

DNSOP Working Group
Internet-Draft
Updates: RFC 2845, RFC 2931 (if
approved) (if approved)
Intended status: Standards Track
Expires: September 6, 2018

R. Bellis
ISC
P. van Dijk
R. Gacogne
PowerDNS
March 05, 2018

DNS X-Proxied-For
draft-bellis-dnsop-xpf-04

Abstract

It is becoming more commonplace to install front end proxy devices in front of DNS servers to provide (for example) load balancing or to perform transport layer conversions.

This document defines a meta resource record that allows a DNS server to receive information about the client's original transport protocol parameters when supplied by trusted proxies.

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 6, 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 (<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. Terminology	3
3. Description	3
3.1. Client Handling	3
3.2. Request Handling	4
3.3. Proxy Handling	4
3.4. Server Handling	4
3.5. Wire Format	4
3.6. Presentation Format	6
3.7. Signed DNS Requests	6
4. Security Considerations	6
5. Implementation status	7
5.1. dnsmdist	7
5.2. PowerDNS Recursor	7
5.3. Wireshark	7
6. Privacy Considerations	7
7. IANA Considerations	8
8. Acknowledgements	8
9. References	8
9.1. Normative References	8
9.2. Informative References	9
Authors' Addresses	9

1. Introduction

It is becoming more commonplace to install front end proxy devices in front of DNS servers [RFC1035] to provide load balancing or to perform transport layer conversions (e.g. to add DNS over TLS [RFC7858] to a DNS server that lacks native support).

This has the unfortunate side effect of hiding the clients' source IP addresses from the server, making it harder to employ server-side technologies that rely on knowing those addresses (e.g. ACLs, DNS Response Rate Limiting, etc).

This document defines the XPF meta resource record (RR) that allows a DNS server to receive information about the client's original transport protocol parameters when supplied by trusted proxies.

Whilst in some circumstances it would be possible to re-use the Client Subnet EDNS Option [RFC7871] to carry a subset of this

information, a new RR is defined to allow both this feature and the Client Subnet Option to co-exist in the same packet.

2. 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.

The XPF RR is analogous to the HTTP "X-Forwarded-For" header, but in DNS the term "forwarder" is usually understood to describe a network component that sits on the outbound query path of a resolver.

Instead we use the term "proxy", which in this document means a network component that sits on the inbound query path in front of a recursive or authoritative DNS server, receiving DNS queries from clients and dispatching them to local servers.

3. Description

The XPF RR contains the entire 6-tuple (IP version, Layer 4 protocol, source address, destination address, source port and destination port) of the packet received from the client by the proxy.

The presence of the source address supports use of ACLs based on the client's IP address.

The source port allows for ACLs to support Carrier Grade NAT whereby different end-users might share a single IP address.

The destination address supports scenarios where the server behaviour depends upon the packet destination (e.g. BIND view's "match-destinations" option)

The protocol and destination port fields allow server behaviour to vary depending on whether DNS over TLS [RFC7858] or DNS over DTLS [RFC8094] are in use.

3.1. Client Handling

Stub resolvers, client-side proxy devices, and recursive resolvers MUST NOT add the XPF RR to DNS requests.

3.2. Request Handling

The rules in this section apply to processing of the XPF RR whether by a proxy device or a DNS server.

If this RR is received from a non-white-listed client the server MUST return a REFUSED response.

If a server finds this RR anywhere other than in the Additional Section of a request it MUST return a REFUSED response.

If the value of the RR's IP version field is not understood by the server it MUST return a REFUSED response.

If the length of the IP addresses contained in the RR are not consistent with that expected for the given IP version then the server MUST return a FORMERR response.

Servers MUST NOT send this RR in DNS responses.

3.3. Proxy Handling

For each request received, proxies MUST generate an XPF RR containing the 6-tuple representing the client's Layer 3 and Layer 4 headers and append it to the Additional Section of the request (updating the ARCOUNT field accordingly) before sending it to the intended DNS server.

