSEC resource record lists two separate things: the next owner name (in the canonical ordering of the zone) that contains authoritative data or a delegation point NS RRset, and the set of RR types present at the NSEC RR's owner name." (Quoted from Section 4 of RFC 4034)

NSEC3: Like the NSEC record, the NSEC3 record also provides authenticated denial of existence; however, NSEC3 records mitigate against zone enumeration and support Opt-Out. NSEC3 resource records require associated NSEC3PARAM resource records. NSEC3 and NSEC3PARAM resource records are defined in [RFC5155].

Note that [RFC6840] says that [RFC5155] "is now considered part of the DNS Security Document Family as described by Section 10 of [RFC4033]". This means that some of the definitions from earlier RFCs that only talk about NSEC records should probably be considered to be talking about both NSEC and NSEC3.

Opt-out: "The Opt-Out Flag indicates whether this NSEC3 RR may cover unsigned delegations." (Quoted from [RFC5155], Section 3.1.2.1.) Opt-out tackles the high costs of securing a delegation to an insecure zone. When using Opt-Out, names that are an insecure delegation (and empty non-terminals that are only derived from insecure delegations) don't require an NSEC3 record or its corresponding RRSIG records. Opt-Out NSEC3 records are not able to prove or deny the existence of the insecure delegations. (Adapted from [RFC7129], Section 5.1)

Insecure delegation: "A signed name containing a delegation (NS RRset), but lacking a DS RRset, signifying a delegation to an unsigned subzone." (Quoted from [RFC4956], Section 2.)

Zone enumeration: "The practice of discovering the full content of a zone via successive queries." (Quoted from [RFC5155], Section 1.3.) This is also sometimes called "zone walking". Zone enumeration is different from zone content guessing where the guesser uses a large dictionary of possible labels and sends

successive queries for them, or matches the contents of NSEC3 records against such a dictionary.

Validation: Validation, in the context of DNSSEC, refers to one of the following:

- * Checking the validity of DNSSEC signatures
- * Checking the validity of DNS responses, such as those including authenticated denial of existence
- * Building an authentication chain from a trust anchor to a DNS response or individual DNS RRsets in a response

The first two definitions above consider only the validity of individual DNSSEC components such as the RRSIG validity or NSEC proof validity. The third definition considers the components of the entire DNSSEC authentication chain, and thus requires "configured knowledge of at least one authenticated DNSKEY or DS RR" (as described in [RFC4035], Section 5).

[RFC4033], Section 2, says that a "Validating Security-Aware Stub Resolver... performs signature validation" and uses a trust anchor "as a starting point for building the authentication chain to a signed DNS response", and thus uses the first and third definitions above. The process of validating an RRSIG resource record is described in [RFC4035], Section 5.3.

[RFC5155] refers to validating responses throughout the document, in the context of hashed authenticated denial of existence; this uses the second definition above.

The term "authentication" is used interchangeably with "validation", in the sense of the third definition above. [RFC4033], Section 2, describes the chain linking trust anchor to DNS data as the "authentication chain". A response is considered to be authentic if "all RRsets in the Answer and Authority sections of the response [are considered] to be authentic" (Quoted from [RFC4035]). DNS data or responses deemed to be authentic or validated have a security status of "secure" ([RFC4035], Section 4.3; [RFC4033], Section 5). "Authenticating both DNS keys and data is a matter of local policy, which may extend or even override the [DNSSEC] protocol extensions" (Quoted from [RFC4033], Section 3.1).

The term "verification", when used, is usually synonym for "validation".

Validating resolver: A security-aware recursive name server, security-aware resolver, or security-aware stub resolver that is applying at least one of the definitions of validation (above), as appropriate to the resolution context. For the same reason that the generic term "resolver" is sometimes ambiguous and needs to be evaluated in context (see Section 6), "validating resolver" is a context-sensitive term.

Key signing key (KSK): DNSSEC keys that "only sign the apex DNSKEY RRset in a zone." (Quoted from [RFC6781], Section 3.1)

Zone signing key (ZSK): "DNSSEC keys that can be used to sign all the RRsets in a zone that require signatures, other than the apex DNSKEY RRset." (Quoted from [RFC6781], Section 3.1) Also note that a ZSK is sometimes used to sign the apex DNSKEY RRset.

Combined signing key (CSK): "In cases where the differentiation between the KSK and ZSK is not made, i.e., where keys have the role of both KSK and ZSK, we talk about a Single-Type Signing Scheme." (Quoted from [RFC6781], Section 3.1) This is sometimes called a "combined signing key" or CSK. It is operational practice, not protocol, that determines whether a particular key is a ZSK, a KSK, or a CSK.

Secure Entry Point (SEP): A flag in the DNSKEY RDATA that "can be used to distinguish between keys that are intended to be used as the secure entry point into the zone when building chains of trust, i.e., they are (to be) pointed to by parental DS RRs or configured as a trust anchor. Therefore, it is suggested that the SEP flag be set on keys that are used as KSKs and not on keys that are used as ZSKs, while in those cases where a distinction between a KSK and ZSK is not made (i.e., for a Single-Type Signing Scheme), it is suggested that the SEP flag be set on all keys." (Quoted from [RFC6781], Section 3.2.3.) Note that the SEP flag is only a hint, and its presence or absence may not be used to disqualify a given DNSKEY RR from use as a KSK or ZSK during validation.

The original definition of SEPs was in [RFC3757]. That definition clearly indicated that the SEP was a key, not just a bit in the key. The abstract of [RFC3757] says: "With the Delegation Signer (DS) resource record (RR), the concept of a public key acting as a secure entry point (SEP) has been introduced. During exchanges of public keys with the parent there is a need to differentiate SEP keys from other public keys in the Domain Name System KEY (DNSKEY) resource record set. A flag bit in the DNSKEY RR is defined to indicate that DNSKEY is to be used as a SEP." That definition of

the SEP as a key was made obsolete by [RFC4034], and the definition from [RFC6781] is consistent with [RFC4034].

Trust anchor: "A configured DNSKEY RR or DS RR hash of a DNSKEY RR. A validating security-aware resolver uses this public key or hash as a starting point for building the authentication chain to a signed DNS response. In general, a validating resolver will have to obtain the initial values of its trust anchors via some secure or trusted means outside the DNS protocol." (Quoted from [RFC4033], Section 2)

DNSSEC Policy (DP): A statement that "sets forth the security requirements and standards to be implemented for a DNSSEC-signed zone." (Quoted from [RFC6841], Section 2)

DNSSEC Practice Statement (DPS): "A practices disclosure document that may support and be a supplemental document to the DNSSEC Policy (if such exists), and it states how the management of a given zone implements procedures and controls at a high level." (Quoted from [RFC6841], Section 2)

Hardware security module (HSM): A specialized piece of hardware that is used to create keys for signatures and to sign messages without ever disclosing the private key. In DNSSEC, HSMs are often used to hold the private keys for KSKs and ZSKs and to create the signatures used in RRSIG records at periodic intervals.

Signing software: Authoritative DNS servers that support DNSSEC often contain software that facilitates the creation and maintenance of DNSSEC signatures in zones. There is also stand-alone software that can be used to sign a zone regardless of whether the authoritative server itself supports signing. Sometimes signing software can support particular HSMs as part of the signing process.

11. DNSSEC States

A validating resolver can determine that a response is in one of four states: secure, insecure, bogus, or indeterminate. These states are defined in [RFC4033] and [RFC4035], although the two definitions differ a bit. This document makes no effort to reconcile the two definitions, and takes no position as to whether they need to be reconciled.

Section 5 of [RFC4033] says:

A validating resolver can determine the following 4 states:

Secure: The validating resolver has a trust anchor, has a chain of trust, and is able to verify all the signatures in the response.

Insecure: The validating resolver has a trust anchor, a chain of trust, and, at some delegation point, signed proof of the non-existence of a DS record. This indicates that subsequent branches in the tree are provably insecure. A validating resolver may have a local policy to mark parts of the domain space as insecure.

Bogus: The validating resolver has a trust anchor and a secure delegation indicating that subsidiary data is signed, but the response fails to validate for some reason: missing signatures, expired signatures, signatures with unsupported algorithms, data missing that the relevant NSEC RR says should be present, and so forth.

Indeterminate: There is no trust anchor that would indicate that a specific portion of the tree is secure. This is the default operation mode.

Section 4.3 of [RFC4035] says:

A security-aware resolver must be able to distinguish between four cases:

Secure: An RRset for which the resolver is able to build a chain of signed DNSKEY and DS RRs from a trusted security anchor to the RRset. In this case, the RRset should be signed and is subject to signature validation, as described above.

Insecure: An RRset for which the resolver knows that it has no chain of signed DNSKEY and DS RRs from any trusted starting point to the RRset. This can occur when the target RRset lies in an unsigned zone or in a descendent [sic] of an unsigned zone. In this case, the RRset may or may not be signed, but the resolver will not be able to verify the signature.

Bogus: An RRset for which the resolver believes that it ought to be able to establish a chain of trust but for which it is unable to do so, either due to signatures that for some reason fail to validate or due to missing data that the relevant DNSSEC RRs indicate should be present. This case may indicate an attack but may also indicate a configuration error or some form of data corruption.

Indeterminate: An RRset for which the resolver is not able to determine whether the RRset should be signed, as the resolver is not able to obtain the necessary DNSSEC RRs. This can occur when the security-aware resolver is not able to contact security-aware name servers for the relevant zones.

12. Security Considerations

These definitions do not change any security considerations for the DNS.

13. IANA Considerations

None.

14. References

14.1. Normative References

[IANA_RootFiles]

Internet Assigned Numbers Authority, "IANA Root Files", 2016, <<http://www.iana.org/domains/root/files>>.

- [RFC0882] Mockapetris, P., "Domain names: Concepts and facilities", RFC 882, DOI 10.17487/RFC0882, November 1983, <<https://www.rfc-editor.org/info/rfc882>>.
- [RFC1034] Mockapetris, P., "Domain names - concepts and facilities", STD 13, RFC 1034, DOI 10.17487/RFC1034, November 1987, <<https://www.rfc-editor.org/info/rfc1034>>.
- [RFC1035] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, DOI 10.17487/RFC1035, November 1987, <<https://www.rfc-editor.org/info/rfc1035>>.
- [RFC1123] Braden, R., Ed., "Requirements for Internet Hosts - Application and Support", STD 3, RFC 1123, DOI 10.17487/RFC1123, October 1989, <<https://www.rfc-editor.org/info/rfc1123>>.
- [RFC1912] Barr, D., "Common DNS Operational and Configuration Errors", RFC 1912, DOI 10.17487/RFC1912, February 1996, <<https://www.rfc-editor.org/info/rfc1912>>.
- [RFC1996] Vixie, P., "A Mechanism for Prompt Notification of Zone Changes (DNS NOTIFY)", RFC 1996, DOI 10.17487/RFC1996, August 1996, <<https://www.rfc-editor.org/info/rfc1996>>.
- [RFC2136] Vixie, P., Ed., Thomson, S., Rekhter, Y., and J. Bound, "Dynamic Updates in the Domain Name System (DNS UPDATE)", RFC 2136, DOI 10.17487/RFC2136, April 1997, <<https://www.rfc-editor.org/info/rfc2136>>.
- [RFC2181] Elz, R. and R. Bush, "Clarifications to the DNS Specification", RFC 2181, DOI 10.17487/RFC2181, July 1997, <<https://www.rfc-editor.org/info/rfc2181>>.
- [RFC2182] Elz, R., Bush, R., Bradner, S., and M. Patton, "Selection and Operation of Secondary DNS Servers", BCP 16, RFC 2182, DOI 10.17487/RFC2182, July 1997, <<https://www.rfc-editor.org/info/rfc2182>>.
- [RFC2308] Andrews, M., "Negative Caching of DNS Queries (DNS NCACHE)", RFC 2308, DOI 10.17487/RFC2308, March 1998, <<https://www.rfc-editor.org/info/rfc2308>>.
- [RFC3731] Hollenbeck, S., "Extensible Provisioning Protocol (EPP) Domain Name Mapping", RFC 3731, DOI 10.17487/RFC3731, March 2004, <<https://www.rfc-editor.org/info/rfc3731>>.

- [RFC4033] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "DNS Security Introduction and Requirements", RFC 4033, DOI 10.17487/RFC4033, March 2005, <<https://www.rfc-editor.org/info/rfc4033>>.
- [RFC4034] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "Resource Records for the DNS Security Extensions", RFC 4034, DOI 10.17487/RFC4034, March 2005, <<https://www.rfc-editor.org/info/rfc4034>>.
- [RFC4035] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "Protocol Modifications for the DNS Security Extensions", RFC 4035, DOI 10.17487/RFC4035, March 2005, <<https://www.rfc-editor.org/info/rfc4035>>.
- [RFC4592] Lewis, E., "The Role of Wildcards in the Domain Name System", RFC 4592, DOI 10.17487/RFC4592, July 2006, <<https://www.rfc-editor.org/info/rfc4592>>.
- [RFC5155] Laurie, B., Sisson, G., Arends, R., and D. Blacka, "DNS Security (DNSSEC) Hashed Authenticated Denial of Existence", RFC 5155, DOI 10.17487/RFC5155, March 2008, <<https://www.rfc-editor.org/info/rfc5155>>.
- [RFC5358] Damas, J. and F. Neves, "Preventing Use of Recursive Nameservers in Reflector Attacks", BCP 140, RFC 5358, DOI 10.17487/RFC5358, October 2008, <<https://www.rfc-editor.org/info/rfc5358>>.
- [RFC5730] Hollenbeck, S., "Extensible Provisioning Protocol (EPP)", STD 69, RFC 5730, DOI 10.17487/RFC5730, August 2009, <<https://www.rfc-editor.org/info/rfc5730>>.
- [RFC5855] Abley, J. and T. Manderson, "Nameservers for IPv4 and IPv6 Reverse Zones", BCP 155, RFC 5855, DOI 10.17487/RFC5855, May 2010, <<https://www.rfc-editor.org/info/rfc5855>>.
- [RFC5936] Lewis, E. and A. Hoenes, Ed., "DNS Zone Transfer Protocol (AXFR)", RFC 5936, DOI 10.17487/RFC5936, June 2010, <<https://www.rfc-editor.org/info/rfc5936>>.
- [RFC6561] Livingood, J., Mody, N., and M. O'Reirdan, "Recommendations for the Remediation of Bots in ISP Networks", RFC 6561, DOI 10.17487/RFC6561, March 2012, <<https://www.rfc-editor.org/info/rfc6561>>.

- [RFC6781] Kolkman, O., Mekking, W., and R. Gieben, "DNSSEC Operational Practices, Version 2", RFC 6781, DOI 10.17487/RFC6781, December 2012, <<https://www.rfc-editor.org/info/rfc6781>>.
- [RFC6840] Weiler, S., Ed. and D. Blacka, Ed., "Clarifications and Implementation Notes for DNS Security (DNSSEC)", RFC 6840, DOI 10.17487/RFC6840, February 2013, <<https://www.rfc-editor.org/info/rfc6840>>.
- [RFC6841] Ljunggren, F., Eklund Lowinder, AM., and T. Okubo, "A Framework for DNSSEC Policies and DNSSEC Practice Statements", RFC 6841, DOI 10.17487/RFC6841, January 2013, <<https://www.rfc-editor.org/info/rfc6841>>.
- [RFC6891] Damas, J., Graff, M., and P. Vixie, "Extension Mechanisms for DNS (EDNS(0))", STD 75, RFC 6891, DOI 10.17487/RFC6891, April 2013, <<https://www.rfc-editor.org/info/rfc6891>>.
- [RFC7344] Kumari, W., Gudmundsson, O., and G. Barwood, "Automating DNSSEC Delegation Trust Maintenance", RFC 7344, DOI 10.17487/RFC7344, September 2014, <<https://www.rfc-editor.org/info/rfc7344>>.
- [RFC7719] Hoffman, P., Sullivan, A., and K. Fujiwara, "DNS Terminology", RFC 7719, DOI 10.17487/RFC7719, December 2015, <<https://www.rfc-editor.org/info/rfc7719>>.
- [RFC8310] Dickinson, S., Gillmor, D., and T. Reddy, "Usage Profiles for DNS over TLS and DNS over DTLS", RFC 8310, DOI 10.17487/RFC8310, March 2018, <<https://www.rfc-editor.org/info/rfc8310>>.

14.2. Informative References

- [I-D.ietf-doh-dns-over-https]
Hoffman, P. and P. McManus, "DNS Queries over HTTPS (DoH)", draft-ietf-doh-dns-over-https-14 (work in progress), August 2018.
- [IANA_Resource_Registry]
Internet Assigned Numbers Authority, "Resource Record (RR) TYPES", 2017, <<http://www.iana.org/assignments/dns-parameters/>>.

- [RFC0819] Su, Z. and J. Postel, "The Domain Naming Convention for Internet User Applications", RFC 819, DOI 10.17487/RFC0819, August 1982, <<https://www.rfc-editor.org/info/rfc819>>.
- [RFC0952] Harrenstien, K., Stahl, M., and E. Feinler, "DoD Internet host table specification", RFC 952, DOI 10.17487/RFC0952, October 1985, <<https://www.rfc-editor.org/info/rfc952>>.
- [RFC1713] Romao, A., "Tools for DNS debugging", FYI 27, RFC 1713, DOI 10.17487/RFC1713, November 1994, <<https://www.rfc-editor.org/info/rfc1713>>.
- [RFC1995] Ohta, M., "Incremental Zone Transfer in DNS", RFC 1995, DOI 10.17487/RFC1995, August 1996, <<https://www.rfc-editor.org/info/rfc1995>>.
- [RFC2133] Gilligan, R., Thomson, S., Bound, J., and W. Stevens, "Basic Socket Interface Extensions for IPv6", RFC 2133, DOI 10.17487/RFC2133, April 1997, <<https://www.rfc-editor.org/info/rfc2133>>.
- [RFC2775] Carpenter, B., "Internet Transparency", RFC 2775, DOI 10.17487/RFC2775, February 2000, <<https://www.rfc-editor.org/info/rfc2775>>.
- [RFC3172] Huston, G., Ed., "Management Guidelines & Operational Requirements for the Address and Routing Parameter Area Domain ("arpa")", BCP 52, RFC 3172, DOI 10.17487/RFC3172, September 2001, <<https://www.rfc-editor.org/info/rfc3172>>.
- [RFC3425] Lawrence, D., "Obsoleting IQUERY", RFC 3425, DOI 10.17487/RFC3425, November 2002, <<https://www.rfc-editor.org/info/rfc3425>>.
- [RFC3757] Kolkman, O., Schlyter, J., and E. Lewis, "Domain Name System KEY (DNSKEY) Resource Record (RR) Secure Entry Point (SEP) Flag", RFC 3757, DOI 10.17487/RFC3757, April 2004, <<https://www.rfc-editor.org/info/rfc3757>>.
- [RFC3912] Daigle, L., "WHOIS Protocol Specification", RFC 3912, DOI 10.17487/RFC3912, September 2004, <<https://www.rfc-editor.org/info/rfc3912>>.
- [RFC4641] Kolkman, O. and R. Gieben, "DNSSEC Operational Practices", RFC 4641, DOI 10.17487/RFC4641, September 2006, <<https://www.rfc-editor.org/info/rfc4641>>.