If a valid XPF RR is received from a white-listed client the original XPF RR MUST be preserved instead.

3.4. Server Handling

When this RR is received from a white-listed client the DNS server SHOULD use the transport information contained therein in preference to the packet's own transport information for any data processing logic (e.g. ACLs) that would otherwise depend on the latter.

3.5. Wire Format

The XPF RR is formatted like any standard RR, but none of the fields except RDLLENGTH and RDATA have any meaning in this specification. All multi-octet fields are transmitted in network order (i.e. big-endian).

The required values of the RR header fields are as follows:

NAME: MUST contain a single 0 octet (i.e. the root domain).

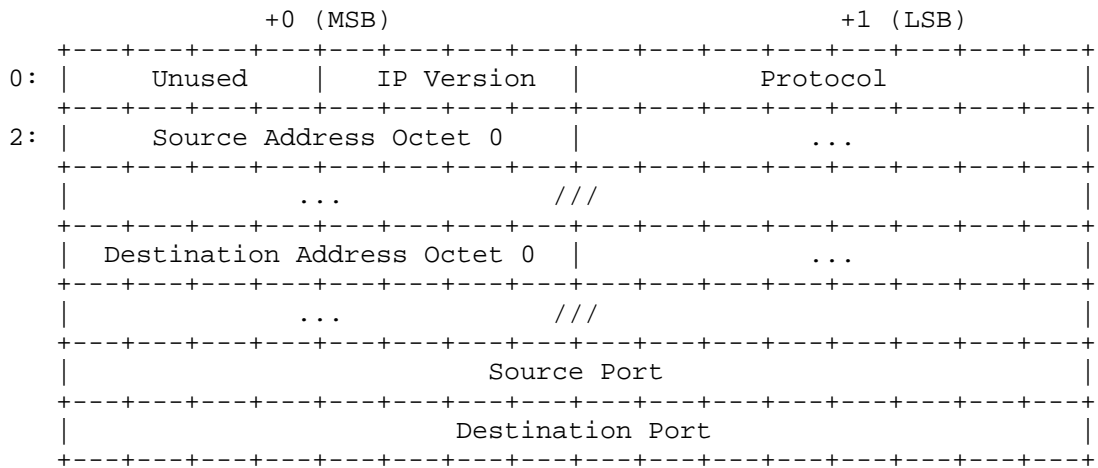
TYPE: MUST contain TBD1 (XPF).

CLASS: MUST contain 1 (IN).

TTL: MUST contain 0 (zero).

RDLENGTH: specifies the length in octets of the RDATA field.

The RDATA of the XPF RR is as follows:



Unused: Currently reserved. These bits MUST be zero unless redefined in a subsequent specification.

IP Version: The IP protocol version number used by the client, as defined in the IANA IP Version Number Registry [IANA-IP]. Implementations MUST support IPv4 (4) and IPv6 (6).

Protocol: The Layer 4 protocol number (e.g. UDP or TCP) as defined in the IANA Protocol Number Registry [IANA-PROTO].

Source Address: The source IP address of the client.

Destination Address: The destination IP address of the request, i.e. the IP address of the proxy on which the request was received.

Source Port: The source port used by the client.

Destination Port: The destination port of the request.

The length of the Source Address and Destination Address fields will be variable depending on the IP Version used by the client.

3.6. Presentation Format

XPF is a meta RR that cannot appear in master format zone files, but a standardised presentation format is defined here for use by debugging utilities that might need to display the contents of an XPF RR.

The Unused bits and the IP Version field are treated as a single octet and presented as an unsigned decimal integer with range 0 .. 255.

The Protocol field is presented as an unsigned decimal integer with range 0 .. 255.

The Source and Destination Address fields are presented either as IPv4 or IPv6 addresses according to the IP Version field. In the case of IPv6 the recommendations from [RFC5952] SHOULD be followed.

The Source and Destination Port fields are presented as unsigned decimal integers with range 0 .. 65535.

3.7. Signed DNS Requests

Any XPF RRs found in a packet MUST be ignored for the purposes of calculating or verifying any signatures used for Secret Key Transaction Authentication for DNS [RFC2845] or DNS Request and Transaction Signatures (SIG(0)) [RFC2931].

Typically it is expected that proxies will append the XPF RR to the packet after any existing TSIG or SIG(0) RRs, and that servers will remove the XPF RR from the packet prior to verification of the original signature, with the ARCOUNT field updated as appropriate.

If either TSIG or SIG(0) are configured between the proxy and server then any XPF RRs MUST be ignored when the proxy calculates the packet signature.

4. Security Considerations

If the white-list of trusted proxies is implemented as a list of IP addresses, the server administrator MUST have the ability to selectively disable this feature for any transport where there is a possibility of the proxy's source address being spoofed.

This does not mean to imply that use over UDP is impossible - if for example the network architecture keeps all proxy-to-server traffic on a dedicated network and clients have no direct access to the servers then the proxies' source addresses can be considered unspoofable.

5. Implementation status

[RFC Editor Note: Please remove this entire section prior to publication as an RFC.]

This section records the status of known implementations of the protocol defined by this specification at the time of posting of this Internet-Draft, and is based on a proposal described in [RFC7942]. The description of implementations in this section is intended to assist the IETF in its decision processes in progressing drafts to RFCs. Please note that the listing of any individual implementation here does not imply endorsement by the IETF. Furthermore, no effort has been spent to verify the information presented here that was supplied by IETF contributors. This is not intended as, and must not be construed to be, a catalog of available implementations or their features. Readers are advised to note that other implementations may exist.

According to [RFC7942], "this will allow reviewers and working groups to assign due consideration to documents that have the benefit of running code, which may serve as evidence of valuable experimentation and feedback that have made the implemented protocols more mature. It is up to the individual working groups to use this information as they see fit".

5.1. dnssdist

Support for adding an XPF RR to proxied packets is provided in the git version of dnssdist. The code point is configurable.

5.2. PowerDNS Recursor

Support for extracting the XPF RR from received packets (when coming from a trusted source) is available in the git version of the PowerDNS Recursor. The code point is configurable.

5.3. Wireshark

Support for dissecting XPF RRs is present in Wireshark 2.5.0, using a temporary code point of 65422.

6. Privacy Considerations

Used incorrectly, this RR could expose internal network information, however it is not intended for use on proxy / forwarder devices that sit on the client-side of a DNS request.

This specification is only intended for use on server-side proxy devices that are under the same administrative control as the DNS servers themselves. As such there is no change in the scope within which any private information might be shared.

Use other than as described above would be contrary to the principles of [RFC6973].

7. IANA Considerations

<< a copy of the RFC 6895 IANA RR TYPE application template will appear here >>

8. Acknowledgements

Mark Andrews, Robert Edmonds, Duane Wessels

9. References

9.1. Normative References

- [IANA-IP] IANA, "IANA IP Version Registry", n.d.,
<<http://www.iana.org/assignments/version-numbers/>>.
- [IANA-PROTO] IANA, "IANA Protocol Number Registry", n.d.,
<<http://www.iana.org/assignments/protocol-numbers/>>.
- [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>>.
- [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>>.
- [RFC2931] Eastlake 3rd, D., "DNS Request and Transaction Signatures (SIG(0)s)", RFC 2931, DOI 10.17487/RFC2931, September 2000, <<https://www.rfc-editor.org/info/rfc2931>>.

- [RFC6973] Cooper, A., Tschofenig, H., Aboba, B., Peterson, J., Morris, J., Hansen, M., and R. Smith, "Privacy Considerations for Internet Protocols", RFC 6973, DOI 10.17487/RFC6973, July 2013, <<https://www.rfc-editor.org/info/rfc6973>>.
- [RFC7942] Sheffer, Y. and A. Farrel, "Improving Awareness of Running Code: The Implementation Status Section", BCP 205, RFC 7942, DOI 10.17487/RFC7942, July 2016, <<https://www.rfc-editor.org/info/rfc7942>>.
- [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

- [RFC5952] Kawamura, S. and M. Kawashima, "A Recommendation for IPv6 Address Text Representation", RFC 5952, DOI 10.17487/RFC5952, August 2010, <<https://www.rfc-editor.org/info/rfc5952>>.
- [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>>.
- [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>>.
- [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>>.