- [RFC4697] Larson, M. and P. Barber, "Observed DNS Resolution Misbehavior", BCP 123, RFC 4697, DOI 10.17487/RFC4697, October 2006, <<https://www.rfc-editor.org/info/rfc4697>>.
- [RFC4786] Abley, J. and K. Lindqvist, "Operation of Anycast Services", BCP 126, RFC 4786, DOI 10.17487/RFC4786, December 2006, <<https://www.rfc-editor.org/info/rfc4786>>.
- [RFC4956] Arends, R., Kosters, M., and D. Blacka, "DNS Security (DNSSEC) Opt-In", RFC 4956, DOI 10.17487/RFC4956, July 2007, <<https://www.rfc-editor.org/info/rfc4956>>.
- [RFC5625] Bellis, R., "DNS Proxy Implementation Guidelines", BCP 152, RFC 5625, DOI 10.17487/RFC5625, August 2009, <<https://www.rfc-editor.org/info/rfc5625>>.
- [RFC5890] Klensin, J., "Internationalized Domain Names for Applications (IDNA): Definitions and Document Framework", RFC 5890, DOI 10.17487/RFC5890, August 2010, <<https://www.rfc-editor.org/info/rfc5890>>.
- [RFC5891] Klensin, J., "Internationalized Domain Names in Applications (IDNA): Protocol", RFC 5891, DOI 10.17487/RFC5891, August 2010, <<https://www.rfc-editor.org/info/rfc5891>>.
- [RFC5892] Faltstrom, P., Ed., "The Unicode Code Points and Internationalized Domain Names for Applications (IDNA)", RFC 5892, DOI 10.17487/RFC5892, August 2010, <<https://www.rfc-editor.org/info/rfc5892>>.
- [RFC5893] Alvestrand, H., Ed. and C. Karp, "Right-to-Left Scripts for Internationalized Domain Names for Applications (IDNA)", RFC 5893, DOI 10.17487/RFC5893, August 2010, <<https://www.rfc-editor.org/info/rfc5893>>.
- [RFC5894] Klensin, J., "Internationalized Domain Names for Applications (IDNA): Background, Explanation, and Rationale", RFC 5894, DOI 10.17487/RFC5894, August 2010, <<https://www.rfc-editor.org/info/rfc5894>>.
- [RFC6055] Thaler, D., Klensin, J., and S. Cheshire, "IAB Thoughts on Encodings for Internationalized Domain Names", RFC 6055, DOI 10.17487/RFC6055, February 2011, <<https://www.rfc-editor.org/info/rfc6055>>.

- [RFC6265] Barth, A., "HTTP State Management Mechanism", RFC 6265, DOI 10.17487/RFC6265, April 2011, <<https://www.rfc-editor.org/info/rfc6265>>.
- [RFC6303] Andrews, M., "Locally Served DNS Zones", BCP 163, RFC 6303, DOI 10.17487/RFC6303, July 2011, <<https://www.rfc-editor.org/info/rfc6303>>.
- [RFC6335] Cotton, M., Eggert, L., Touch, J., Westerlund, M., and S. Cheshire, "Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry", BCP 165, RFC 6335, DOI 10.17487/RFC6335, August 2011, <<https://www.rfc-editor.org/info/rfc6335>>.
- [RFC6365] Hoffman, P. and J. Klensin, "Terminology Used in Internationalization in the IETF", BCP 166, RFC 6365, DOI 10.17487/RFC6365, September 2011, <<https://www.rfc-editor.org/info/rfc6365>>.
- [RFC6672] Rose, S. and W. Wijngaards, "DNAME Redirection in the DNS", RFC 6672, DOI 10.17487/RFC6672, June 2012, <<https://www.rfc-editor.org/info/rfc6672>>.
- [RFC6762] Cheshire, S. and M. Krochmal, "Multicast DNS", RFC 6762, DOI 10.17487/RFC6762, February 2013, <<https://www.rfc-editor.org/info/rfc6762>>.
- [RFC7129] Gieben, R. and W. Mekking, "Authenticated Denial of Existence in the DNS", RFC 7129, DOI 10.17487/RFC7129, February 2014, <<https://www.rfc-editor.org/info/rfc7129>>.
- [RFC7480] Newton, A., Ellacott, B., and N. Kong, "HTTP Usage in the Registration Data Access Protocol (RDAP)", RFC 7480, DOI 10.17487/RFC7480, March 2015, <<https://www.rfc-editor.org/info/rfc7480>>.
- [RFC7481] Hollenbeck, S. and N. Kong, "Security Services for the Registration Data Access Protocol (RDAP)", RFC 7481, DOI 10.17487/RFC7481, March 2015, <<https://www.rfc-editor.org/info/rfc7481>>.
- [RFC7482] Newton, A. and S. Hollenbeck, "Registration Data Access Protocol (RDAP) Query Format", RFC 7482, DOI 10.17487/RFC7482, March 2015, <<https://www.rfc-editor.org/info/rfc7482>>.

- [RFC7483] Newton, A. and S. Hollenbeck, "JSON Responses for the Registration Data Access Protocol (RDAP)", RFC 7483, DOI 10.17487/RFC7483, March 2015, <<https://www.rfc-editor.org/info/rfc7483>>.
- [RFC7484] Blanchet, M., "Finding the Authoritative Registration Data (RDAP) Service", RFC 7484, DOI 10.17487/RFC7484, March 2015, <<https://www.rfc-editor.org/info/rfc7484>>.
- [RFC7485] Zhou, L., Kong, N., Shen, S., Sheng, S., and A. Servin, "Inventory and Analysis of WHOIS Registration Objects", RFC 7485, DOI 10.17487/RFC7485, March 2015, <<https://www.rfc-editor.org/info/rfc7485>>.
- [RFC7858] Hu, Z., Zhu, L., Heidemann, J., Mankin, A., Wessels, D., and P. Hoffman, "Specification for DNS over Transport Layer Security (TLS)", RFC 7858, DOI 10.17487/RFC7858, May 2016, <<https://www.rfc-editor.org/info/rfc7858>>.
- [RFC8094] Reddy, T., Wing, D., and P. Patil, "DNS over Datagram Transport Layer Security (DTLS)", RFC 8094, DOI 10.17487/RFC8094, February 2017, <<https://www.rfc-editor.org/info/rfc8094>>.
- [RFC8109] Koch, P., Larson, M., and P. Hoffman, "Initializing a DNS Resolver with Priming Queries", BCP 209, RFC 8109, DOI 10.17487/RFC8109, March 2017, <<https://www.rfc-editor.org/info/rfc8109>>.
- [RSSAC026] Root Server System Advisory Committee (RSSAC), "RSSAC Lexicon", 2017, <<https://www.icann.org/en/system/files/files/rssac-026-14mar17-en.pdf>>.

Appendix A. Definitions Updated by this Document

The following definitions from RFCs are updated by this document:

- o Forwarder in [RFC2308]
- o QNAME in [RFC2308]
- o Secure Entry Point (SEP) in [RFC3757]; note, however, that this RFC is already obsolete

Appendix B. Definitions First Defined in this Document

The following definitions are first defined in this document:

- o "Alias" in Section 2
- o "Apex" in Section 7
- o "arpa" in Section 7
- o "Bailiwick" in Section 7
- o "Class independent" in Section 5
- o "Delegation-centric zone" in Section 7
- o "Delegation" in Section 7
- o "DNS operator" in Section 9
- o "DNSSEC-aware" in Section 10
- o "DNSSEC-unaware" in Section 10
- o "Forwarding" in Section 6
- o "Full resolver" in Section 6
- o "Fully qualified domain name" in Section 2
- o "Global DNS" in Section 2
- o "Hardware Security Module (HSM)" in Section 10
- o "Host name" in Section 2
- o "IDN" in Section 2
- o "In-bailiwick" in Section 7
- o "Iterative resolution" in Section 6
- o "Label" in Section 2
- o "Locally served DNS zone" in Section 2
- o "Naming system" in Section 2

- o "Negative response" in Section 3
- o "Non-recursive query" in Section 6
- o "Open resolver" in Section 6
- o "Out-of-bailiwick" in Section 7
- o "Passive DNS" in Section 6
- o "Policy-implementing resolver" in Section 6
- o "Presentation format" in Section 5
- o "Priming" in Section 6
- o "Private DNS" in Section 2
- o "Recursive resolver" in Section 6
- o "Referrals" in Section 4
- o "Registrant" in Section 9
- o "Registrar" in Section 9
- o "Registry" in Section 9
- o "Root zone" in Section 7
- o "Secure Entry Point (SEP)" in Section 10
- o "Signing software" in Section 10
- o "Split DNS" in Section 6
- o "Stub resolver" in Section 6
- o "Subordinate" in Section 8
- o "Superordinate" in Section 8
- o "TLD" in Section 2
- o "Validating resolver" in Section 10
- o "Validation" in Section 10

- o "View" in Section 6
- o "Zone transfer" in Section 6

Index

A

Address records 15
Alias 9
Anycast 21
Apex 22
Asterisk label 26
Authoritative data 23
Authoritative server 18
Authoritative-only server 18
arpa: Address and Routing Parameter Area Domain 26

C

CNAME 9
Canonical name 9
Child 22
Class 10
Class independent 15
Closest enclosure 26
Closest provable enclosure 27
Combined signing key (CSK) 32

D

DNS operator 28
DNSSEC Policy (DP) 33
DNSSEC Practice Statement (DPS) 33
DNSSEC-aware and DNSSEC-unaware 29
Delegation 23
Delegation-centric zone 25
Domain name 4

E

EDNS 14
EPP 27
Empty non-terminals (ENT) 25

F

FORMERR 9
Fast flux DNS 25
Forward lookup 26
Forwarder 20
Forwarding 19
Full resolver 17

Full-service resolver 17
Fully qualified domain name (FQDN) 7

G

Global DNS 5
Glue records 23

H

Hardware security module (HSM) 33
Hidden master 19
Host name 8

I

IDN 8
In-bailiwick 24
Insecure delegation 30
Instance 21
Iterative mode 16
Iterative resolution 17

K

Key signing key (KSK) 32

L

Label 5
Lame delegation 23
Locally served DNS zone 7

M

Master file 13
Master server 19
Multicast DNS 7

N

NODATA 10
NOERROR 9
NOTIMP 10
NS 18
NSEC 30
NSEC3 30
NXDOMAIN 9
Naming system 4
Negative caching 18
Negative response 10
Next closer name 27
Non-recursive query 17

O

OPT 14
Occluded name 25
Open resolver 20
Opt-out 30
Origin 22
Out-of-bailiwick 24
Owner 14

P

Parent 22
Passive DNS 21
Policy-implementing resolver 20
Presentation format 13
Primary master 19
Primary server 19
Priming 17
Privacy-enabling DNS server 21
Private DNS 6
Public suffix 28

Q

QNAME 11

R

RDAP 28
REFUSED 10
RR 13
RRset 13
Recursive mode 16
Recursive query 17
Recursive resolver 17
Referrals 12
Registrant 27
Registrar 27
Registry 27
Resolver 15
Reverse DNS, reverse lookup 25
Root hints 18
Root zone 25

S

SERVFAIL 9
SOA 14
SOA field names 14
Secondary server 19
Secure Entry Point (SEP) 32
Service name 26
Signed zone 29

- Signing software 33
- Slave server 18
- Source of Synthesis 27
- Split DNS 20
- Split-horizon DNS 20
- Stealth server 19
- Stub resolver 16
- Subdomain 9
- Subordinate 28
- Superordinate 28

T

- TLD 8
- TTL 14
- Trust anchor 33

U

- Unsigned zone 29

V

- Validating resolver 32
- Validation 31
- View 21

W

- WHOIS 27
- Wildcard 26
- Wildcard domain name 26

Z

- Zone 21
- Zone cut 22
- Zone enumeration 30
- Zone signing key (ZSK) 32
- Zone transfer 18

Acknowledgements

The following is the Acknowledgements for RFC 7719.

The authors gratefully acknowledge all of the authors of DNS-related RFCs that proceed this one. Comments from Tony Finch, Stephane Bortzmeyer, Niall O'Reilly, Colm MacCarthaigh, Ray Bellis, John Kristoff, Robert Edmonds, Paul Wouters, Shumon Huque, Paul Ebersman, David Lawrence, Matthijs Mekking, Casey Deccio, Bob Harold, Ed Lewis, John Klensin, David Black, and many others in the DNSOP Working Group helped shape RFC 7719.

Most of the major changes between RFC 7719 and this document came from active discussion on the DNSOP WG. Specific people who contributed material to this document include: Bob Harold, Dick Franks, Evan Hunt, John Dickinson, Mark Andrews, Martin Hoffmann, Paul Vixie, Peter Koch, Duane Wessels, Allison Mankin, Giovane Moura, Roni Even, Dan Romascanu, and Vladmir Cunat.

Authors' Addresses

Paul Hoffman
ICANN

Email: paul.hoffman@icann.org

Andrew Sullivan

Email: ajs@anvilwalrusden.com

Kazunori Fujiwara
Japan Registry Services Co., Ltd.
Chiyoda First Bldg. East 13F, 3-8-1 Nishi-Kanda
Chiyoda-ku, Tokyo 101-0065
Japan

Phone: +81 3 5215 8451
Email: fujiwara@jprs.co.jp

Internet Engineering Task Force
Internet-Draft
Intended status: Experimental
Expires: December 15, 2016

M. Sivaraman
S. Morris
R. Bellis
W. Krecicki
Internet Systems Consortium
June 13, 2016

DNS catalog zones
draft-muks-dnsop-dns-catalog-zones-01

Abstract

This document describes a method for automatic zone catalog provisioning and synchronization among DNS primary and secondary nameservers by storing and transferring the catalogs as regular DNS zones.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on December 15, 2016.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of

the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
2. Catalog zones	3
2.1. Description	3
2.2. Resource record fields	4
2.3. SOA and NS records at apex	4
2.4. Zone properties map and owner names	5
2.5. Zone property value data types	6
2.5.1. Strings	6
2.5.2. Booleans	6
2.5.3. Integers	6
2.5.4. Floating-point values	7
2.5.5. Single domain names	7
2.5.6. Unordered list of domain names	8
2.5.7. List of network addresses	8
2.5.8. Single host address	9
2.5.9. Comments	9
2.6. Catalog zone schema version	9
2.7. List of member zones	9
2.8. Zone configuration properties	10
2.8.1. zone-soa-default-serial	10
2.8.2. zone-soa-default-refresh	10
2.9. Zone properties specific to a member zone	10
2.10. Example of a catalog zone	11
3. Nameserver behavior and requirements	11
3.1. General requirements	11
3.2. Updating catalog zones	12
3.3. Implementation notes	12
4. Security considerations	13
5. IANA considerations	13
6. Acknowledgements	13
7. References	14
7.1. Normative references	14
7.2. Informative references	15
Appendix A. Glossary	15
Appendix B. Open issues and discussion (to be removed before final publication)	16
Appendix C. Change History (to be removed before final publication)	17
Authors' Addresses	17

1. Introduction

DNS nameservers implement AXFR and IXFR for zone data synchronization among a zone's primary and secondary nameservers, but the list of zones served by the primary (called a catalog in [RFC1035]) is not automatically synchronized. The administrator of a DNS nameserver farm has to synchronize such zone catalogs among primaries and their secondary nameservers manually or via an external application layer. This can be inconvenient, error-prone and dependent on the nameserver implementation.

A method for automatic zone catalog provisioning and synchronization is useful, so that the zone catalog can be maintained in a reference location by an administrator, similar to zone data.

This document describes one such method, in which the catalog is represented as a regular DNS zone called a "catalog zone", and transferred using DNS zone transfers. The representation of catalogs within DNS zones is specified and nameserver requirements are listed so that DNS implementations can support catalog zones.

The contents and representation of catalog zones are described in Section 2. Nameserver behavior is described in Section 3. A glossary of some terms used in this memo is provided in Appendix A.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

2. Catalog zones

2.1. Description

A catalog zone is a specially crafted DNS zone that contains, as DNS zone data, a list of DNS zones called member zones, associated template zone configuration common to all its member zones, and zone-specific configuration that applies to a respective zone. An implementation of catalog zones MAY allow catalog zones to include other catalog zones, but template zone configuration present in a catalog zone only applies to its immediate member zones. A catalog zone is meant to be used to provision DNS catalogs to secondary nameservers via zone transfers, for the purpose of setting up member zones to be served from these secondary nameservers.

A catalog zone uses some RR TYPEs such as PTR with alternate semantics for its purposes. Although this may be controversial, the situation is similar to other similar zone-based representations such as response-policy zones [RPZ]. A design criterion of catalog zones

is that none of the RR TYPES used therein may incur any additional section processing during DNS QUERY.

Member zones' configuration is specified as a map of zone properties, represented as a subtree of a node [RFC1034] in the domain name space inside a catalog zone. This is described in Section 2.4. Each zone property has a name and an associated value of a specific data type. Zone property value data types are described in Section 2.5. A list of permitted zone property names and their data types is given in Section 2.8.

TBD: Transitive catalogs

2.2. Resource record fields

A catalog zone contains various resource records (RRs). They have NAME, TYPE, CLASS, TTL, RDLENGTH and RDATA as fields [RFC1035].

The NAME field contains the owner name of the respective RR. As with all DNS zones, the owner name must be a child of the catalog zone name.

The TYPE field depends on the type of catalog zone property value being represented. Section 2.5 describes how various zone property value types are represented.

The CLASS field of the RR MUST be set to IN(1) [RFC1035]. This is because some RR TYPES such as APL used by catalog zones are defined only for the IN CLASS.

The TTL field's value is not specially defined by this memo. Catalog zones are for nameserver management only and are not intended for general querying. Operators should use whatever value seems convenient for any management applications that may query the catalog zone.

The RDLENGTH field contains the length of the RDATA field.

The content of the RDATA field depends on the type of catalog zone property value being represented. Section 2.5 describes how various zone property value types are represented.

2.3. SOA and NS records at apex

Similar to any other DNS zone, a catalog zone would be expected to have a syntactically correct SOA record and one or more NS records at its apex.

The SOA record's SERIAL, REFRESH, RETRY and EXPIRE fields [RFC1035] are used during zone transfer. A catalog zone's SOA SERIAL field SHOULD increase when an update is made to the catalog zone's contents as per serial number arithmetic defined in [RFC1982]. Otherwise, secondary nameservers may not notice updates to the catalog zone's contents.

The SOA record's MINIMUM field's value is not specially defined by this memo. Although they are regular DNS zones, catalog zones contain only information for the management of a set of nameservers. For this reason, operators may want to limit the systems able to query these zones.

As catalog zones do not participate in the DNS, NS records at the apex are not used but they are still required so that catalog zones are syntactically correct DNS zones. No parent delegation for the catalog zone is required. Any valid DNS name can be used in the NSDNAME field of such NS records [RFC1035] and they MUST be ignored. A single NS RR with an NSDNAME field containing the absolute name "invalid." is recommended [RFC2606].

2.4. Zone properties map and owner names

Member zones' configuration is specified as a map of zone properties, represented as a subtree of a node [RFC1034] in the domain name space inside a catalog zone. A subtree of child nodes is used for a nested map, occupying another label level. A map element's key (property name) is represented in the label at that level. For example, if a catalog zone is named "catalog1.example.org." and contains a property with name "prop0", the corresponding owner name of the node representing that property is "prop0.catalog1.example.org."

Zone property names are case-insensitive. Each zone property may use only one data type for its values. A list of permitted zone property names and their data types is given in Section 2.8.

Many properties are single-valued, but some properties can be collections with thousands of values. An example is the list of member zones within a catalog zone, which can be larger than any single RDATA instance can allow. Multiple RRs are used to represent such properties.

TBD: Currently a hashing method in owner names is used to split the elements of such properties with multiple RRs into individual RRsets, one per RR. This needs to be revisited as IXFR and DNS UPDATE both allow individual RRs within an RRset to be modified. The hashing method used is described in the appropriate property value data types in Section 2.5.

2.5. Zone property value data types

2.5.1. Strings

A property with a string value is specified using a single TXT RR [RFC1035] with owner name set to the name of the property as a sub-domain of the catalog zone name, and RDATA set to the property value.

For example, if a catalog zone is named "catalog1.example.org." and contains a property "prop0" with string value "Example", the corresponding RR would appear as follows:

```
prop0.catalog1.example.org. 3600 IN TXT "Example"
```

Here, "prop0" can contain multiple TXT RRs at that node of the domain name space [RFC1034]. The single string property SHOULD be checked by the implementation.