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

Peter van Dijk
PowerDNS.COM B.V.
Den Haag
The Netherlands

Email: peter.van.dijk@powerdns.com

Remi Gacogne
PowerDNS.COM B.V.
Den Haag
The Netherlands

Email: remi.gacogne@powerdns.com

Domain Name System Operations
Internet-Draft
Intended status: Standards Track
Expires: September 6, 2018

A. Gavrichenkov
Qrator Labs
March 05, 2018

Domain Name System Service Application Programming Interface
draft-gavrichenkov-dnsop-dnssapi-00

Abstract

Managed DNS services are widely used to maintain DNS zones. Virtually all of them have an API of some sort, in most cases an XML-RPC or JSON-RPC API, while most of them lack the support of zone transfers. The latter is unlikely to change any time soon due to the reasons outlined below. This document describes a protocol, a common denominator of existing API protocols, that both a service provider and its customer can use to exchange information about DNS zones and policies.

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 6, 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 (<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. Functionality supported by managed DNS service providers . .	3
2.1. Failover	3
2.2. Location-based DNS routing	4
2.3. Firewalling	4
2.4. Load balancing	4
2.5. Rate limiting	5
2.6. Statistics	5
3. General policy on additional extensions	5
4. DNSSAPI protocol specification	5
5. Normative References	5
Author's Address	6

1. Introduction

Today, managed DNS services are a common solution for setting up and maintaining a DNS infrastructure for an enterprise. Those services often offer convenient functionality out of the box, e.g. failover, granular load balancing or geotargeting, while being more resilient to distributed denial-of-service attacks than a simple in-house solution could be.

However, the main challenge with managed DNS services is managing them. In case there's an update in the DNS setup, an enterprise would want it to be propagated to the managed service as soon as possible. However, existing mechanisms like zone transfer [RFC5936] or dynamic updates [RFC2136] are rarely implemented by managed DNS service providers, leaving an enterprise with an uncomfortable choice of either using a Web interface to manually set up zones and policies, or using an API of that provider.

There are reasons why existing mechanisms fail to gain popularity among service providers and their customers. First, zone transfer doesn't support virtually any of the features a customer might want from a service provider, except for a trivial name resolution. For instance, it is impossible to propagate a geography-based policy towards a service provider using zone transfer itself, with accordance to standard; this can be achieved ad hoc if both a customer and a provider agree on a particular zone naming policy, however, as this is not supported by an Internet standard, it makes changing a service provider or adding a new one a touch challenge.

Second, an enterprise which is using a managed DNS service might not be operating its own primary DNS server at all, sticking with simple deployment database exports. An XML-RPC or JSON-RPC API fits that model rather well, as handlers for those are highly likely to be implemented by a personnel quite familiar with the concepts of XML and/or JSON-RPC. However, implementing a binary protocol might be viewed as another challenge.

Next, both zone transfers and dynamic updates go in one direction, while an enterprise generally might want a feedback, including, but not limited to, traffic statistics overview, average response time, query type statistics, and so on. This is, once again, usually incorporated in a managed DNS service API.

However, the main issue with the latter is that there's currently no Internet standard providing a guidance for the API design. As the result, each DNS provider implements and maintains its own API, with its own naming schemes and type layouts, once again making migration from one provider to another - or operating more than one provider simultaneously - a challenge for network and system operations departments.

This might be viewed by some as a sort of a vendor lock-in, however, this issue alone is highly unlikely to really help retaining a customer who is somehow dissatisfied with the service and is eager to change the provider. What is beyond doubt is that a customer will just be further disappointed after they will face all the projected issues while moving to another service.