2.5.2. Booleans

A property with a boolean value is specified using a single TXT RR with owner name set to the name of the property as a sub-domain of the catalog zone name, and RDATA set to "true" for true condition and "false" for false condition. The RDATA is case-insensitive.

For example, if a catalog zone is named "catalog1.example.org." and contains a property "active" with boolean value false, the corresponding RR would appear as follows:

```
active.catalog1.example.org. 3600 IN TXT "false"
```

Here, "active" can contain multiple TXT RRs at that node of the domain name space [RFC1034]. The single boolean property SHOULD be checked by the implementation.

2.5.3. Integers

A property with an integer value is specified using a single TXT RR for signed integers or unsigned integers, with owner name set to the name of the property as a sub-domain of the catalog zone name, and RDATA set to the property value.

A signed integer's TXT RDATA uses the representation of an unsuffixed "integer constant" as defined in the C programming language standard [ISO.9899.1990] (of the type matching a 64-bit signed integer on that platform), with an optional minus prefix. The representation MUST be specified using a single <character-string> [RFC1034].

An unsigned integer's TXT RDATA uses the representation of an unsuffixed "integer constant" as defined in the C programming language standard [ISO.9899.1990] (of the type matching a 64-bit unsigned integer on that platform). The representation MUST be specified using a single <character-string> [RFC1034].

For example, if a catalog zone is named "catalog1.example.org." and contains a property "min-ttl" with unsigned integer value 300, the corresponding RR would appear as follows:

```
min-ttl.catalog1.example.org. 3600 IN TXT "300"
```

Here, "min-ttl" can contain multiple TXT RRs at that node of the domain name space [RFC1034]. The single integer property SHOULD be checked by the implementation.

2.5.4. Floating-point values

A property with a floating-point value is specified using a single TXT RR with owner name set to the name of the property as a sub-domain of the catalog zone name, and RDATA set to the property value.

A floating-point value's TXT RDATA uses the representation of an unsuffixed "floating constant" as defined in the C programming language standard [ISO.9899.1990]. The representation MUST be specified using a single <character-string> [RFC1034].

For example, if a catalog zone is named "catalog1.example.org." and contains a property "decay-rate" with value 0.15, the corresponding RR may appear as follows:

```
decay-rate.catalog1.example.org. 3600 IN TXT "15e-2"
```

Here, "decay-rate" can contain multiple TXT RRs at that node of the domain name space [RFC1034]. The single floating-point property SHOULD be checked by the implementation.

2.5.5. Single domain names

A property with a single domain name as value is specified using a PTR RR [RFC1035] with owner name set to the name of the property as a sub-domain of the catalog zone name, and RDATA set to the property value.

For example, if a catalog zone is named "catalog1.example.org." and contains a property "prop1" with value "vall.example.com.", the corresponding RR would appear as follows:

```
prop1.catalog1.example.org. 3600 IN PTR val1.example.com.
```

Here, "prop1" can contain multiple PTR RRs at that node of the domain name space [RFC1034]. The single domain name property SHOULD be checked by the implementation.

2.5.6. Unordered list of domain names

Let N be an absolute name formed by concatenating the RDATA hash (see Appendix A), the name of the property, and the catalog zone name in that order, such that N is a unique owner name in the catalog zone.

Then, a property containing an unordered list of domain names as value is specified using multiple PTR RRs [RFC1035] with owner name set to N, and each RR's RDATA set to each domain name in the list of the property's value respectively.

For example, if a catalog zone is named "catalog1.example.org." and contains a property "prop2" with its value being an unordered list of two names "a.example.com." and "b.example.com.", the corresponding RRs would appear as follows:

```
<hash1>.prop2.catalog1.example.org. 3600 IN PTR a.example.com.  
<hash2>.prop2.catalog1.example.org. 3600 IN PTR b.example.com.
```

Here, "prop2"'s subtree child nodes (in the domain name space [RFC1034]) can contain multiple PTR RRs at each child. For example, <hash1>.prop2 may contain multiple PTR RRs at that node. The single domain name property SHOULD be checked by the implementation.

2.5.7. List of network addresses

A property with a list of network addresses as value is specified using a single APL RR [RFC3123] with owner name set to the name of the property as a sub-domain of the catalog zone name, and RDATA set to the property value. In its presentation format, the "!" character (corresponding to the negation flag) is used to negate a network element. The exact meaning of a negated network element is left to be described by the property that APL is used for. Note that the APL RR TYPE is defined only for the IN(1) RR CLASS.

For example, if a catalog zone is named "catalog1.example.org." and contains a property "allow-query" with value [192.0.2.0/24, 198.51.100.0/24] as the list of networks, the corresponding RR would appear as follows:

```
allow-query.catalog1.example.org. 3600 IN APL (  
    1:192.0.2.0/24 2:2001:db8::/32)
```

Here, "allow-query" can contain multiple APL RRs at that node of the domain name space [RFC1034]. The single APL RR property SHOULD be checked by the implementation.

2.5.8. Single host address

A single host address is represented using the list of network addresses data type (see Section 2.5.7) with a suitable network and prefix to result in a single host address.

2.5.9. Comments

Comments may be added anywhere in a catalog zone using a scheme such as NOTE RRs [I-D.hunt-note-rr]. This memo does not depend on NOTE RRs and it is only suggested here as an informative reference.

2.6. Catalog zone schema version

The catalog zone schema version is specified by an unsigned integer property with the property name "version". All catalog zones MUST have this property present. Primary and secondary nameservers MUST NOT use catalog zones with an unexpected value value in this property, but they may be transferred as ordinary zones. For this memo, the "version" property value MUST be set to 1.

For example, if a catalog zone is named "catalog1.example.org.", the corresponding RR MUST look as follows:

```
version.catalog1.example.org. 3600 IN TXT "1"
```

Here, "version" can contain multiple TXT RRs at that node of the domain name space [RFC1034]. The single TXT RR property SHOULD be checked by the implementation.

2.7. List of member zones

The list of member zones are specified as an unordered list (see Section 2.5.6) of domain names under the owner name "zones" where "zones" is a sub-domain of the catalog zone.

The names of member zones are represented on the RDATA side instead of as part of owner names so that all valid domain names may be represented regardless of their length. [RFC1035]

For example, if a catalog zone is named "catalog1.example.org." and lists 3 zones "example.com.", "example.net." and "example.org.", the RRs would appear as follows:


```
<hash>.zones.catalog1.example.org. 3600 IN PTR example.com.  
<hash>.zones.catalog1.example.org. 3600 IN PTR example.net.  
<hash>.zones.catalog1.example.org. 3600 IN PTR example.org.
```

2.8. Zone configuration properties

TBD: Prepare a list of zone configuration properties that are common to DNS implementations. This is so that a company may manage a catalog zone using a Windows DNS server as the primary, and a secondary nameserver hosting service may pick up the common properties and may use a different nameserver implementation such as BIND or NSD on a POSIX operating system to serve it.

TBD: We may specify that unrecognized zone property names must be ignored, or that nameserver specific properties must be specified using the "x-" prefix similar to MIME type naming.

TBD: Any list of zone properties is ideally maintained as a registry rather than within this memo.

2.8.1. zone-soa-default-serial

TBD.

2.8.2. zone-soa-default-refresh

TBD.

2.9. Zone properties specific to a member zone

Member zones in a catalog zone share template zone configuration that is common to all member zones in that catalog. This section describes the syntax that can be used to specify zone properties specific to single member zones.

Let N be an absolute name formed by concatenating the member zone name hash as a label (see Appendix A), the label "zones", and the catalog zone name in that order, such that N is a unique owner name in the catalog zone.

Zone properties specific to a particular member zone are specified under the respective sub-domain N.

For example, if a catalog zone is named "catalog1.example.org." and a member zone "example.com." contains a property "prop0" with string (see Section 2.5.1) value "Example", the corresponding RR would appear as follows:

```
prop0.<m-hash>.zones.catalog1.example.org. 3600 IN TXT "Example"
```

As another example, if a catalog zone is named "cat1.example.org." and a member zone "example.com." contains a property "prop2" with its value being an unordered list (see Section 2.5.6) of two domain names "a.example.com." and "b.example.com.", the corresponding RRs would appear as follows:

```
(<hash>.prop2.<m-hash>.zones.cat1.example.org.  
3600 IN PTR a.example.com.)  
(<hash>.prop2.<m-hash>.zones.cat1.example.org.  
3600 IN PTR b.example.com.)
```

2.10. Example of a catalog zone

```
$ORIGIN catalog.example.org.  
@           IN SOA  . . 1 3600 3600 86400 3600  
           IN NS   invalid.  
version     IN TXT  "1"  
(5960775ba382e7a4e09263fc06e7c00569b6a05c.zones  
IN PTR domain1.example.com.)
```

3. Nameserver behavior and requirements

3.1. General requirements

TBD: Explain nameserver behavior in a more detailed way here. It is under-specified.

As it is a regular DNS zone, a catalog zone can be transferred using DNS zone transfers among nameservers.

Although they are regular DNS zones, catalog zones contain only information for the management of a set of nameservers. For this reason, operators may want to limit the systems able to query these zones. It may be inconvenient to serve some contents of catalog zones via DNS queries anyway due to the nature of their representation. A separate method of querying entries inside the catalog zone may be made available by nameserver implementations (see Section 3.3).

Catalog updates should be automatic, i.e., when a nameserver that supports catalog zones completes a zone transfer for a catalog zone, it SHOULD apply changes to the catalog within the running nameserver automatically without any manual intervention.

As with regular zones, primary and secondary nameservers for a catalog zone may be operated by different administrators. The

secondary nameservers may be configured to synchronize catalog zones from the primary, but the primary's administrators may not have any administrative access to the secondaries.

A catalog zone can be updated via DNS UPDATE on a reference primary nameserver, or via zone transfers. Nameservers MAY allow loading and transfer of broken zones with incorrect catalog zone syntax (as they are treated as regular zones), but nameservers MUST NOT process such broken zones as catalog zones. For the purpose of catalog processing, the broken catalogs MUST be ignored. If a broken catalog zone was transferred, the newly transferred catalog zone MUST be ignored (but the older copy of the catalog zone SHOULD be left running subject to values in SOA fields).

If there is a clash between an existing member zone's name and an incoming member zone's name (via transfer or update), the new instance of the zone MUST be ignored and an error SHOULD be logged.

When zones are introduced into a catalog zone, a primary SHOULD first make the new zones available for transfers before making the updated catalog zone available for transfer, or sending NOTIFY for the catalog zone to secondaries. Note that secondary nameservers may attempt to transfer the catalog zone upon refresh timeout, so care must be taken to make the member zones available before any update to the list of member zones is visible in the catalog zone.

When zones are deleted from a catalog zone, a primary MAY delete the member zone immediately after notifying secondaries. It is up to the secondary nameserver to handle this condition correctly.

TBD: Transitive primary-secondary relationships

3.2. Updating catalog zones

TBD: Explain updating catalog zones using DNS UPDATE.

3.3. Implementation notes

Catalog zones on secondary nameservers would have to be setup manually, perhaps as static configuration, similar to how ordinary DNS zones are configured. Members of such catalog zones will be automatically synchronized by the secondary after the catalog zone is configured.

An administrator would want to look at data inside a catalog zone. Typical queries may include dumping the list of member zones, dumping a member zone's effective configuration, querying a specific property value of a member zone, etc. Because of the syntax of catalog zones,

it may not be possible to perform these queries intuitively, or in some cases, at all, using DNS QUERY. The list of member zones may not fit in a single DNS message. The set of present properties for a zone cannot be queried using a single DNS QUERY.

Implementations are advised to provide a tool that uses either the output of AXFR or an out-of-band method to perform queries on catalog zones.

4. Security considerations

As catalog zones are transmitted using DNS zone transfers, it is absolutely essential for these transfers to be protected from unexpected modifications on the route. So, it is a requirement that catalog zone transfers SHOULD be authenticated using TSIG [RFC2845]. A primary nameserver SHOULD NOT serve a catalog zone for transfer without using TSIG and a secondary nameserver SHOULD abandon an update to a catalog zone that was received without using TSIG.

DNS UPDATE [RFC2136] to catalog zones similarly SHOULD be authenticated using TSIG.

Zone transfers of member zones SHOULD similarly be authenticated using TSIG [RFC2845]. The TSIG shared secrets used for member zones MUST NOT be mentioned anywhere in the catalog zone data. However, key identifiers may be shared within catalog zones.

Catalog zones do not need to be signed using DNSSEC; their zone transfers being authenticated by TSIG. Signed zones MUST be handled normally by nameservers, and their contents MUST NOT be DNSSEC-validated.

5. IANA considerations

This document has no IANA actions.

6. Acknowledgements

Catalog zones originated as the chosen method among various proposals that were evaluated at ISC for easy zone management. The chosen method of storing the catalog as a regular DNS zone was proposed by Stephen Morris.

We later discovered that Paul Vixie's earlier [Metazones] proposal implemented a similar approach and reviewed it. Catalog zones borrows some syntax ideas from Metazones, as both share this scheme of representing the catalog as a regular DNS zone.

Thanks to Brian Conry, Evan Hunt, and Victoria Risk for reviewing draft proposals and providing support, comments and suggestions.

Thanks to BIND users who reviewed draft proposals and offered comments and suggestions.

7. References

7.1. Normative references

- [FIPS.180-4.2015]
National Institute of Standards and Technology, "Secure Hash Standard", FIPS PUB 180-4, August 2015,
<<http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf>>.
- [ISO.9899.1990]
International Organization for Standardization,
"Programming languages - C", ISO Standard 9899, 1990.
- [RFC1034] Mockapetris, P., "Domain names - concepts and facilities", STD 13, RFC 1034, DOI 10.17487/RFC1034, November 1987,
<<http://www.rfc-editor.org/info/rfc1034>>.
- [RFC1035] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, DOI 10.17487/RFC1035, November 1987, <<http://www.rfc-editor.org/info/rfc1035>>.
- [RFC1982] Elz, R. and R. Bush, "Serial Number Arithmetic", RFC 1982, DOI 10.17487/RFC1982, August 1996,
<<http://www.rfc-editor.org/info/rfc1982>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997,
<<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC2136] Vixie, P., Ed., Thomson, S., Rekhter, Y., and J. Bound, "Dynamic Updates in the Domain Name System (DNS UPDATE)", RFC 2136, DOI 10.17487/RFC2136, April 1997,
<<http://www.rfc-editor.org/info/rfc2136>>.
- [RFC2606] Eastlake 3rd, D. and A. Panitz, "Reserved Top Level DNS Names", BCP 32, RFC 2606, DOI 10.17487/RFC2606, June 1999,
<<http://www.rfc-editor.org/info/rfc2606>>.

- [RFC2845] Vixie, P., Gudmundsson, O., Eastlake 3rd, D., and B. Wellington, "Secret Key Transaction Authentication for DNS (TSIG)", RFC 2845, DOI 10.17487/RFC2845, May 2000, <<http://www.rfc-editor.org/info/rfc2845>>.
- [RFC3123] Koch, P., "A DNS RR Type for Lists of Address Prefixes (APL RR)", RFC 3123, DOI 10.17487/RFC3123, June 2001, <<http://www.rfc-editor.org/info/rfc3123>>.
- [RFC7719] Hoffman, P., Sullivan, A., and K. Fujiwara, "DNS Terminology", RFC 7719, DOI 10.17487/RFC7719, December 2015, <<http://www.rfc-editor.org/info/rfc7719>>.

7.2. Informative references

- [I-D.hunt-note-rr]
Hunt, E. and D. Mahoney, "A DNS Resource Record for Confidential Comments (NOTE RR)", draft-hunt-note-rr-01 (work in progress), May 2014.
- [Metazones]
Vixie, P., "Federated Domain Name Service Using DNS Metazones", 2005, <<http://ss.vix.su/~vixie/mz.pdf>>.
- [RPZ]
Vixie, P. and V. Schryver, "DNS Response Policy Zones (DNS RPZ)", 2010, <<http://ftp.isc.org/isc/dnsrpz/isc-tn-2010-1.txt>>.

Appendix A. Glossary

Catalog zone: A DNS zone containing a DNS catalog, that is, a list of DNS zones and associated template zone configuration common to all member zones.

Member zone: A DNS zone whose configuration is published inside a catalog zone.

Primary nameserver: An authoritative server configured to be the source of zone transfer to one or more [secondary] nameservers. Also see [RFC7719].

RDATA hash: The hexadecimal format 40-digit SHA-1 [FIPS.180-4.2015] digest, of the RDATA of the corresponding RR. For RDATA containing DNS names, no name compression must be in use, i.e., the name must be in its full expanded wire data format when it is hashed.

Member zone name hash: The hexadecimal format 40-digit SHA-1 [FIPS.180-4.2015] digest, of a zone name in uncompressed wire format.

Secondary nameserver: An authoritative server which uses zone transfer to retrieve the zone. Also see [RFC7719].

Zone property: A configuration parameter of a zone, sometimes also called a zone option.

Appendix B. Open issues and discussion (to be removed before final publication)

1. Config options

We want catalog zones to be adopted by multiple DNS implementations. Towards this, we have to generalize zone config options and adopt a minimal set that we can expect most implementations to support.

2. Catalog zone and member zones on different primary nameservers

Will it be possible to setup a catalog zone on one nameserver as primary, and allow its member zones to be served by different primary nameservers?

3. Transitive relationships

For a catalog zone, a secondary nameserver may be a primary nameserver to a different set of nameservers in a nameserver farm. In these transitive relationships, zone configuration options (such as also-notify and allow-transfer) may differ based on the location of the primary in the hierarchy. It may not be possible to specify this within a catalog zone.

4. Templates

Are support for config and zone data templates useful at this level? They would add complexity across implementations. If added, it would be better to restrict templates at the primary nameserver and let the secondary receive regular expanded zones.

5. Overriding controls

A way to override zone config options (as prescribed by the catalog zones) on secondary nameservers was requested. As this would be configured outside catalog zones, it may be better to leave this to implementations.

6. Use of hashing

Should use of hashing be completely removed, and replaced with the same common owner name for all property RRs in a collection? Both IXFR and DNS UPDATE allow changing individual RRs in a RRset.

7. Choice of hash function

Should a different faster hash function be chosen to replace SHA-1 when computing catalog member zone name hashes?

8. Overriding existing RR types

This memo currently overrides only the PTR RR TYPE's meaning as PTR is currently used for reverse lookups. But such overridden use seems like a non-issue as PTR is defined to be a pointer to any name in [RFC1035].

9. APL limits

APL can only support as many networks as can fit in its RDATA. Though a very large number of networks can be listed in a single RDATA field, it is still limited in size. Will this limitation become a problem for any users?

Appendix C. Change History (to be removed before final publication)

- o draft-muks-dnsop-dns-catalog-zones-00
Initial public draft.
- o draft-muks-dnsop-dns-catalog-zones-01
Added Witold, Ray as authors. Fixed typos, consistency issues. Fixed references. Updated Area. Removed newly introduced custom RR TYPES. Changed schema version to 1. Changed TSIG requirement from MUST to SHOULD. Removed restrictive language about use of DNS QUERY. When zones are introduced into a catalog zone, a primary SHOULD first make the new zones available for transfers first (instead of MUST). Updated examples, esp. use IPv6 in examples per Fred Baker. Add catalog zone example.

Authors' Addresses

Mukund Sivaraman
Internet Systems Consortium
950 Charter Street
Redwood City, CA 94063
US

Email: muks@mukund.org
URI: <http://www.isc.org/>

Stephen Morris
Internet Systems Consortium
950 Charter Street
Redwood City, CA 94063
US

Email: stephen@isc.org
URI: <http://www.isc.org/>

Ray Bellis
Internet Systems Consortium
950 Charter Street
Redwood City, CA 94063
US

Email: ray@isc.org
URI: <http://www.isc.org/>

Witold Krecicki
Internet Systems Consortium
950 Charter Street
Redwood City, CA 94063
US

Email: wpk@isc.org
URI: <http://www.isc.org/>

Internet Engineering Task Force
Internet-Draft
Intended status: Experimental
Expires: September 2, 2018

M. Sivaraman
S. Morris
R. Bellis
W. Krecicki
Internet Systems Consortium
March 1, 2018

DNS Catalog Zones
draft-muks-dnsop-dns-catalog-zones-04

Abstract

This document describes a method for automatic DNS zone provisioning among DNS primary and secondary nameservers by storing and transferring the catalog of zones to be provisioned as one or more regular DNS zones.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 2, 2018.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of

the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
2. Terminology	3
3. Description	3
4. Catalog Zone Structure	4
4.1. SOA and NS Records	4
4.2. Zone Data	4
4.2.1. Resource Record Format	5
4.2.2. Multi-valued Properties	5
4.2.3. Vendor-specific Properties	6
4.3. Zone Structure	6
4.3.1. List of Member Zones	6
4.3.2. Catalog Zone Schema Version	7
4.3.3. Default Zone Configuration	7
4.3.4. Zone Properties Specific to a Member Zone	7
5. Data Types	8
5.1. String	8
5.2. Booleans	8
5.3. Integers	8
5.4. Floating-Point Values	9
5.5. Domain Name	9
5.6. IP Prefix	9
5.7. Single Host Address	10
6. Nameserver Behavior	10
6.1. General Requirements	10
6.2. Updating Catalog Zones	11
6.3. Implementation Notes	11
7. Security Considerations	11
8. IANA Considerations	12
9. Acknowledgements	12
10. References	12
10.1. Normative references	12
10.2. Informative references	13
Appendix A. Open issues and discussion (to be removed before final publication)	14
Appendix B. Change History (to be removed before final publication)	14
Authors' Addresses	15

1. Introduction

The data in a DNS zone is synchronized amongst its primary and secondary nameservers using AXFR and IXFR. However, the list of zones served by the primary (called a catalog in [RFC1035]) is not

automatically synchronized with the secondaries. To add or remove a zone, the administrator of a DNS nameserver farm not only has to add or remove the zone from the primary, they must also add/remove the zone from all secondaries, either manually or via an external application. This can be both inconvenient and error-prone; it will also be dependent on the nameserver implementation.

This document describes a method in which the catalog is represented as a regular DNS zone (called a "catalog zone" here), and transferred using DNS zone transfers. As zones are added to or removed from the catalog zone, the changes are propagated to the secondary nameservers in the normal way. The secondary nameservers then add/remove/modify the zones they serve in accordance with the changes to the zone.

The contents and representation of catalog zones are described in Section 3. Nameserver behavior is described in Section 6.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

Catalog zone: A DNS zone containing a DNS catalog, that is, a list of DNS zones and associated zone configuration.

Member zone: A DNS zone whose configuration is published inside a catalog zone.

Zone property: A configuration parameter of a zone, sometimes also called a zone option, represented as a key/value pair.

\$CATZ: Used in examples as a placeholder to represent the domain name of the catalog zone itself (c.f. \$ORIGIN).

3. Description

A catalog zone is a specially crafted DNS zone that contains, as DNS zone data:

- o A list of DNS zones (called "member zones").
- o Default zone configuration information common to all member zones.
- o Zone-specific configuration information.

An implementation of catalog zones MAY allow the catalog to contain other catalog zones as member zones, but default zone configuration present in a catalog zone only applies to its immediate member zones.

Although the contents of a catalog zone are interpreted and acted upon by nameservers, a catalog zone is a regular DNS zone and so must adhere to the standards for such zones.

A catalog zone is primarily intended for the management of a farm of authoritative nameservers. It is not expected that the content of catalog zones will be accessible from any recursive nameserver.

4. Catalog Zone Structure

4.1. SOA and NS Records

As with any other DNS zone, a catalog zone MUST have a syntactically correct SOA record and one or more NS records at its apex.

The SOA record's SERIAL, REFRESH, RETRY and EXPIRE fields [RFC1035] are used during zone transfer. A catalog zone's SOA SERIAL field MUST increase when an update is made to the catalog zone's contents as per serial number arithmetic defined in [RFC1982]. Otherwise, secondary nameservers might not notice updates to the catalog zone's contents.

Should the zone be made available for querying, the SOA record's MINIMUM field's value is the negative cache time (as defined in [RFC2308]). Since recursive nameservers are not expected to be able to access (and subsequently cache) entries from a catalog zone a value of zero (0) is RECOMMENDED.

Since there is no requirement to be able to query the catalog zone via recursive nameservers the NS records at the apex will not be used and no parent delegation is required. However, they are still required so that catalog zones are syntactically correct DNS zones. Any valid DNS name can be used in the NSDNAME field of such NS records [RFC1035] and they MUST be ignored. A single NS RR with an NSDNAME field containing the absolute name "invalid." is RECOMMENDED [RFC2606].

4.2. Zone Data

A catalog zone contains a set of key/value pairs, where each key is encapsulated within the owner name of a DNS RR and the corresponding value is stored in the RR's RDATA. The specific owner name depends on whether the property relates to the catalog zone itself, a member

zone thereof, or to default zone properties described in Section 4.3. The owner names are case insensitive.

4.2.1. Resource Record Format

Each key/value pair has a defined data type, and each data type accordingly uses a particular RR TYPE to represent its possible values, as specified in Section 5.

The general form of a catalog zone record is as follows:

```
[<unique-id>.]<key>.<path>.$CATZ 0 IN <RRTYPE> <value>
```

where <path> is a sequence of labels with values depending on the purpose (and hence position) of the record within the catalog zone (see Section 4.3) and where the <unique-id> prefix is only present for multi-valued properties (see Section 4.2.2).

NB: Catalog zones use some RR TYPES (such as PTR) with alternate semantics to those originally defined for them. Although this may be controversial, the situation is similar to other similar zone-based representations such as response-policy zones [RPZ].

The CLASS field of every RR in a catalog zone MUST be IN (1). This is because some RR TYPES such as APL used by catalog zones are defined only for the IN class.

The TTL field's value is not specially defined by this memo. Catalog zones are for authoritative nameserver management only and are not intended for general querying via recursive resolvers and therefore a value of zero (0) is RECOMMENDED.

It is an error for any single owner name within a catalog zone (other than the apex of the zone itself) to have more than one RR associated with it.

4.2.2. Multi-valued Properties

Some properties do not represent single values but instead represent a collection of values. The specification for each property describes whether it is single-valued or multi-valued. A multi-valued property is encoded as multiple RRs where the owner name of each individual RR contains a unique (user specified) DNS label.

So, while a single-valued key might be represented like this:

```
<key>.<path>.$CATZ IN TXT "value"
```

a multi-valued key would be represented like this:

```
<unique-id-1>.<key>.<path>.$CATZ IN TXT "value 1"  
<unique-id-2>.<key>.<path>.$CATZ IN TXT "value 2"  
...
```

NB: a property that is specified to be multi-valued MUST be encoded using the unique prefixed key syntax even if there is only one value present.

The specification of any multi-valued property MUST document whether the collection represents either an ordered or un-ordered list. In the former case the ordering of the prefixes according to the usual DNS canonical name ordering will determine the sort order.

4.2.3. Vendor-specific Properties

TBD: Prepare a list of zone configuration properties that are common to DNS implementations. This is so that a company may manage a catalog zone using a Windows DNS server as the primary, and a secondary nameserver hosting service may pick up the common properties and may use a different nameserver implementation such as BIND or NSD on a POSIX operating system to serve it.

TBD: We may specify that unrecognized zone property names must be ignored, or that nameserver specific properties must be specified using the "x-" prefix similar to MIME type naming.

TBD: Any list of zone properties is ideally maintained as a registry rather than within this memo.

4.3. Zone Structure

4.3.1. List of Member Zones

The list of member zones is specified as a multi-valued collection of domain names under the owner name "zones" where "zones" is a direct child domain of the catalog zone.

The names of member zones are represented on the RDATA side (instead of as a part of owner names) so that all valid domain names may be represented regardless of their length [RFC1035].

For example, if a catalog zone lists three zones "example.com.", "example.net." and "example.org.", the RRs would appear as follows:

```
<m-unique-1>.zones.$CATZ 0 IN PTR example.com.  
<m-unique-2>.zones.$CATZ 0 IN PTR example.net.  
<m-unique-3>.zones.$CATZ 0 IN PTR example.org.
```

where <m-unique-N> is a label that uniquely tags each record in the collection, as described in Section 4.2.2.

Although any legal label could be used for <m-unique-N> it is RECOMMENDED that it be a value deterministically derived from the fully-qualified member zone name. The BIND9 implementation uses the 40 character hexadecimal representation of the SHA-1 digest [FIPS.180-4.2015] of the lower-cased member zone name as encoded in uncompressed wire format.

4.3.2. Catalog Zone Schema Version

The catalog zone schema version is specified by an unsigned integer property with the property name "version". All catalog zones MUST have this property present. Primary and secondary nameservers MUST NOT use catalog zones with an unexpected value in this property, but they may be transferred as ordinary zones. For this memo, the "version" property value MUST be set to 2, i.e.

```
version.$CATZ 0 IN TXT "2"
```

NB: Version 1 was used in a draft version of this memo and reflected the implementation first found in BIND 9.11.

4.3.3. Default Zone Configuration

Default zone configuration comprises a set of properties that are applied to all member zones listed in the catalog zone unless overridden by member zone-specific information.

All such properties are stored as child nodes of the owner name "defaults" itself a direct child node of the catalog zone, e.g.:

```
example-prop.defaults.$CATZ 0 IN TXT "Example"
```

4.3.4. Zone Properties Specific to a Member Zone

Default zone properties can be overridden on a per-zone basis by specifying the property under the the sub-domain associated with the member zone in the list of zones, e.g.:

```
example-prop.<m-unique>.zones.$CATZ 0 IN TXT "Example"
```


where "m-unique" is the label that uniquely identifies the member zone name as described in Section 4.3.1.

NB: when a zone-specific property is multi-valued the owner name will contain two unique identifiers, the left-most tagging being associated with the individual value (<unique-id-N>) and the other (<m-unique>) associated with the member zone itself, e.g.:

```
$ORIGIN <m-unique>.zones.$CATZ
<unique-id-1>.example-prop 0 IN TXT "Value 1"
<unique-id-2>.example-prop 0 IN TXT "Value 2"
...
```

5. Data Types

This section lists the various data types defined for use within catalog zones.

5.1. String

A key with a string value is represented with a TXT RR [RFC1035], e.g.:

```
example-prop.<m-unique>.zones.$CATZ 0 IN TXT "Example"
```

If the RDATA is split into multiple <character-string> elements the MUST be directly concatenated without any separating character.

5.2. Booleans

A key with a boolean value is represented with a TXT RR containing a single <character-string> with a value of "true" for true condition and "false" for false condition, e.g:

```
example-prop.<m-unique>.zones.$CATZ 0 IN TXT "false"
```

The RDATA is case-insensitive.

5.3. Integers

A key with an integer value is specified using a TXT RR containing a single <character-string>.

A signed integer's TXT RDATA uses the representation of an unsuffixed "integer constant" as defined in the C programming language standard [ISO.9899.1990] (of the type matching a 64-bit signed integer on that platform), with an optional minus prefix.

An unsigned integer's TXT RDATA uses the representation of an unsuffixed "integer constant" as defined in the C programming language standard [ISO.9899.1990] (of the type matching a 64-bit unsigned integer on that platform).

For example, a property with an unsigned integer value of 300 would appear as follows:

```
example-prop.<m-unique>.zones.$CATZ 0 IN TXT "300"
```

5.4. Floating-Point Values

A key with a floating-point value is specified using a TXT RR containing a single <character-string>.

A floating-point value's TXT RDATA uses the representation of an unsuffixed "floating constant" as defined in the C programming language standard [ISO.9899.1990].

For example, a property with an unsigned integer value of 0.15 may appear as follows:

```
example-prop.<m-unique>.zones.$CATZ 0 IN TXT "15e-2"
```

5.5. Domain Name

A key whose value is a domain name is specified using a PTR RR [RFC1035], e.g.:

```
example-prop.defaults.$CATZ 0 IN PTR ns1.example.com.
```

5.6. IP Prefix

A property whose value is an IP network prefix is specified using an APL RR [RFC3123]. The negation flag ("!" in presentation format) may be used to indicate all addresses not included within that prefix, e.g. for use in Access Control Lists, e.g.:

Although a single APL record is capable of containing multiple prefixes, for consistency of representation lists of prefixes MUST use the multi-valued property syntax as documented in Section 4.2.2, e.g.:

```
$ORIGIN <m-unique>.zones.$CATZ
<unique-id-1>.example-prop 0 IN APL ( 1:192.0.2.0/24 )
<unique-id-2>.example-prop 0 IN APL ( !1:0.0.0.0/0 )
```

Implementations MUST accept only the first prefix within each APL record and MUST ignore any subsequent prefixes found therein.

5.7. Single Host Address

A single host address is represented using either an A or AAAA record as appropriate, e.g.:

```
example-prop1.<m-unique>.zones.$CATZ 0 IN A 192.0.2.1
example-prop2.<m-unique>.zones.$CATZ 0 IN AAAA 2001:db8::1
```

6. Nameserver Behavior

6.1. General Requirements

As it is a regular DNS zone, a catalog zone can be transferred using DNS zone transfers among nameservers.

Although they are regular DNS zones, catalog zones contain only information for the management of a set of authoritative nameservers. For this reason, operators may want to limit the systems able to query these zones. It may be inconvenient to serve some contents of catalog zones via DNS queries anyway due to the nature of their representation. A separate method of querying entries inside the catalog zone may be made available by nameserver implementations (see Section 6.3).

Catalog updates should be automatic, i.e., when a nameserver that supports catalog zones completes a zone transfer for a catalog zone, it SHOULD apply changes to the catalog within the running nameserver automatically without any manual intervention.

As with regular zones, primary and secondary nameservers for a catalog zone may be operated by different administrators. The secondary nameservers may be configured to synchronize catalog zones from the primary, but the primary's administrators may not have any administrative access to the secondaries.

A catalog zone can be updated via DNS UPDATE on a reference primary nameserver, or via zone transfers. Nameservers MAY allow loading and transfer of broken zones with incorrect catalog zone syntax (as they are treated as regular zones), but nameservers MUST NOT process such broken zones as catalog zones. For the purpose of catalog processing, the broken catalogs MUST be ignored. If a broken catalog zone was transferred, the newly transferred catalog zone MUST be ignored (but the older copy of the catalog zone SHOULD be left running subject to values in SOA fields).

If there is a clash between an existing member zone's name and an incoming member zone's name (via transfer or update), the new instance of the zone MUST be ignored and an error SHOULD be logged.

When zones are introduced into a catalog zone, a primary SHOULD first make the new zones available for transfers before making the updated catalog zone available for transfer, or sending NOTIFY for the catalog zone to secondaries. Note that secondary nameservers may attempt to transfer the catalog zone upon refresh timeout, so care must be taken to make the member zones available before any update to the list of member zones is visible in the catalog zone.

When zones are deleted from a catalog zone, a primary MAY delete the member zone immediately after notifying secondaries. It is up to the secondary nameserver to handle this condition correctly.

TBD: Transitive primary-secondary relationships

6.2. Updating Catalog Zones

TBD: Explain updating catalog zones using DNS UPDATE.

6.3. Implementation Notes

Catalog zones on secondary nameservers would have to be setup manually, perhaps as static configuration, similar to how ordinary DNS zones are configured. Members of such catalog zones will be automatically synchronized by the secondary after the catalog zone is configured.

An administrator may want to look at data inside a catalog zone. Typical queries might include dumping the list of member zones, dumping a member zone's effective configuration, querying a specific property value of a member zone, etc. Because of the structure of catalog zones, it may not be possible to perform these queries intuitively, or in some cases, at all, using DNS QUERY. For example it is not possible to enumerate the contents of a multi-valued property (such as the list of member zones) with a single QUERY. Implementations are therefore advised to provide a tool that uses either the output of AXFR or an out-of-band method to perform queries on catalog zones.

7. Security Considerations

As catalog zones are transmitted using DNS zone transfers, it is absolutely essential for these transfers to be protected from unexpected modifications on the route. So, catalog zone transfers SHOULD be authenticated using TSIG [RFC2845]. A primary nameserver

SHOULD NOT serve a catalog zone for transfer without using TSIG and a secondary nameserver SHOULD abandon an update to a catalog zone that was received without using TSIG.

Use of DNS UPDATE [RFC2136] to modify the content of catalog zones SHOULD similarly be authenticated using TSIG.

Zone transfers of member zones SHOULD similarly be authenticated using TSIG [RFC2845]. The TSIG shared secrets used for member zones MUST NOT be mentioned anywhere in the catalog zone data. However, key identifiers may be shared within catalog zones.

Catalog zones do not need to be signed using DNSSEC, their zone transfers being authenticated by TSIG. Signed zones MUST be handled normally by nameservers, and their contents MUST NOT be DNSSEC-validated.

8. IANA Considerations

This document has no IANA actions.

9. Acknowledgements

Catalog zones originated as the chosen method among various proposals that were evaluated at ISC for easy zone management. The chosen method of storing the catalog as a regular DNS zone was proposed by Stephen Morris.

We later discovered that Paul Vixie's earlier [Metazones] proposal implemented a similar approach and reviewed it. Catalog zones borrows some syntax ideas from Metazones, as both share this scheme of representing the catalog as a regular DNS zone.

Thanks to Brian Conry, Tony Finch, Evan Hunt, Patrik Lundin, Victoria Risk and Carsten Strettmann for reviewing draft proposals and offering comments and suggestions.

10. References

10.1. Normative references

[FIPS.180-4.2015]
National Institute of Standards and Technology, "Secure Hash Standard", FIPS PUB 180-4, August 2015,
<<http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf>>.

- [ISO.9899.1990]
International Organization for Standardization,
"Programming languages - C", ISO Standard 9899, 1990.
- [RFC1035] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, DOI 10.17487/RFC1035, November 1987, <<https://www.rfc-editor.org/info/rfc1035>>.
- [RFC1982] Elz, R. and R. Bush, "Serial Number Arithmetic", RFC 1982, DOI 10.17487/RFC1982, August 1996, <<https://www.rfc-editor.org/info/rfc1982>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC2136] Vixie, P., Ed., Thomson, S., Rekhter, Y., and J. Bound, "Dynamic Updates in the Domain Name System (DNS UPDATE)", RFC 2136, DOI 10.17487/RFC2136, April 1997, <<https://www.rfc-editor.org/info/rfc2136>>.
- [RFC2308] Andrews, M., "Negative Caching of DNS Queries (DNS NCACHE)", RFC 2308, DOI 10.17487/RFC2308, March 1998, <<https://www.rfc-editor.org/info/rfc2308>>.
- [RFC2606] Eastlake 3rd, D. and A. Panitz, "Reserved Top Level DNS Names", BCP 32, RFC 2606, DOI 10.17487/RFC2606, June 1999, <<https://www.rfc-editor.org/info/rfc2606>>.
- [RFC2845] Vixie, P., Gudmundsson, O., Eastlake 3rd, D., and B. Wellington, "Secret Key Transaction Authentication for DNS (TSIG)", RFC 2845, DOI 10.17487/RFC2845, May 2000, <<https://www.rfc-editor.org/info/rfc2845>>.
- [RFC3123] Koch, P., "A DNS RR Type for Lists of Address Prefixes (APL RR)", RFC 3123, DOI 10.17487/RFC3123, June 2001, <<https://www.rfc-editor.org/info/rfc3123>>.

10.2. Informative references

- [Metazones]
Vixie, P., "Federated Domain Name Service Using DNS Metazones", 2005, <<http://ss.vix.su/~vixie/mz.pdf>>.
- [RPZ]
Vixie, P. and V. Schryver, "DNS Response Policy Zones (DNS RPZ)", 2010, <<http://ftp.isc.org/isc/dnsrpz/isc-tn-2010-1.txt>>.