This way, it might be useful to agree on a common API protocol, JSON-RPC-based, with a built-in support for all the features offered by managed DNS services today, and extensible in order to add more features in future. The purpose of this document is to provide a description of such a protocol. This protocol might then be viewed as a guidance for new DNS providers which are going to implement their API, or for existing providers refactoring their code.

2. Functionality supported by managed DNS service providers

Here is the list of features implemented by managed DNS service providers (MDNSSP) today.

2.1. Failover

Enterprises are often in a high demand for online business continuity, and as the result, they opt for some redundancy. E.g. if they operate a Web site, they often have more than one server, on

more than one IP address, serving the Web content. There are mainly two options to implement that redundancy:

- o Those servers may be put in an anycast IP prefix, announced from different locations, so that if a location goes down, its traffic is then served by nearest network locations
- o Those servers may operate simultaneously, on a round-robin basis, all being put in a DNS A record entry.

The issue with the latter approach is that one has to set up monitoring and keep-alive checks of some sort to take a failing server out of round-robin as soon as possible. MDNSSP often offer convenient built-in features to do that.

2.2. Location-based DNS routing

Geography-based DNS routing, known also as geo-balancing, is a widely used method to reduce the latency between network clients and services by looking at the IP source of a DNS query and returning an answer with an IP address which is as close to a client as possible in terms of geolocation. The distance between a client and each in the set of servers may be measured in different ways, including looking at the country a source IP address belongs to, a region or city, or even comparing latitude and longitude.

However, due to routing policies of network operators and also due to the reported inaccuracy of regional internet registries' databases (which are the only officially recognized source of the mapping between IP addresses and countries and geographic regions), there might be latency issues now with geography-based DNS routing. Some MDNSSP handle that by allowing more specific policies to be set up, e.g. ASN-based or prefix-based policies.

2.3. Firewalling

Firewall access rules might be viewed as a subset of location-based policies, except for a simpler policy of just dropping the traffic instead of processing it. However, sometimes further requirements may take place, e.g. forcing a challenge towards a source IP address, and so on. Those features are a subject of a different extension than location-based routing, being applied before it.

2.4. Load balancing

There are a lot of things a DNS server can do to balance traffic towards a set of servers. The simplest example would be shuffling answers based on their weight.

2.5. Rate limiting

An MDNSSP may limit the amount of requests coming towards a single server by returning intentionally wrong response to an A query, e.g. NXDOMAIN. This might help to keep a server running in case of a sudden traffic spike.

The exact amount of queries triggering that condition must be specified as an argument during the setup.

2.6. Statistics

Generally, an MDNSSP offers metrics regarding the overall inbound and outbound network traffic, query count, average and/or median response time, and all or some of this data for different query names, types, response codes and so on.

3. General policy on additional extensions

The API is designed to be extensible. An MDNSSP SHOULD implement functionality in a way specified by this document in case this functionality can be handled by methods described in this specification. However, an MDNSSP MAY implement its own private extension if the standard functionality doesn't fit their needs.

An extension for the DNSSAPI protocol must either follow the naming structure for the private extensions' domain or use an IANA-allocated extension name.

4. DNSSAPI protocol specification

...

5. Normative References

[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>>.

[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>>.

Author's Address

Artyom Gavrichenkov
Qrator Labs

Email: ag@qrator.net

DNS Operations
Internet-Draft
Intended status: Standards Track
Expires: January 9, 2020

T. Finch
University of Cambridge
E. Hunt
ISC
P. van Dijk
PowerDNS
A. Eden
DNSimple
W. Mekking
ISC
July 8, 2019

Address-specific DNS aliases (ANAME)
draft-ietf-dnsop-aname-04

Abstract

This document defines the "ANAME" DNS RR type, to provide similar functionality to CNAME, but only for address queries. Unlike CNAME, an ANAME can coexist with other record types. The ANAME RR allows zone owners to make an apex domain name into an alias in a standards compliant manner.