Appendix A. Open issues and discussion (to be removed before final publication)

1. Config options

We want catalog zones to be adopted by multiple DNS implementations. Towards this, we have to generalize zone config options and adopt a minimal set that we can expect most implementations to support.

2. Catalog zone and member zones on different primary nameservers

Will it be possible to setup a catalog zone on one nameserver as primary, and allow its member zones to be served by different primary nameservers?

3. Transitive relationships

For a catalog zone, a secondary nameserver may be a primary nameserver to a different set of nameservers in a nameserver farm. In these transitive relationships, zone configuration options (such as also-notify and allow-transfer) may differ based on the location of the primary in the hierarchy. It may not be possible to specify this within a catalog zone.

4. Overriding controls

A way to override zone config options (as prescribed by the catalog zones) on secondary nameservers was requested. As this would be configured outside catalog zones, it may be better to leave this to implementations.

Appendix B. Change History (to be removed before final publication)

- o draft-muks-dnsop-dns-catalog-zones-00
Initial public draft.
- o draft-muks-dnsop-dns-catalog-zones-01
Added Witold, Ray as authors. Fixed typos, consistency issues. Fixed references. Updated Area. Removed newly introduced custom RR TYPES. Changed schema version to 1. Changed TSIG requirement from MUST to SHOULD. Removed restrictive language about use of DNS QUERY. When zones are introduced into a catalog zone, a primary SHOULD first make the new zones available for transfers first (instead of MUST). Updated examples, esp. use IPv6 in examples per Fred Baker. Add catalog zone example.
- o draft-muks-dnsop-dns-catalog-zones-02

Addressed some review comments by Patrik Lundin.

- o draft-muks-dnsop-dns-catalog-zones-03
Revision bump.
- o draft-muks-dnsop-dns-catalog-zones-04
Reordering of sections into more logical order.
Separation of multi-valued properties into their own category.

Authors' Addresses

Mukund Sivaraman
Internet Systems Consortium
950 Charter Street
Redwood City, CA 94063
US

Email: muks@mukund.org
URI: <http://www.isc.org/>

Stephen Morris
Internet Systems Consortium
950 Charter Street
Redwood City, CA 94063
US

Email: stephen@isc.org
URI: <http://www.isc.org/>

Ray Bellis
Internet Systems Consortium
950 Charter Street
Redwood City, CA 94063
US

Email: ray@isc.org
URI: <http://www.isc.org/>

Witold Krecicki
Internet Systems Consortium
950 Charter Street
Redwood City, CA 94063
US

Email: wpk@isc.org
URI: <http://www.isc.org/>

Network Working Group
Internet-Draft
Updates: 6781 (if approved)
Intended status: Informational
Expires: March 17, 2018

M. Pounsett
Rightside Group, Ltd.
September 13, 2017

Change of Operator Procedures for Automatically Published DNSSEC Zones
draft-pounsett-transferring-automated-dnssec-zones-03

Abstract

Section 4.3.5.1 of [RFC6781] "DNSSEC Operational Practices, version 2" describes a procedure for transitioning a DNSSEC signed zone from one (cooperative) operator to another. The procedure works well in many situations, but makes the assumption that it is feasible for the two operators to simultaneously publish slightly different versions of the zone being transferred. In some cases, such as with TLD registries, operational considerations require both operators to publish identical versions of the zone for the duration of the transition. This document describes a modified transition procedure which can be used in these cases.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on March 17, 2018.

Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of

publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
2. Applicability	3
3. Changing Between Cooperating DNS Operators	3
3.1. Assumptions	3
3.2. Procedure Overview	3
3.3. Procedure Notation	3
3.4. The Procedure	4
4. Security Considerations	6
5. IANA Considerations	6
6. Normative References	6
Appendix A. Document Source	7
Appendix B. Changelist	7
B.1. Version pounsett-03	7
B.2. Version pounsett-02	7
B.3. Version pounsett-01	7
B.4. Version pounsett-00	8
Author's Address	8

1. Introduction

The process described in "DNSSEC Operational Practices, version 2" ([RFC6781]), section 4.3.5.1 for cooperating DNS operators to move a DNSSEC signed zone cannot be followed in all cases. When operators are moving a zone that is automatically published and/or changes rapidly, such as with a TLD or any other zone published from a registration database, it may not be feasible for the operators to publish different versions of the same zone.

In these cases, it would be necessary for one or both operators to have the capability to add, remove, or alter arbitrary records inline along the zone transfer path (such as modifying the NSSet, and stripping RRSIGs). It cannot be assumed that this capability exists, since few (if any) common DNS implementations include these functions, and many custom implementations exist whose feature sets cannot be predicted.

As a result, it must be assumed that operators moving an automatically generated or frequently updated zone must be able to

publish an identical zone while transitioning it from one operator to another.

2. Applicability

This document is intended for operators intending to transfer operational responsibility for DNSSEC-signed zones, while publishing consistent zone data on both sets of name servers throughout the transition.

Other procedures exist for operators who are unable to consistently replicate data between both sets of name servers (e.g. through zone transfer) or who do not require this level of zone consistency during the transfer.

3. Changing Between Cooperating DNS Operators

3.1. Assumptions

In this scenario, it is assumed that the operators will not exchange any private key material, but are otherwise fully cooperative. It is also assumed that the zone publishing process will be transferred between operators independently of the DNS operations. The simplest case is to transition the publishing process after the DNS operations move has been completed, and is the order that is assumed in this document, although the reverse order is possible. During the transition, the losing operator will provide the zone contents to the gaining operator by some automatic means (typically zone transfer). The transition of the publishing process is out of scope of this document.

3.2. Procedure Overview

The DNS operations transition uses a modified pre-publish KSK and ZSK rollover, whereby the losing operator pre-publishes the public KSK and ZSK of the gaining operator. Part way through the transition, the losing operator stops signing the zone and begins providing an unsecure zone to the gaining operator, who begins signing. Once that is done, the gaining operator continues to post-publish the public keys of the losing operator until the TTLs of the original RRSIGs expire.

3.3. Procedure Notation

In the timeline below, the losing operator is operator A, and the gaining operator is operator B. Records representing data generated by each operator are appended with the operator letter. DNSKEY records are appended with the keytype; DNSKEY_Z is a ZSK, and

DNSKEY_K is a KSK. RRSIGs are appended with the DNSKEY type they were generated from, as well as the operator who generated them; RRSIGs include in parentheses the RRSET type that they sign. For example, RRSIG_Z_A(SOA) is the RRSIG generated with DNSKEY_Z_A and signs the SOA record.

3.4. The Procedure

The KSK and ZSK rollover from the losing to gaining operator keys involves six stages as described in Figure 1 and Figure 2.

Initial

The initial version of the zone. This zone is published and signed by the losing operator. During this stage the gaining operator should begin serving the zone on its name servers.

Pre-Publish

Operator A begins pre-publishing DNSKEY_K_B and DNSKEY_Z_B. At the same time, the NS set at the apex of the zone is changed to NS_B to begin moving traffic to the gaining name servers. The operators MUST wait at least the time it takes for the data to propagate to all authoritative servers plus the TTL of the DNSKEY set before the Signing Migration step can be begun.

Re-Delegation

The parent changes the parent NS set to be NS_B, to match the NS set at the apex of the child zone. The parent also begins publishing the DS record for the gaining operator. The operators MUST wait at least the time it takes for the data to propagate to all the parent servers plus the TTL of the DS set before the Signing Migration step can be begun.

initial	pre-publish	re-delegation
Parent: NS_A DS_A	Parent: NS_A DS_A	Parent: NS_B DS_A DS_B
Child: Published by A Signed by A SOA_A0 RRSIG_Z_A(SOA) NS_A RRSIG_Z_A(NS) DNSKEY_Z_A DNSKEY_K_A RRSIG_K_A(DNSKEY)	Child: Published by A Signed by A SOA_A1 RRSIG_Z_A(SOA) NS_B RRSIG_Z_A(NS) DNSKEY_Z_A DNSKEY_Z_B DNSKEY_K_A DNSKEY_K_B RRSIG_K_A(DNSKEY)	Child: Published by A Signed by A SOA_A1 RRSIG_Z_A(SOA) NS_B RRSIG_Z_A(NS) DNSKEY_Z_A DNSKEY_Z_B DNSKEY_K_A DNSKEY_K_B RRSIG_K_A(DNSKEY)

Figure 1: Rollover for Cooperating Operators, Steps 1-3

Signing Migration

Once the new (pre-published) DNSKEY records and DS_B are in the caches of validating clients the operators can swap signing duties; DNSKEY_K_B and DNSKEY_Z_B become the active signing keys. When the gaining operator begins signing the zone, either they MUST strip the RRSIGs of the losing operator, or the losing operator MUST begin producing an unsigned zone. The operators MUST wait at least the largest TTL of any RRSIG in the zone before moving on to the Old DS Removal step.

Old DS Removal

The parent now removes the old DS_A from its zone. The operators MUST wait at least the TTL of the DS set before moving on to the Post Migration step.

Post Migration

After the old DS set has expired from caches the gaining operator may remove the losing operator DNSKEYs from the zone. At this time it is assumed that the gaining operator stops whatever

process was transferring zone data from the losing operator, and the gaining operator begins publishing the zone.

signing migration	old DS removal	post migration
Parent:	Parent:	Parent:
NS_B	NS_B	NS_B
DS_A		
DS_B	DS_B	DS_B
Child:	Child:	Child:
Published by A	Published by A	Published by B
Signed by B	Signed by B	Signed by B
SOA_A2	SOA_A2	SOA_B0
RRSIG_Z_B(SOA)	RRSIG_Z_B(SOA)	RRSIG_Z_B(SOA)
NS_B	NS_B	NS_B
RRSIG_Z_B(NS)	RRSIG_Z_B(NS)	RRSIG_Z_B(NS)
DNSKEY_Z_A	DNSKEY_Z_A	
DNSKEY_Z_B	DNSKEY_Z_B	DNSKEY_Z_B
DNSKEY_K_A	DNSKEY_K_A	
DNSKEY_K_B	DNSKEY_K_B	DNSKEY_K_B
RRSIG_K_B(DNSKEY)	RRSIG_K_B(DNSKEY)	RRSIG_K_B(DNSKEY)

Figure 2: Rollover for Cooperating Operators, Steps 4-6

4. Security Considerations

This document raises no new security considerations. Please see Section 6 of [RFC6781].

5. IANA Considerations

This document has no actions for IANA.

6. Normative References

- [RFC6781] Kolkman, O., Mekking, W., and R. Gieben, "DNSSEC Operational Practices, Version 2", RFC 6781, DOI 10.17487/RFC6781, December 2012, <<https://www.rfc-editor.org/info/rfc6781>>.

Appendix A. Document Source

[RFC Editor: Please remove this section before publication.]

This document is maintained at Github at
<<https://github.com/mpounsett/operator-transfer>>. Issue reports and
pull requests are gratefully accepted here.

The XML and TXT versions of this document are generated from Markdown
using mmark by Miek Gieben. mmark is available at
<<https://github.com/miekg/mmark>>.

Appendix B. Changelist

[RFC Editor: Please remove this section before publication.]

B.1. Version pounsett-03

- o grammar and spelling nits
- o fix RRSIG/DNSKEY typo in "Signing Migration" step

B.2. Version pounsett-02

- o grammar and spelling nits
- o separate procedural section into subsections
- o improve notation explanation
- o rename procedure steps for clarity
- o add prose description of operator key roll

B.3. Version pounsett-01

- o grammar and spelling nits
- o added Security Considerations section
- o added IANA Considerations section
- o added Document Source and Changelist
- o call out publishing migration as out of scope

B.4. Version pounsett-00

Initial Submission

Author's Address

Matthew Pounsett
Rightside Group, Ltd.
Toronto, ON
Canada

Email: matt@conundrum.com

DNSOP
Internet-Draft
Intended status: Best Current Practice
Expires: April 29, 2017

P. Wallstrom
J. Schlyter
Kirei AB
October 26, 2016

DNS Delegation Requirements
draft-wallstrom-dnsop-dns-delegation-requirements-03

Abstract

This document outlines a set of requirements on a well-behaved DNS delegation of a domain name. A large number of tools have been developed to test DNS delegations, but each tool uses a different set of requirements for what is a correct setup for a delegated domain name. However, there are few requirements on how to set up DNS in order to just make the delegation work. In order to have a well-behaved delegation that is robust to failures and also makes DNS resolvers behave consistently, there are a large number of things to consider.

Based on this document, it should be possible to set up a fully functional DNS delegation for a domain name, but also to create a set of test specifications for how to test a DNS delegation.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on April 29, 2017.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
1.1. DNS Terminology	4
1.2. Reserved Words	4
2. Basic requirements	5
2.1. The domain name MUST be valid	5
2.2. The domain MUST have a parent domain	5
2.3. The domain MUST have at least one working name server . .	5
3. Address requirements	5
3.1. Name server address MUST be globally routable	5
3.2. The IP address of a name server MUST be delegated by IANA	6
4. Connectivity requirements	6
4.1. All name servers MUST have UDP connectivity over port 53	7
4.2. All name servers MUST have TCP connectivity over port 53	7
5. Name server requirements	7
5.1. Authoritative name servers SHOULD NOT be recursive . . .	7
5.2. Name servers SHOULD support EDNS0	7
5.3. Name servers MUST process QNAME case insensitive	8
6. Consistency requirements	8
6.1. All name servers SHOULD respond with the same SOA serial number	8
6.2. All name servers SHOULD respond with the same SOA RNAME .	9
6.3. All name servers SHOULD respond with the same SOA parameters	9
6.4. All name servers MUST respond with the same NS RR Set . .	9
7. Delegation requirements	9
7.1. The delegation SHOULD contain at least two name servers .	9
7.2. The NS RR set in the parent SHOULD be a subset of the NS RR set in the child	10
7.3. The name servers SHOULD have network path diversity . . .	10
7.4. The name servers MUST have distinct IP addresses	10
7.5. The referral SHOULD fit into a non-truncated 512 byte UDP packet	10
7.6. All name servers MUST be authoritative for the domain name	11
7.7. The delegation name MUST exactly match the apex of the child zone	11

7.8.	Glue records in delegation SHOULD exactly match records in child zone	11
7.9.	SOA MNAME SHOULD be authoritative for the zone	11
8.	DNSSEC requirements	12
8.1.	The DS Digest Type MUST be assigned by IANA	12
8.2.	The DNSKEY algorithm MUST be assigned by IANA	12
8.3.	The chain of trust for the delegation MUST be valid	12
8.4.	One DS MUST match a least one DNSKEY in the child zone	12
8.5.	The number of NSEC3 iterations must not be higher than what is allowed	13
8.6.	RRSIG validity period SHOULD NOT be too short nor too long	13
8.7.	The name server MUST include RRSIG in all responses to DNSSEC queries	13
8.8.	The name servers MUST include valid NSEC/NSEC3 record in NXDOMAIN responses	13
9.	Syntax requirements	14
9.1.	Illegal characters MUST NOT be in the domain name	14
9.2.	Hyphens SHOULD NOT be in position 3 and 4 of the domain name	14
9.3.	The NS names MUST be valid hostnames	14
9.4.	The NS names MUST NOT be an alias	14
9.5.	The SOA RNAME MUST not contain the '@' character	14
9.6.	The SOA RNAME MUST be a legal hostname	15
9.7.	The SOA MNAME MUST be a legal hostname	15
9.8.	The MX record in apex MUST point to a valid hostname	15
10.	Security Considerations	15
11.	IANA Considerations	15
12.	Acknowledgements	15
13.	References	16
13.1.	Normative References	16
13.2.	Informative References	17
	Authors' Addresses	20

1. Introduction

This document outlines a set of requirements on a well-behaved DNS delegation of a domain name. Many domain name registries use a set of requirements on what they may consider a valid delegation. Such requirements can be used to implement tools that are used for pre- or post-delegation checks of the delegations in that registry.

To test the quality of the delegation there has been a number of different tools developed, each based on a different set of requirements. This document outlines a set of baseline requirements on a correct setup for a delegated domain name. This document is based on current RFCs and documents requirements that are protocol

specific, but also administrative policy requirements drawn from best practices and recommendations.

The DNS requirements are split into these different areas, to easier differentiate between what they are for:

- o Basic
- o Address
- o Connectivity
- o Name server
- o Consistency
- o Delegation
- o DNSSEC
- o Syntax

A secondary name server operator should follow the advice in the BCP document [RFC2182].

Nothing in this document precludes others testing servers for protocol compliance. DNS operators should test their servers to ensure that their vendors have shipped protocol compliant products. Name server vendors can use these tests as a part of this release processes. Registrants can use these tests to check their DNS operators servers.

1.1. DNS Terminology

This document attempts to fully follow the DNS terminology as defined in [RFC7719].

Many requirements in this document deal with the properties of a name server that is used as part of a delegation, therefore the wording mentioning the use - authoritative or recursive - of a name server as part of this is omitted.

1.2. Reserved Words

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

2. Basic requirements

The Basic requirements are fundamental to a working DNS delegation. Without these properties, the rest of the requirements are irrelevant.

2.1. The domain name MUST be valid

The domain name MUST follow the rules defined in Section 2.1 of [RFC1123] in order to be able to map the domain into a DNS packet. A domain name is normally valid if the name has been registered with a domain name registry.

Internationalized domain names, [RFC5891], are expected to be encoded using Punycode [RFC3492], thus following the rules outlined in Section 2.3.1 of [RFC1035]. Any validation of the domain name in U-label form is out of scope for this document.

2.2. The domain MUST have a parent domain

A DNS delegation MUST have a parent domain from which it is delegated. The concept of zone cuts was first described in [RFC1034] and later clarified in Section 6 of [RFC2182]. The only exception is the root zone, which do not have a parent zone.

2.3. The domain MUST have at least one working name server

A fully working DNS delegation has a parent zone delegating the zone to a set of child name servers. At least one name server MUST be able to answer DNS queries in order to be able to authoritatively serve data for the child zone.

3. Address requirements

A delegation in the public Internet DNS hierarchy will use the globally unique address space.

3.1. Name server address MUST be globally routable

In order for the domain and its resources to be accessible from the Internet, authoritative name servers must have addresses in the routable public addressing space.

IANA is responsible for global coordination of the IP addressing system. Aside its address allocation activities, it maintains reserved address ranges for special uses. There are two IANA registries for Special-Purpose Addresses, the IANA IPv6 Special-

Purpose Address Registry and the IANA IPv4 Special-Purpose Address Registry.

[RFC6890] instructs IANA on how to structure the IPv4 and IPv6 Special-Purpose Address Registries. The registries [IANA-IPv4-Special] and [IANA-IPv6-Special] are maintained by IANA, and are also described in Section 2.2 and 2.3 of [RFC7249].

A name server MUST NOT be using any IP address within any of these registries that are marked with False in the Global column.

3.2. The IP address of a name server MUST be delegated by IANA

IP addresses not delegated by IANA MUST NOT be used by a name server. Thus, IP addresses within a prefix not delegated to a RIR by IANA MUST be rejected.

The IANA registry [IANA-IPv6-Unicast] SHOULD be used to determine the status of an IPv6 prefix. Only prefixes with the status ALLOCATED are allowed.

The IANA registry [IANA-IPv4-Registry] SHOULD be used to determine the status of an IPv4 prefix. Only prefixes with the status ALLOCATED and LEGACY are allowed. Note that IPv4 LEGACY is not allocated to a RIR.

Martians [RFC1208] is a humorous term applied to packets that turn up unexpectedly on the wrong network because of bogus routing entries. Bogons [RFC3871] are packets sourced from addresses that have not yet been allocated by IANA or a RIR, or not delegated to a RIR by IANA as described above. Martians and Bogons SHOULD NOT be used as an address used by a name server.

4. Connectivity requirements

The use of underlying protocols for DNS is described in Section 4.2 of [RFC1035].

The Internet supports name server access using TCP on server port 53 (decimal) as well as datagram access using UDP on UDP port 53 (decimal). Today DNS is used in conjunction with both IPv4 and IPv6,

Name servers configured for a zone in a delegation MUST be able to answer queries using the DNS protocol.

4.1. All name servers MUST have UDP connectivity over port 53

DNS queries are sent using UDP on port 53, as described in Section 4.2.1 of [RFC1035]. A name server MUST respond to DNS queries over UDP for each IP address configured for that name server.

4.2. All name servers MUST have TCP connectivity over port 53

In addition to UDP, DNS queries can also be sent using TCP on port 53, as described in Section 4.2.2 of [RFC1035]. A name server MUST respond to DNS queries over TCP for each IP address configured for that name server. This requirement has also been further clarified in [RFC7766], which makes TCP a REQUIRED part of a full DNS protocol implementation.

It should be noted that even though [RFC7766] requires TCP for a DNS protocol implementation, it does not make specific recommendations to operators of DNS servers. However, it also notes that failure to support TCP (or the blocking of DNS over TCP at the network layer) may result in resolution failure and/or application-level timeouts. The operational requirements on DNS Transport over TCP are further discussed [I-D.kristoff-dnsop-dns-tcp-requirements].

5. Name server requirements

5.1. Authoritative name servers SHOULD NOT be recursive

To ensure consistency in DNS, an authoritative name server SHOULD NOT be configured to do recursive lookups. Also, open recursive resolvers are considered bad Internet practice due to their capability of assisting in large scale DDoS attacks. The introduction to [RFC5358] elaborates on mixing recursor and authoritative functionality. Section 2.5 of [RFC2870] have very specific requirement on disabling recursion functionality on root name servers.

5.2. Name servers SHOULD support EDNS0

EDNS0 is a mechanism to announce capabilities of a DNS implementation, and is now basically required by any new functionality in DNS such as DNSSEC. Initially standardized in [RFC2671] and later updated by [RFC6891], EDNS0 is a mechanism to announce capabilities of a DNS implementation.

5.3. Name servers MUST process QNAME case insensitive

The DNS standards require that name servers treat names with case insensitivity. That is, the names `example.com` and `EXAMPLE.COM` should resolve to the same IP address. However, in the response, most name servers echo back the name as it appeared in the request, preserving the original case. This is specified in [RFC1034] and [RFC1035], and further clarified by [RFC4343].

Therefore, another way to add entropy to requests is to randomly vary the case of letters in domain names queried. This technique, also known as "0x20" because bit 0x20 is used to set the case of of US-ASCII letters, was first proposed in [I-D.vixie-dnsext-dns0x20], Use of Bit 0x20 in DNS Labels to Improve Transaction Identity. With this technique, the name server response must match not only the query name, but the case of every letter in the name string; for example, `wWw.eXaMpLe.CoM` or `WwW.ExamPLe.COM`. This may add little or no entropy to queries for the top-level and root domains, but it's effective for most hostnames.

6. Consistency requirements

For DNS resolver behaviour to be consistent for a domain, it is important that the authoritative data for the domain to be consistent. All authoritative name servers for a zone should serve the same data, although it should be noted that there exists cases where authoritative name servers are configured to reply with different answers depending on the client source address and/or query options such as EDNS0 client subnet option as specified in [RFC7871].

An indicator of inconsistency is that infrastructure records (e.g., SOA and NS) differs between the authoritative name servers.

Section 4.6 in [RFC4786] advises that data synchronisation in an anycast setup should be done in a manner so that anycast nodes operate in a consistent manner.

6.1. All name servers SHOULD respond with the same SOA serial number

An indication that not all authoritative name servers have a consistent and updated copy of the zone is that the serial numbers differ. When querying for the SOA RR all name servers SHOULD respond with the same SOA serial number.

Section 4.3.5 in [RFC1034] explains the typical function of the serial numbers in zone maintenance and transfers.

One should note that even though different SOA serial numbers are a strong indicator of an inconsistent setup, there are several scenarios where the serial number varies between name servers. One example is a zone with frequent updates to zone data, where propagation delay between the name servers may result in limited inconsistency.

6.2. All name servers SHOULD respond with the same SOA RNAME

As per Section 3.3.13 of [RFC1035], the RNAME field in the SOA RDATA refers to the mailbox of the person responsible for the zone. An indication that not all authoritative name servers have a consistent and updated copy of the zone is that the RNAME differs. When querying for the SOA RR all name servers SHOULD respond with the same SOA RNAME.

6.3. All name servers SHOULD respond with the same SOA parameters

The inconsistency of the SOA parameters REFRESH, RETRY, EXPIRE and MINIMUM, defined in Section 3.3.13 of [RFC1035], might lead to operational problems for the zone. These SOA parameters SHOULD be consistent for all authoritative name servers for the zone.

6.4. All name servers MUST respond with the same NS RR Set

All authoritative name servers MUST serve the same NS record set in order to ensure consistency in the zone cut as described in Section 4.2.2 of [RFC1034]. Any inconsistency of NS records described in Section 3.3.11 of RFC 1035 might result in operational failures.

7. Delegation requirements

[RFC2182] is a BCP on how to select and operate secondary name servers, and summarize many operational issues with the delegation of a zone. For a delegation to work continuously if one component fails, there are operational considerations to ensure this.

Section 4.2.2 [RFC1034] also adds that the administrators of both the parent and child zone should ensure that NS and glue RRs on both sides of the zone cut are consistent.

7.1. The delegation SHOULD contain at least two name servers

Section 4.1 [RFC1034] states that by administrative fiat we require every zone to be available on at least two name servers. Section 5 of [RFC2182] that answers the question on how many name servers are needed, the recommendation is that "three servers be provided for

most organisation level zones, with at least one which must be well removed from the others."

In order to avoid any operational problems, a delegation SHOULD contain at least two (2) authoritative name servers.

- 7.2. The NS RR set in the parent SHOULD be a subset of the NS RR set in the child

As per the name resolving algorithm described in [RFC1034] the NS RR in the child zone is authoritative for the zone, and any delegation hints in the parent are discarded in the resolving process. The NS RR set in the parent zone SHOULD be a subset of the NS RR set in the child zone.

- 7.3. The name servers SHOULD have network path diversity

[RFC2182], Section 3.1 states that distinct authoritative name servers for a child domain should be placed in different topological and geographical locations. The objective is to minimise the likelihood of a single failure disabling all of them. Further support for this is given in Section 5:

It is recommended that three servers be provided for most organisation level zones, with at least one which must be well removed from the others.

To avoid any single point of failure in networking, the name servers SHOULD exhibit network path diversity. Using current routing technology, this means that all name servers SHOULD NOT be placed within a single routing domain, or AS (autonomous system).

- 7.4. The name servers MUST have distinct IP addresses

A common workaround to a registry policy that requires at least two name servers is to create two (2) names with the same IP address.

To avoid any operational errors and workaround such as this, all name servers used for the zone MUST use distinct IP addresses.

- 7.5. The referral SHOULD fit into a non-truncated 512 byte UDP packet

The DNS still defaults to using UDP, although efforts into requiring or transitioning to use TCP have come a long way. The UDP packet limit is 512 bytes, and although the EDNS0 [RFC6891] extension mechanism to overcome this limit have been in use for a very long time, many middleboxes and proxies still interfere with DNS packets ([RFC5625]).

To avoid any such problems with the delegation, and to avoid any unexpected truncation of a referral response, the referral containing the delegation from the parent SHOULD fit within 512 bytes.

7.6. All name servers MUST be authoritative for the domain name

A name server that does not answer authoritatively for the zone is a clear sign of misconfiguration, and is a common cause for operational problems.

Section 6.1 of [RFC2181] mandates that the name servers MUST answer authoritatively for the zone.

7.7. The delegation name MUST exactly match the apex of the child zone

The configured zone on the child name servers MUST match the delegated name of the zone. When querying the child name servers for the zone, any authoritative data for another name MUST NOT be in the response.

[RFC2181] states that the SOA RR and the NS RR indicates the origin of the zone, and both are mandatory records in a zone. Both RRs MUST be present and match the name of the zone.

7.8. Glue records in delegation SHOULD exactly match records in child zone

In-bailiwick glue for name servers listed at the parent SHOULD match the in-bailiwick glue for the name servers in the child.

If the glue address mismatch between the parent zone and the child, this is a strong indication of configuration error.

7.9. SOA MNAME SHOULD be authoritative for the zone

The hostname of the MNAME field may or may not be listed among the delegated name servers, but SHOULD still be authoritative for the zone. MNAME may be used for other services, e.g., DNS NOTIFY [RFC1996] and DNS Dynamic Updates [RFC2136].

It should be noted that there are no formal requirement that the name server listed in the SOA MNAME is reachable from the public Internet. Because of this, it may be difficult to implement a reasonable test for this requirement.

8. DNSSEC requirements

If DNSSEC is used for the zone, either by indicating that the zone is signed with a DS record, or the use of a DNSKEY in the zone itself, a number of things are required for a fully functional delegation.

The Domain Name System Security Extensions (DNSSEC) add data origin authentication and data integrity to the Domain Name System, and was first introduced with the RFCs [RFC4033], [RFC4034] and [RFC4035]. There are also a number of additions to DNSSEC such as NSEC3 described in [RFC5155], and a number of algorithms to the cryptographic functions.

8.1. The DS Digest Type MUST be assigned by IANA

The The Digest Type Field is defined as part of the DS RDATA Wire Format of Section 5.1.3 in [RFC4034]. The appendix A.2 defines the initial set of digest algorithm types with possible future algorithms. The IANA registry for DS Digest Types [IANA-DNSSEC-DS] was defined by [RFC3658].

Any DS Digest Type used for a zone MUST be assigned by IANA.

8.2. The DNSKEY algorithm MUST be assigned by IANA

The DNSKEY RR is defined in Section 2 of [RFC4034] as part of the DNSKEY RDATA Wire Format. The appendix A.1 defines the initial list of DNSKEY Algorithm Types. The IANA Registry for DNSKEY Algorithm Types [IANA-DNSSEC-DNSKEY] was created with [RFC3755].

Any DNSKEY algorithm number used for in a zone MUST be assigned by IANA.

8.3. The chain of trust for the delegation MUST be valid

A valid authentication chain from the parent DS, as described in Section 3.1 of [RFC4033], MUST exist for the SOA, DNSKEY and NS records of the child zone if a DS record is published in the parent zone.

8.4. One DS MUST match a least one DNSKEY in the child zone

DNS delegations from a parent to a child are secured with DNSSEC by publishing one or several Delegation Signer (DS) resource records in the parent zone, along with the NS records for the delegation.

As stated in Section 2.4 of [RFC4035], a DS RR SHOULD point to a DNSKEY RR that is present in the child's apex DNSKEY RRset. If there

is a DS RR published at the parent, there MUST be at least one DNSKEY RR in the child zone that matches at least one DS RR for every signature algorithm, otherwise the authentication of the referral will fail, as described in Section 5.2 of [RFC4035].

For each unique algorithm from the DS RRs present, there MUST be a matching DNSKEY using that algorithm in use in the child.

- 8.5. The number of NSEC3 iterations must not be higher than what is allowed

Section 10.3 of [RFC5155] specifies the max number of NSEC3 iterations allowed for different key sizes. This requirement is enforced by several resolver implementations.

The number of NSEC3 iterations MUST NOT be higher than what is allowed by Section 10.3 of [RFC5155]. It should be noted that the values in the table MUST be used independent of the key algorithm.

- 8.6. RRSIG validity period SHOULD NOT be too short nor too long

[RFC6781] describes operational considerations on the choice of validity periods for RRSIGs. Having too short validity periods might cause operational failure in case of unexpected events, but is good for protecting against replay attacks. Having too long validity periods may be good for operational security, but opens up for replay attacks.

The RRSIG validity periods in the zone SHOULD NOT be too short nor too long.

- 8.7. The name server MUST include RRSIG in all responses to DNSSEC queries

If the zone is signed, the name servers MUST be able to include RRSIG RRs as additional data in any response when the query has the DO bit set, as described in Section 3.1.1 of [RFC4035].

- 8.8. The name servers MUST include valid NSEC/NSEC3 record in NXDOMAIN responses

If the zone is signed, the name servers MUST be able to include NSEC/NSEC3 RRs as additional data in any response when the query has the DO bit set, as described in Section 3.1.1 of [RFC4035].

9. Syntax requirements

All domain- and host names in DNS MUST follow the rules outlined in Section 2.3.1 of [RFC1035]. The Name Syntax and LDH Label have been further clarified in Section 11 in [RFC2181] and Section 2.3.1 in [RFC5890]. From this follow the requirements below.

9.1. Illegal characters MUST NOT be in the domain name

There MUST NOT be any illegal characters used in the domain name. The domain name MUST follow the rules defined in Section 2.3.1 of [RFC1035], Section 2.1 of [RFC1123], Section 11 of [RFC2182] and Section 2 of [RFC3696].

9.2. Hyphens SHOULD NOT be in position 3 and 4 of the domain name

The effort of internationalization of domain names and the development of IDNA brought us the extension mechanism of using the string 'xn--' to have a special meaning. To allow future extensions to DNS there SHOULD be no instances of labels in the DNS that start with two characters, followed by two hyphens, where the two characters are not "xn". This has been described in Section 5 of [RFC3696].

9.3. The NS names MUST be valid hostnames

The Name Server name MUST be a valid hostname according to the rules defined in Section 2.3.1 of [RFC1035], in Section 2.1 in [RFC1123], Section 11 in [RFC2181] and Section 2 and 5 in [RFC3696].

9.4. The NS names MUST NOT be an alias

As specified in Section 10.3 of [RFC2181], the Name Server name MUST NOT be an alias (CNAME).

9.5. The SOA RNAME MUST not contain the '@' character

The SOA RNAME field is a mailbox address defined in Section 3.3 of [RFC1034] and Section 2.2 of [RFC1912]. The RNAME field MUST follow the rules of an e-mail address defined in Section 3.4.1 of [RFC2822], and the '@' character MUST be changed so that the whole e-mail address is converted into a single domain name as described in Section 3.3 of [RFC1034] and Section 2.1 of [RFC1123].

9.6. The SOA RNAME MUST be a legal hostname

The SOA RNAME field is a mailbox address. The SOA RNAME field is defined in Section 3.3 of [RFC1034] and Section 2.2 of [RFC1912]. As a field containing a domain name, the content of the RNAME field MUST follow the rules outlined in Section 2.3.1 of [RFC1035] and Section 2.1 of [RFC1123].

9.7. The SOA MNAME MUST be a legal hostname

The SOA MNAME field is a hostname. The SOA MNAME field is defined in Section 3.3.13 of [RFC1035]. As a field containing a domain name, the content of the RNAME field MUST follow the rules outlined in Section 2.3.1 of [RFC1035].

Furthermore, Section 7.3 in [RFC2181] makes it clear that the SOA MNAME field SHOULD NOT be the name of the zone itself.

9.8. The MX record in apex MUST point to a valid hostname

The requirement on the existence of an MX RR in the apex of the child zone may vary by policy from different parent zones. However, it is strongly recommended in Section 7 of [RFC2142] that all domains should have a mailbox named hostmaster@domain. SMTP can make a delivery without the MX, using the A or AAAA record as specified in Section 5.1 of [RFC5321].

If an MX RR exists in the apex of the child zone, the hostname that the MX RR points to MUST follow the rules outlined in Section 2.3.1 of [RFC1035] and Section 2.1 of [RFC1123].

10. Security Considerations

This document has no security considerations (yet).

11. IANA Considerations

This document has no IANA actions.

12. Acknowledgements

The requirements documented in this document were developed within the CENTR Test Requirements Task Force (TRTF). Most of the original requirements and text come from the Zonemaster project.

13. References

13.1. Normative References

- [RFC1034] Mockapetris, P., "Domain names - concepts and facilities", STD 13, RFC 1034, DOI 10.17487/RFC1034, November 1987, <<http://www.rfc-editor.org/info/rfc1034>>.
- [RFC1035] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, DOI 10.17487/RFC1035, November 1987, <<http://www.rfc-editor.org/info/rfc1035>>.
- [RFC1123] Braden, R., Ed., "Requirements for Internet Hosts - Application and Support", STD 3, RFC 1123, DOI 10.17487/RFC1123, October 1989, <<http://www.rfc-editor.org/info/rfc1123>>.
- [RFC2181] Elz, R. and R. Bush, "Clarifications to the DNS Specification", RFC 2181, DOI 10.17487/RFC2181, July 1997, <<http://www.rfc-editor.org/info/rfc2181>>.
- [RFC2182] Elz, R., Bush, R., Bradner, S., and M. Patton, "Selection and Operation of Secondary DNS Servers", BCP 16, RFC 2182, DOI 10.17487/RFC2182, July 1997, <<http://www.rfc-editor.org/info/rfc2182>>.
- [RFC2822] Resnick, P., Ed., "Internet Message Format", RFC 2822, DOI 10.17487/RFC2822, April 2001, <<http://www.rfc-editor.org/info/rfc2822>>.
- [RFC4035] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "Protocol Modifications for the DNS Security Extensions", RFC 4035, DOI 10.17487/RFC4035, March 2005, <<http://www.rfc-editor.org/info/rfc4035>>.
- [RFC4343] Eastlake 3rd, D., "Domain Name System (DNS) Case Insensitivity Clarification", RFC 4343, DOI 10.17487/RFC4343, January 2006, <<http://www.rfc-editor.org/info/rfc4343>>.
- [RFC7719] Hoffman, P., Sullivan, A., and K. Fujiwara, "DNS Terminology", RFC 7719, DOI 10.17487/RFC7719, December 2015, <<http://www.rfc-editor.org/info/rfc7719>>.
- [RFC7766] Dickinson, J., Dickinson, S., Bellis, R., Mankin, A., and D. Wessels, "DNS Transport over TCP - Implementation Requirements", RFC 7766, DOI 10.17487/RFC7766, March 2016, <<http://www.rfc-editor.org/info/rfc7766>>.

13.2. Informative References

- [I-D.kristoff-dnsop-dns-tcp-requirements]
Kristoff, J., "DNS Transport over TCP - Operational Requirements", draft-kristoff-dnsop-dns-tcp-requirements-01 (work in progress), August 2016.
- [I-D.vixie-dnsext-dns0x20]
Vixie, P. and D. Dagon, "Use of Bit 0x20 in DNS Labels to Improve Transaction Identity", draft-vixie-dnsext-dns0x20-00 (work in progress), March 2008.
- [IANA-DNSSEC-DNSKEY]
IANA, "Domain Name System Security (DNSSEC) Algorithm Numbers", November 2003,
<<https://www.iana.org/assignments/dns-sec-alg-numbers/dns-sec-alg-numbers.xhtml>>.
- [IANA-DNSSEC-DS]
IANA, "Delegation Signer (DS) Resource Record (RR) Type Digest Algorithms", Oktober 2003,
<<https://www.iana.org/assignments/ds-rr-types/ds-rr-types.xhtml>>.
- [IANA-IPv4-Registry]
IANA, "IANA IPv4 Address Space Registry", August 2015,
<<https://www.iana.org/assignments/ipv4-address-space>>.
- [IANA-IPv4-Special]
IANA, "IANA IPv4 Special-Purpose Address Registry", January 2006, <<https://www.iana.org/assignments/iana-ipv4-special-registry>>.
- [IANA-IPv6-Special]
IANA, "IANA IPv6 Special-Purpose Address Registry", January 2006, <<https://www.iana.org/assignments/iana-ipv6-special-registry>>.
- [IANA-IPv6-Unicast]
IANA, "IPv6 Global Unicast Address Assignments", October 2015, <<http://www.iana.org/assignments/ipv6-unicast-address-assignments>>.
- [RFC1208] Jacobsen, O. and D. Lynch, "A Glossary of Networking Terms", RFC 1208, DOI 10.17487/RFC1208, March 1991, <<http://www.rfc-editor.org/info/rfc1208>>.

- [RFC1912] Barr, D., "Common DNS Operational and Configuration Errors", RFC 1912, DOI 10.17487/RFC1912, February 1996, <<http://www.rfc-editor.org/info/rfc1912>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC2142] Crocker, D., "Mailbox Names for Common Services, Roles and Functions", RFC 2142, DOI 10.17487/RFC2142, May 1997, <<http://www.rfc-editor.org/info/rfc2142>>.
- [RFC2671] Vixie, P., "Extension Mechanisms for DNS (EDNS0)", RFC 2671, DOI 10.17487/RFC2671, August 1999, <<http://www.rfc-editor.org/info/rfc2671>>.
- [RFC2870] Bush, R., Karrenberg, D., Kesters, M., and R. Plzak, "Root Name Server Operational Requirements", RFC 2870, DOI 10.17487/RFC2870, June 2000, <<http://www.rfc-editor.org/info/rfc2870>>.
- [RFC3492] Costello, A., "Punycode: A Bootstring encoding of Unicode for Internationalized Domain Names in Applications (IDNA)", RFC 3492, DOI 10.17487/RFC3492, March 2003, <<http://www.rfc-editor.org/info/rfc3492>>.
- [RFC3658] Gudmundsson, O., "Delegation Signer (DS) Resource Record (RR)", RFC 3658, DOI 10.17487/RFC3658, December 2003, <<http://www.rfc-editor.org/info/rfc3658>>.
- [RFC3696] Klensin, J., "Application Techniques for Checking and Transformation of Names", RFC 3696, DOI 10.17487/RFC3696, February 2004, <<http://www.rfc-editor.org/info/rfc3696>>.
- [RFC3755] Weiler, S., "Legacy Resolver Compatibility for Delegation Signer (DS)", RFC 3755, DOI 10.17487/RFC3755, May 2004, <<http://www.rfc-editor.org/info/rfc3755>>.
- [RFC3871] Jones, G., Ed., "Operational Security Requirements for Large Internet Service Provider (ISP) IP Network Infrastructure", RFC 3871, DOI 10.17487/RFC3871, September 2004, <<http://www.rfc-editor.org/info/rfc3871>>.
- [RFC4033] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "DNS Security Introduction and Requirements", RFC 4033, DOI 10.17487/RFC4033, March 2005, <<http://www.rfc-editor.org/info/rfc4033>>.

- [RFC4034] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "Resource Records for the DNS Security Extensions", RFC 4034, DOI 10.17487/RFC4034, March 2005, <<http://www.rfc-editor.org/info/rfc4034>>.
- [RFC4786] Abley, J. and K. Lindqvist, "Operation of Anycast Services", BCP 126, RFC 4786, DOI 10.17487/RFC4786, December 2006, <<http://www.rfc-editor.org/info/rfc4786>>.
- [RFC5155] Laurie, B., Sisson, G., Arends, R., and D. Blacka, "DNS Security (DNSSEC) Hashed Authenticated Denial of Existence", RFC 5155, DOI 10.17487/RFC5155, March 2008, <<http://www.rfc-editor.org/info/rfc5155>>.
- [RFC5321] Klensin, J., "Simple Mail Transfer Protocol", RFC 5321, DOI 10.17487/RFC5321, October 2008, <<http://www.rfc-editor.org/info/rfc5321>>.
- [RFC5358] Damas, J. and F. Neves, "Preventing Use of Recursive Nameservers in Reflector Attacks", BCP 140, RFC 5358, DOI 10.17487/RFC5358, October 2008, <<http://www.rfc-editor.org/info/rfc5358>>.
- [RFC5625] Bellis, R., "DNS Proxy Implementation Guidelines", BCP 152, RFC 5625, DOI 10.17487/RFC5625, August 2009, <<http://www.rfc-editor.org/info/rfc5625>>.
- [RFC5890] Klensin, J., "Internationalized Domain Names for Applications (IDNA): Definitions and Document Framework", RFC 5890, DOI 10.17487/RFC5890, August 2010, <<http://www.rfc-editor.org/info/rfc5890>>.
- [RFC5891] Klensin, J., "Internationalized Domain Names in Applications (IDNA): Protocol", RFC 5891, DOI 10.17487/RFC5891, August 2010, <<http://www.rfc-editor.org/info/rfc5891>>.
- [RFC6781] Kolkman, O., Mekking, W., and R. Gieben, "DNSSEC Operational Practices, Version 2", RFC 6781, DOI 10.17487/RFC6781, December 2012, <<http://www.rfc-editor.org/info/rfc6781>>.
- [RFC6890] Cotton, M., Vegoda, L., Bonica, R., Ed., and B. Haberman, "Special-Purpose IP Address Registries", BCP 153, RFC 6890, DOI 10.17487/RFC6890, April 2013, <<http://www.rfc-editor.org/info/rfc6890>>.

- [RFC6891] Damas, J., Graff, M., and P. Vixie, "Extension Mechanisms for DNS (EDNS(0))", STD 75, RFC 6891, DOI 10.17487/RFC6891, April 2013, <<http://www.rfc-editor.org/info/rfc6891>>.
- [RFC7249] Housley, R., "Internet Numbers Registries", RFC 7249, DOI 10.17487/RFC7249, May 2014, <<http://www.rfc-editor.org/info/rfc7249>>.
- [RFC7871] Contavalli, C., van der Gaast, W., Lawrence, D., and W. Kumari, "Client Subnet in DNS Queries", RFC 7871, DOI 10.17487/RFC7871, May 2016, <<http://www.rfc-editor.org/info/rfc7871>>.

Authors' Addresses

Patrik Wallstrom

Email: pawal@blipp.com

Jakob Schlyter
Kirei AB

Email: jakob@kirei.se

dnsop
Internet-Draft
Intended status: Standards Track
Expires: March 21, 2018

J. Yao
P. Vixie
CNNIC-Farsight Joint Laboratory
N. Kong
X. Li
CNNIC
September 17, 2017

A DNS Query including A Main Question with Accompanying Questions
draft-yao-dnsop-accompanying-questions-04

Abstract

This document enables DNS initiators to send a main question accompanying with several related questions in a single DNS query, and enables DNS responders to put the answers into a single DNS response. This extension enables a range of initiators to look up "X, or failing that, Y" in a better way than both current alternatives. This mechanism can reduce the number of DNS round-trips per application work-unit.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on March 21, 2018.

Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents

carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.

Table of Contents

1. Introduction	2
2. Terminology	3
3. Mechanism for a main question with accompanying questions . .	3
4. Responder Processing	6
5. Initiator Processing	7
6. Query and Response Example	7
7. IANA Considerations	9
8. Security Considerations	9
9. Acknowledgements	9
10. Change History	9
10.1. draft-yao-dnsop-accompanying-questions: Version 00 . . .	9
10.2. draft-yao-dnsop-accompanying-questions: Version 01 . . .	9
10.3. draft-yao-dnsop-accompanying-questions: Version 02 . . .	9
10.4. draft-yao-dnsop-accompanying-questions: Version 03 . . .	10
10.5. draft-yao-dnsop-accompanying-questions: Version 04 . . .	10
11. Normative References	10
Authors' Addresses	10

1. Introduction

Sometimes, when DNS lookup of X, an application will lookup Y if X fails. For examples, the initiator may fall back to A record if the lookup of MX record fails.

Some initiators do it in sequence, X and after a few seconds, then Y. Although it is simple, this leads to unpleasant waiting whenever X times out or answers negatively.

Some initiators use concurrent X/Y lookups and a state machine to decide whether to use X or Y. If an answer to Y arrives but none to X, the initiator needs to wait a little or else fall back to Y inappropriately. Concurrent lookup is faster if the X lookup takes time and falling back to Y is appropriate, but rather complex, with four states to test, and the initiator needs to wait for an answer to X or a timeout before it can use Y.

This document enables a quicker, more easily tested failover. There is no need to test different answer sequences, there's no need for a state machine, there's no need for timeouts beyond receiving the reply. This document describes a method by which DNS initiators can send a main question accompanying with several related questions in a single DNS query, and enables DNS responders place all related answers into a single DNS response. This mechanism can reduce the number of DNS round-trips per application work-unit, by carrying several related queries in a single query transaction. It has the following advantages compared to other solutions.

- o Compared to sequential lookups: It's roughly as simple, but much faster in case a fallback to Y is necessary.
- o Compared to the concurrent mechanism: It is slightly faster (if the initiator needs to wait for an X timeout) and/or prevents inappropriate fallback (if the answer to X arrives too late), and it has a simpler state machine.

This mechanism can also be used in the scenarios when the application needs more records of the same domain name or its sub-domain name. For examples, when asking about a QTYPE=A RRset, a QTYPE=AAAA RRset may also be of use [RFC 5321]; When asking for some RRset of www.example.com about A and AAAA, records of a sub-domain name such as _443._tcp.www.example.com for TLSA may be of interest[RFC 6698].

2. Terminology

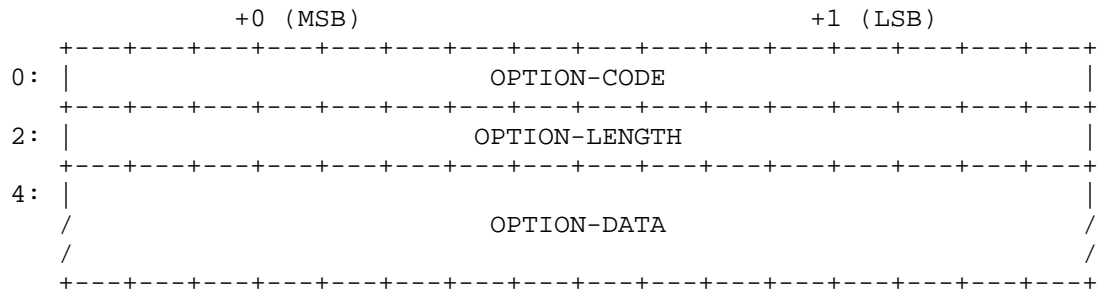
The basic key words such as "MUST", "MUST NOT", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "MAY", and "MAYNOT" are to be interpreted as described in [RFC2119].

The basic DNS terms used in this specification are defined in the documents [RFC1034] and [RFC1035].

3. Mechanism for a main question with accompanying questions

The initiator still puts a main question into the question section of the DNS query packet, as described in [RFC1035]. Accompanying questions will be put into the variable part of an OPT RR [RFC6891].

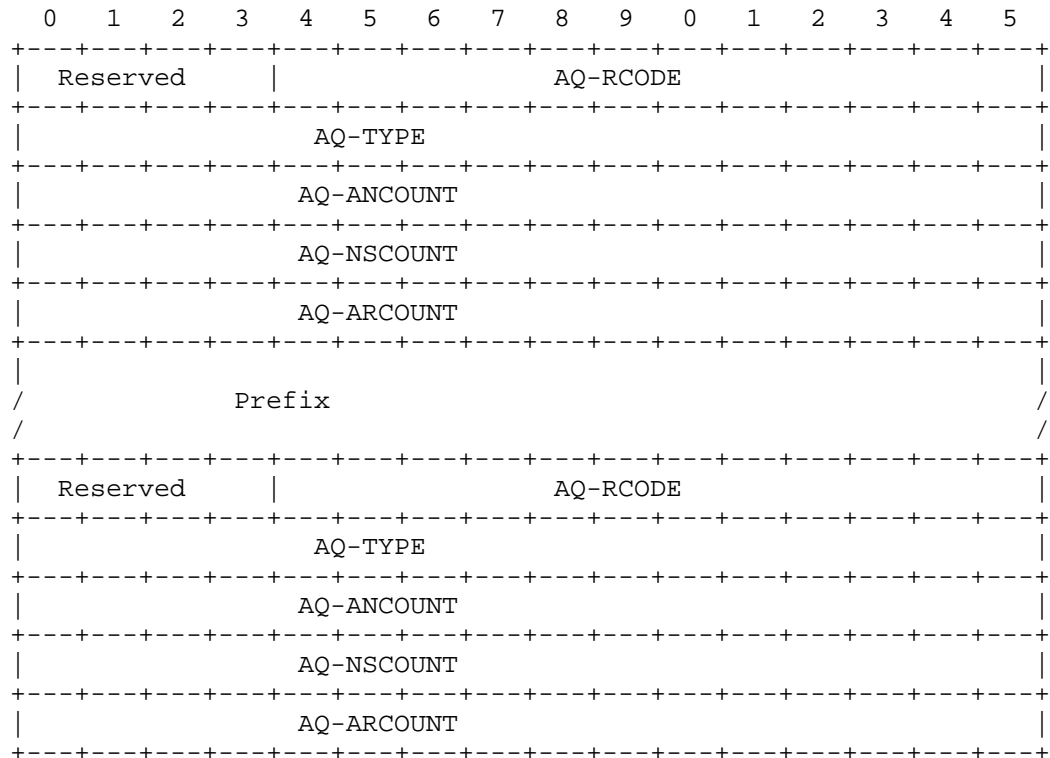
The variable part of an OPT RR is encoded in its RDATA and is structured as the following:

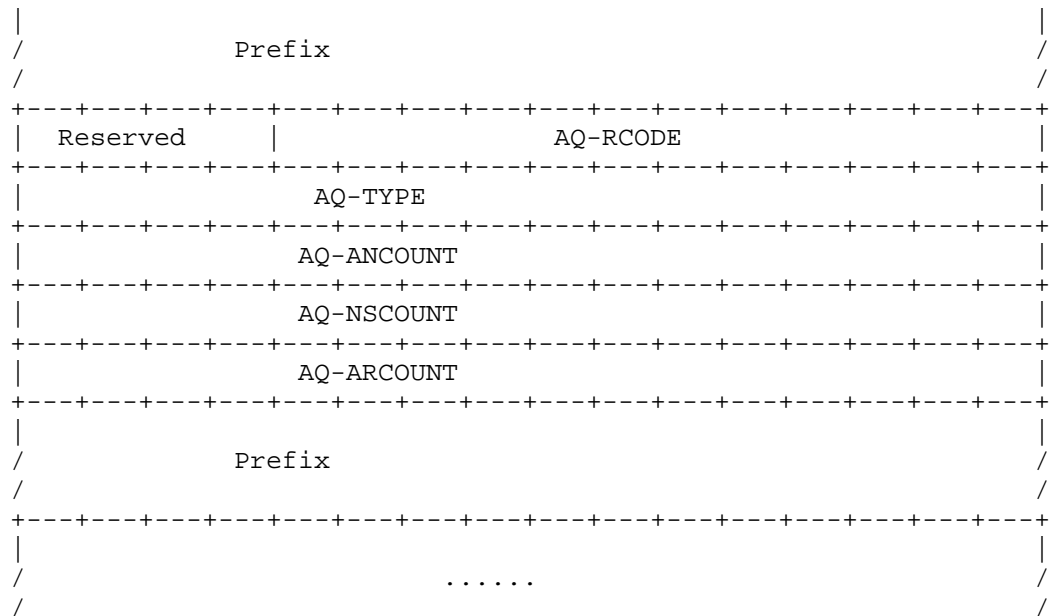


OPTION-CODE (Assigned by IANA.)

OPTION-LENGTH Size (in octets) of OPTION-DATA.

OPTION-DATA including at most 6 accompanying questions with AQ-RCODE.





- o Reserved field is kept for the future use.
- o AQ-RCODE field will be set to 111111110100 bits when being initialized. The AQ-RCODE with the value of 111111110100 bits means that the mechanism for accompanying has not been implemented, where "0100" in the RCODE value means "not been implemented". The AQ aware responders will put the RCODE value for the query of this question into AQ-RCODE fields.
- o AQ-ANCOUNT field will indicate the number of resource records in the answer section for this accompanying question. The AQ aware responders will put the ANCOUNT value for the query of this question into AQ-ANCOUNT field.
- o AQ-NSCOUNT field will indicate the number of name server resource records in the authority records section for this accompanying question. The AQ aware responders will put the NSCOUNT value for the query of this question into AQ-NSCOUNT field.
- o AQ-ARCOUNT field will indicate the number of resource records in the additional records section for this accompanying question. The AQ aware responders will put the ARCOUNT value for the query of this question into AQ-ARCOUNT field.

- o Prefix field indicates a domain name with the form of a dot or a sequence of labels ending with a pointer using the message compression defined in section 4.1.4. of RFC 1035. The domain name for accompanying questions MUST be same with the domain name for a main question or be children name of it. For an example, if the main domain name is example.com and the accompanying domain name is mail.example.com., the prefix is "mail." ending with a pointer pointing to "example.com.".

4. Responder Processing

The AQ aware responder will check the main question first, and put the results into the DNS response packet following RFC 1034. If the AQ OPT is present, the responder assembles the prefix with the main domain name and makes it to be an accompanying question, checks the accompanying questions in order, and put the results into the DNS answer section, authority section or additional records section of the response following RFC 1034; but the response code is placed in the respective AQ-RCODE field in AQ OPT of the response. The RCODE field in the DNS response header refers to the main question only. The AQ aware responders will put the ANCOUNT, NSCOUNT and ARCOUNT value for the query of this accompanying question into the respective AQ-ANCOUNT, AQ-NSCOUNT and AQ-ARCOUNT fields. The ANCOUNT, NSCOUNT and ARCOUNT fields in the DNS response header refer to the main question and its accompanying questions. Since the value for the accompanying questions' ANCOUNT, NSCOUNT and ARCOUNT can be known from the respective value of AQ-ANCOUNT, AQ-NSCOUNT and AQ-ARCOUNT, the actual value of the main question's ANCOUNT, NSCOUNT and ARCOUNT can be calculated from the ANCOUNT, NSCOUNT and ARCOUNT in the DNS response header. When the answer is negative for the accompanying question, the SOA resource record will be put in the authority section.

The mechanism proposed in this document is intended for both between stub resolvers and recursive resolvers, and between recursive resolvers and authoritative servers. If some DNS resource records are needed to be processed at the same time, the DNS administrator may configure it together. In case of that some children domain names are delegated and not in the main domain name's zone, the delegation information will be returned to the recursive resolvers. The recursive resolvers then check the children domain based on the delegation information, and get the answer for the respective children domain names.

When a stub resolver sends an AQ query to the recursive resolver, the recursive resolver may have some answers for one or more questions in the cache, but not for all questions. Under that case, the recursive resolver SHOULD forward this AQ query to some relative authoritative

servers for full answers instead of using the existing insufficient cache information.

An AQ unaware responder is expected to ignore the AQ OPT of the query, and may echo the received OPT back into additional section of the response message.

5. Initiator Processing

An AQ aware initiator will put the main question into the question section of the DNS query packet, and put each accompanying question into the related accompanying question fields of OPTION-DATA of OPT RR. AQ-RCODE value will be sent as 11111110100 bits. The AQ-TYPE value should be set as the query type related to accompanying questions. The Prefix value should be set as a dot or a sequence of labels ending with a pointer pointing to the the main domain name of the main question for the respective accompanying domain name of the accompanying question.

An AQ aware initiator SHOULD set the limitation of what is the maximum number of accompanying questions a AQ query can bring. This document suggests that the maximum number is six since most DNS resource records which need parallel query will not larger than six. The implementers may set six as the default value in the implementation. The responder can refuse to answer the AQ query if the maximum number of the accompanying questions is larger than the default maximum value, and return "not been implemented, too many accompanying-questions." information to the initiator.

If the initial value of the AQ-RCODE is unchanged in the response or the AQ OPT is not echo back, it indicates that the responder is AQ unaware. In that case, the responder will deal with the main question only. The initiator should sent the accompanying questions one by one via the normal DNS query. In such followup related queries, AQ processing should probably not be attempted, to reduce waste of network resources.

6. Query and Response Example

Example: one main question with 2 accompanying questions

The query would look like:

Header	OPCODE=SQUERY
Question	QNAME=EXAMPLE.COM., QCLASS=IN, QTYPE=A

Answer		
Authority	<empty>	
Additional		
	AQ-TYPE=AAAA, AQ-RCODE=111111110100,	
	Prefix=.,	
	AQ-TYPE=TLSA, , AQ-RCODE=111111110100,	
	Prefix=_443._tcp.,	

The response from AQ aware responders would be:

Header		
	OPCODE=SQUERY, RESPONSE, AA, RCODE=NOERROR	
	ANCOUNT=3, ARCOUNT=1, NSCOUNT=0	
Question		
	QNAME=EXAMPLE.COM., QCLASS=IN, QTYPE=A	
Answer		
	example.com IN A 192.168.0.1	
	example.com. IN AAAA 2001:cc8::1	
	_443._tcp.example.com. IN TLSA	
	(3 0 0 30820307308201efa003020102020...)	
Authority		
	<empty>	
Additional		
	AQ-TYPE=AAAA, AQ-RCODE=NOERROR, AQ-ANCOUNT=1,	
	AQ-ARCOUNT=0, AQ-NSCOUNT=0,	
	Prefix=.,	
	AQ-TYPE=TLSA, AQ-RCODE=NOERROR, AQ-ANCOUNT=1,	
	AQ-ARCOUNT=0, AQ-NSCOUNT=0,	
	Prefix=_443._tcp.,	

The response from AQ unaware responders would be:

Header		
	OPCODE=SQUERY, RESPONSE, AA, RCODE=NOERROR	
Question		
	QNAME=EXAMPLE.COM., QCLASS=IN, QTYPE=A	
Answer		
	example.com. IN A 192.168.0.1	
Authority		
	<empty>	
Additional		

```

| AQ-TYPE=AAAA,AQ-RCODE=111111110100,      |
| Prefix=.,                                |
| AQ-TYPE=TLSA, AQ-RCODE=111111110100,      |
| Prefix=_443._tcp.,                        |
+-----+

```

7. IANA Considerations

IANA should allocate DNS EDNS0 Option Codes (OPT) following this document. IANA should reserve RCODE with the value of 111111110100 bits for this document.

8. Security Considerations

TBD

9. Acknowledgements

The authors thank the members in DNSOP mailing list for helpful discussions, and especially thank Kazunori Fujiwara, JINMEI Tatuya, Bob Harold, Arnt Gulbrandsen, Olafur Gudmundsson and Stephane Bortzmeyer for kind comments, suggestions and improvements for the document. The authors also thanks Likun Zhang for helpful discussion about some topics related to implementation.

10. Change History

RFC Editor: Please remove this section.

10.1. draft-yao-dnsop-accompanying-questions: Version 00

- o A Mechanism for DNS query including one main question with several accompanying questions

10.2. draft-yao-dnsop-accompanying-questions: Version 01

- o Simplify the mechanism.

10.3. draft-yao-dnsop-accompanying-questions: Version 02

- o Remove the AQ and Count bits, and add AQ-ANCOUNT AQ-ARCOUNT AQ-NSCOUNT

10.4. draft-yao-dnsop-accompanying-questions: Version 03

- o Improve the introduction and explains the motivation of this draft

10.5. draft-yao-dnsop-accompanying-questions: Version 04

- o Improve the document

11. Normative References

- [RFC1034] Mockapetris, P., "Domain names - concepts and facilities", STD 13, RFC 1034, DOI 10.17487/RFC1034, November 1987, <<https://www.rfc-editor.org/info/rfc1034>>.
- [RFC1035] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, DOI 10.17487/RFC1035, November 1987, <<https://www.rfc-editor.org/info/rfc1035>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC5321] Klensin, J., "Simple Mail Transfer Protocol", RFC 5321, DOI 10.17487/RFC5321, October 2008, <<https://www.rfc-editor.org/info/rfc5321>>.
- [RFC6698] Hoffman, P. and J. Schlyter, "The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA", RFC 6698, DOI 10.17487/RFC6698, August 2012, <<https://www.rfc-editor.org/info/rfc6698>>.
- [RFC6891] Damas, J., Graff, M., and P. Vixie, "Extension Mechanisms for DNS (EDNS(0))", STD 75, RFC 6891, DOI 10.17487/RFC6891, April 2013, <<https://www.rfc-editor.org/info/rfc6891>>.

Authors' Addresses

Jiankang Yao
CNNIC-Farsight Joint Laboratory
4 South 4th Street, Zhongguancun, Haidian District
Beijing, Beijing 100190
China

Phone: +86 10 5881 3007
Email: yaojk@cnnic.cn

Paul Vixie
CNNIC-Farsight Joint Laboratory
4 South 4th Street, Zhongguancun, Haidian District
Beijing, Beijing 100190
China

Phone: +1 650 489 7919
Email: vixie@fsi.io

Ning Kong
CNNIC
4 South 4th Street, Zhongguancun, Haidian District
Beijing, Beijing 100190
China

Phone: +86 10 5881 3147
Email: nkong@cnnic.cn

Xiaodong Li
CNNIC
4 South 4th Street, Zhongguancun, Haidian District
Beijing, Beijing 100190
China

Phone: +86 10 5881 3020
Email: xl@cnnic.cn

DNSOP
Internet-Draft
Intended status: Informational
Expires: April 24, 2019

D. York
Internet Society
O. Sury
ISC
P. Wouters
Red Hat
O. Gudmundsson
CloudFlare
October 21, 2018

Observations on Deploying New DNSSEC Cryptographic Algorithms
draft-york-dnsop-deploying-dnssec-crypto-algs-06

Abstract

As new cryptographic algorithms are developed for use in DNSSEC signing and validation, this document captures the steps needed for new algorithms to be deployed and enter general usage. The intent is to ensure a common understanding of the typical deployment process and potentially identify opportunities for improvement of operations.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on April 24, 2019.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents

carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
1.1. Terminology	3
2. Aspects of Deploying New Algorithms	3
2.1. DNS Resolvers Performing Validation	4
2.1.1. Resolvers and Unknown Algorithms	4
2.2. Authoritative DNS Servers	5
2.3. Signing Software	5
2.3.1. NSEC3 Iterations	5
2.4. Registries	6
2.5. Registrars	6
2.6. DNS Hosting Operators	7
2.7. Applications	7
3. Conclusion	7
4. IANA Considerations	8
5. Security Considerations	8
6. References	8
6.1. Normative References	8
6.2. Informative References	9
Appendix A. Acknowledgements	10
Appendix B. Changes	10
Authors' Addresses	11

1. Introduction

The DNS Security Extensions (DNSSEC), broadly defined in [RFC4033], [RFC4034] and [RFC4035], make use of cryptographic algorithms in both the signing of DNS records and the validation of DNSSEC signatures by recursive resolvers.

The current list of cryptographic algorithms can be found in the IANA "Domain Name System Security (DNSSEC) Algorithm Numbers" registry located at <http://www.iana.org/assignments/dns-sec-alg-numbers/>. Algorithms are added to this IANA registry through a process defined in [RFC6014]. Note that [RFC6944] provides some guidance as to which of these algorithms should be implemented and supported.

Historically DNSSEC signatures have primarily used cryptographic algorithms based on RSA keys. As deployment of DNSSEC has increased there has been interest in using newer and more secure algorithms, particularly those using elliptic curve cryptography.

The ECDSA algorithm [RFC6605] has seen some adoption and a new signing algorithm is now available: Edwards-curve Digital Signature Algorithm (EdDSA) using a choice of two curves, Ed25519 and Ed448, [RFC8080].

The challenge is that the deployment of a new cryptographic algorithm for DNSSEC is not a simple process. DNSSEC algorithms are used throughout the DNS infrastructure for tasks such as:

- o Generation of keys ("DNSKEY" record) for signing
- o Creation of DNSSEC signatures in zone files ("RRSIG")
- o Usage in a Delegation Signer ("DS") record [RFC3658] for the "chain of trust" connecting back to the root of DNS
- o Generation of NSEC/NSEC3 responses by authoritative DNS servers
- o Validation of DNSSEC signatures by DNS resolvers

In order for a new cryptographic algorithm to be fully deployed, all aspects of the DNS infrastructure that interact with DNSSEC must be updated to use the new algorithm.

This document outlines the current understanding of the components of the DNS infrastructure that need to be updated to deploy a new cryptographic algorithm.

It should be noted that DNSSEC is not alone in complexity of deployment. The IAB documented "Guidelines for Cryptographic Algorithm Agility" in [RFC7696] to highlight the importance of this issue.

1.1. Terminology

In this document, the key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" are to be interpreted as described in BCP 14, RFC 2119 [RFC2119].

2. Aspects of Deploying New Algorithms

For a new cryptographic algorithm to be deployed in DNSSEC, the following aspects of the DNS infrastructure must be updated:

- o DNS resolvers performing validation
- o Authoritative DNS servers

- o Signing software
- o Registries
- o Registrars
- o DNS Hosting Operators
- o Applications

Each of these aspects is discussed in more detail below.

2.1. DNS Resolvers Performing Validation

DNS recursive resolvers perform "validation" to check the DNSSEC signatures of records received in a DNS query. To validate the signatures, the resolvers need to be able to understand the algorithm used to create the signatures.

In the case of a new algorithm, the resolver software needs to be updated. In some cases this could require waiting until an underlying library is updated to support the new algorithm.

Once the software is updated, the updates need to be deployed to all resolvers using that software. This can be challenging in cases of customer-premises equipment (CPE) that does not have any mechanism for automatic updating.

2.1.1. Resolvers and Unknown Algorithms

It should be noted that section 5.2 of [RFC4035] states:

"If the resolver does not support any of the algorithms listed in an authenticated DS RRset, then the resolver will not be able to verify the authentication path to the child zone. In this case, the resolver SHOULD treat the child zone as if it were unsigned."

This means that signing a zone with a new algorithm that is not widely supported by DNS resolvers would result in the signatures being ignored and the zone treated as unsigned until resolvers were updated to recognize the new algorithm.

Note that in at least one 2016 case the resolver software deployed on customer premises by an Internet service provider (ISP) turned out not to be compliant with RFC 4035. Instead of ignoring the signatures using unknown algorithms and treating the zones as unsigned, the validating resolver rejected the signatures and returned a SERVFAIL to the DNS query. This resulted in the ISP

turning off DNSSEC validation on the equipment. Further investigation showed that a newer version of the resolver software did correctly support ECDSA, but now all customer premises equipment must be updated to this new version.

The point is that it is not safe to assume all resolver software will correctly implement this part of RFC 4035.

2.2. Authoritative DNS Servers

Authoritative DNS servers serve out signed DNS records. Serving new DNSSEC signing algorithms should not be a problem as a well-written authoritative DNS server implementation should be agnostic to the RR DATA they serve.

The one exception is if the new cryptographic algorithms are used in the creation of NSEC/NSEC3 responses. In the case of new NSEC/NSEC3 algorithms, the authoritative DNS server software would need to be updated to be able to use the new algorithms.

Note that some authoritative server implementations could include DNSSEC signing as part of the server and thus also fall into the "Signing Software" category below.

2.3. Signing Software

The software performing the signing of the records needs to be updated with the new cryptographic algorithm.

User interfaces that allow users to interact with the DNSSEC signing software may also need to be updated to reflect the existence of the new algorithm.

Note that the key and signatures with the new algorithm will need to co-exist with the existing key and signatures for some period of time. This will have an impact on the size of the DNS records.

One issue that has been identified is that not all commonly-used signing software releases include support for an algorithm rollover. This software would need to be updated to support rolling an algorithm before any new algorithms could be deployed.

2.3.1. NSEC3 Iterations

[additional text needed]

2.4. Registries

The registry for a top-level domain (TLD) needs to accept DS records using the new cryptographic algorithm.

Observations to date have shown that some registries only accept DS records with certain algorithms. Registry representatives have indicated that they verify the accuracy of DS records to reduce technical support incidents and ensure customers do not mistakenly create any outages.

However, this means that registries who perform this level of checking must be able to understand new algorithms in order to successfully verify the DS records.

Separately, feedback from registrars has indicated that they do not currently have any mechanism to understand what DNSSEC algorithms a registry can accept.

2.5. Registrars

Registrars perform a critical role in the DNSSEC "chain of trust" of passing the DS record up to the Registry to ensure that the signed zone can be authenticated from the root of DNS all the way to the zone.

If the registrar is also providing the DNS hosting services for a domain, the registrar can easily create the "DS" record from the "DNSKEY" record and pass the DS record up to the registry.

However, if the authoritative servers for a domain are not with the registrar, then the registrar needs to provide some mechanism to accept a DS record to pass that up to the registry. Typically this is done through a web interface.

An issue is that many registrar web interfaces only allow the input of DS records using a listed set of DNSSEC algorithms. Any new cryptographic algorithms need to be added to the web interface in order to be accepted into the registrar's system.

Additionally, in a manner similar to registries, many registrars perform some level of verification on the DS record to ensure it was entered "correctly". To do this verification, the registrar's software needs to understand the algorithm used in the DS record. This requires the software to be updated to support the new algorithm.

Note that work has been standardized in [RFC8078] to provide an automated mechanism to update the DS records with a registry. If this method becomes widely adopted, registrar web interfaces may no longer be needed.

2.6. DNS Hosting Operators

DNS hosting operators are entities that are operating the authoritative DNS servers for domains and with DNSSEC are also providing the signing of zones. In many cases they may also be the registrar for domain names, but in other cases they are a separate entity providing DNS services to customers.

DNS hosting operators need to update their authoritative DNS server software to understand new cryptographic algorithms, but they also need to update their web interfaces and provisioning software to allow configuration and support of new algorithms.

2.7. Applications

Beyond the recursive resolvers, authoritative servers, web interfaces and provisioning software, it has been observed that some applications (or "apps"), particularly in the mobile environment, are including their own DNS resolvers within the app itself. These recursive resolvers are used by the app instead of the recursive resolver included with the underlying operating system. These applications that perform DNSSEC validation would need to also be updated to understand a new algorithm.

In many cases, it may be that an underlying developer library needs to be updated first. It will then depend upon how long it takes the application developer to pull in the updated library.

Outside of applications, these developer libraries are also typically used by recursive resolver software and signing software.

3. Conclusion

This document provides a view into the steps necessary for the deployment of new cryptographic algorithms in DNSSEC at the time of this publication. In order to more rapidly roll out new DNSSEC algorithms, these steps must be understood and hopefully improved over time.

It should be noted that a common theme to emerge from all discussions is a general reluctance to update or change any DNS-related software. "If it isn't broken, don't fix it" is a common refrain. While

perhaps understandable from a stability point of view, this attitude creates a challenge for deploying new algorithms.

One potential idea suggested during discussions was for some kind of web-based testing tool that could assist people in understanding what algorithms are supported by different servers and sites.

It is also quite clear that any deployment of new algorithms for DNSSEC use will take a few years to propagate throughout the infrastructure. This needs to be factored in as new algorithms are proposed.

4. IANA Considerations

This document does not make any requests of IANA.

5. Security Considerations

No new security considerations are created by this document.

It should be noted that there are security considerations regarding changing DNSSEC algorithms that are mentioned in both [RFC6781] and [RFC7583].

6. References

6.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC4033] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "DNS Security Introduction and Requirements", RFC 4033, DOI 10.17487/RFC4033, March 2005, <<https://www.rfc-editor.org/info/rfc4033>>.
- [RFC4034] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "Resource Records for the DNS Security Extensions", RFC 4034, DOI 10.17487/RFC4034, March 2005, <<https://www.rfc-editor.org/info/rfc4034>>.
- [RFC4035] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "Protocol Modifications for the DNS Security Extensions", RFC 4035, DOI 10.17487/RFC4035, March 2005, <<https://www.rfc-editor.org/info/rfc4035>>.

6.2. Informative References

- [RFC3658] Gudmundsson, O., "Delegation Signer (DS) Resource Record (RR)", RFC 3658, DOI 10.17487/RFC3658, December 2003, <<https://www.rfc-editor.org/info/rfc3658>>.
- [RFC6014] Hoffman, P., "Cryptographic Algorithm Identifier Allocation for DNSSEC", RFC 6014, DOI 10.17487/RFC6014, November 2010, <<https://www.rfc-editor.org/info/rfc6014>>.
- [RFC6605] Hoffman, P. and W. Wijngaards, "Elliptic Curve Digital Signature Algorithm (DSA) for DNSSEC", RFC 6605, DOI 10.17487/RFC6605, April 2012, <<https://www.rfc-editor.org/info/rfc6605>>.
- [RFC6781] Kolkman, O., Mekking, W., and R. Gieben, "DNSSEC Operational Practices, Version 2", RFC 6781, DOI 10.17487/RFC6781, December 2012, <<https://www.rfc-editor.org/info/rfc6781>>.
- [RFC6944] Rose, S., "Applicability Statement: DNS Security (DNSSEC) DNSKEY Algorithm Implementation Status", RFC 6944, DOI 10.17487/RFC6944, April 2013, <<https://www.rfc-editor.org/info/rfc6944>>.
- [RFC7583] Morris, S., Ihren, J., Dickinson, J., and W. Mekking, "DNSSEC Key Rollover Timing Considerations", RFC 7583, DOI 10.17487/RFC7583, October 2015, <<https://www.rfc-editor.org/info/rfc7583>>.
- [RFC7696] Housley, R., "Guidelines for Cryptographic Algorithm Agility and Selecting Mandatory-to-Implement Algorithms", BCP 201, RFC 7696, DOI 10.17487/RFC7696, November 2015, <<https://www.rfc-editor.org/info/rfc7696>>.
- [RFC8078] Gudmundsson, O. and P. Wouters, "Managing DS Records from the Parent via CDS/CDNSKEY", RFC 8078, DOI 10.17487/RFC8078, March 2017, <<https://www.rfc-editor.org/info/rfc8078>>.
- [RFC8080] Sury, O. and R. Edmonds, "Edwards-Curve Digital Security Algorithm (EdDSA) for DNSSEC", RFC 8080, DOI 10.17487/RFC8080, February 2017, <<https://www.rfc-editor.org/info/rfc8080>>.

Appendix A. Acknowledgements

The information in this document evolved out of several mailing list discussions and also through engagement with participants in the following sessions or events:

- o DNSSEC Workshop at ICANN 53 (Buenos Aires)
- o DNSSEC Workshop at ICANN 55 (Marrakech)
- o Spring 2016 DNS-OARC meeeting (Buenos Aires)
- o various IETF 95 working groups (Buenos Aires)
- o Panel session at RIPE 72 (Copenhagen)
- o DNSSEC Workshop at ICANN 56 (Helsinki)

The authors thank the participants of the various sessions for their feedback.

Appendix B. Changes

NOTE TO RFC EDITOR - Please remove this "Changes" section prior to publication. Thank you.

- o Revision -06 added in references to RFCs 8080 and 8078 and updated Ondrej Sury's affiliation to ISC.
- o Revision -05 corrected typos around two other references that did not appear in -04.
- o Revision -04 corrected the references which did not appear in -03 due to an error in the markdown source.
- o Revision -03 removed the reference to the location of the ISP in the text added in version -02.
- o Revision -02 added text to the resolver section about an example where resolver software did not correctly follow RFC 4035 and treat packets with unknown algorithms as unsigned. The markdown source of this I-D was also migrated to the markdown syntax favored by the 'mmark' tool.
- o Revision -01 adds text about authoritative servers needing an update if the algorithm is for NSEC/NSEC3. Also expands acknowledgements.

Authors' Addresses

Dan York
Internet Society

Email: york@isoc.org
URI: <https://www.internetsociety.org/>

Ondrej Sury
ISC

Email: ondrej@isc.org

Paul Wouters
Red Hat

Email: pwouters@redhat.com

Olafur Gudmundsson
CloudFlare

Email: olafur+ietf@cloudflare.com