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 January 9, 2020.

Copyright Notice

Copyright (c) 2019 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
 - 1.1. Overview
 - 1.2. Terminology
2. The ANAME resource record
 - 2.1. Presentation and wire format

- 2.2. Coexistence with other types
- 3. Substituting ANAME sibling address records
- 4. ANAME processing by primary masters
 - 4.1. Zone transfers
 - 4.2. DNSSEC
 - 4.3. TTLs
- 5. ANAME processing by resolvers
- 6. Query processing
 - 6.1. Authoritative servers
 - 6.1.1. Address queries
 - 6.1.2. ANAME queries
 - 6.2. Resolvers
 - 6.2.1. Address queries
 - 6.2.2. ANAME queries
- 7. IANA considerations
- 8. Security considerations
- 9. Acknowledgments
- 10. Changes since the last revision
 - 10.1. Version -04
 - 10.2. Version -03
 - 10.3. Version -02
- 11. References
 - 11.1. Normative References
 - 11.2. Informative References
 - 11.3. URIs
- Appendix A. Implementation status
- Appendix B. Historical note
- Appendix C. On preserving TTLs
 - C.1. Query bunching
 - C.2. Upstream caches
 - C.3. ANAME chains
 - C.4. ANAME substitution inside the name server
 - C.5. TTLs and zone transfers
- Appendix D. Alternative setups
 - D.1. Reducing query volume
 - D.2. Zone transfer scalability
 - D.3. Tailored responses
- Appendix E. ANAME loops
- Authors' Addresses

1. Introduction

It can be desirable to provide web sites (and other services) at a bare domain name (such as "example.com") as well as a service-specific subdomain ("www.example.com").

If the web site is hosted by a third-party provider, the ideal way to provision its name in the DNS is using a CNAME record, so that the third party provider retains control over the mapping from names to IP address(es). It is now common for name-to-address mappings to be highly dynamic, dependent on client location, server load, etc.

However, CNAME records cannot coexist with other records with the same owner name. (The reason why is explored in Appendix B). This restriction means they cannot appear at a zone apex (such as "example.com") because of the SOA, NS, and other records that have to be present there. CNAME records can also conflict at subdomains, for example, if "department.example.edu" has separately hosted mail and web servers.

Redirecting website lookups to an alternate domain name via SRV or URI resource records would be an effective solution from the DNS point of view, but to date, browser vendors have not accepted this approach.

As a result, the only widely supported and standards-compliant way to publish a web site at a bare domain is to place address records (A

and/or AAAA) at the zone apex. The flexibility afforded by CNAME is not available.

This document specifies a new RR type "ANAME", which provides similar functionality to CNAME, but only for address queries (i.e., for type A or AAAA). The basic idea is that the address records next to an ANAME record are automatically copied from and kept in sync with the ANAME target's address records. The ANAME record can be present at any DNS node, and can coexist with most other RR types, enabling it to be present at a zone apex, or any other name where the presence of other records prevents the use of a CNAME record.

Similar authoritative functionality has been implemented and deployed by a number of DNS software vendors and service providers, using names such as ALIAS, ANAME, apex CNAME, CNAME flattening, and top-level redirection. These mechanisms are proprietary, which hinders the ability of zone owners to have the same data served from multiple providers or to move from one provider to another. None of these proprietary implementations includes a mechanism for resolvers to follow the redirection chain themselves.

1.1. Overview

The core functionality of this mechanism allows zone administrators to start using ANAME records unilaterally, without requiring secondary servers or resolvers to be upgraded.

- o The resource record definition in Section 2 is intended to provide zone data portability between standards-compliant DNS servers and the common core functionality of existing proprietary ANAME-like facilities.
- o The zone maintenance mechanism described in Section 4 keeps the ANAME's sibling address records in sync with the ANAME target.

This definition is enough to be useful by itself. However, it can be less than optimal in certain situations: for instance, when the ANAME target uses clever tricks to provide different answers to different clients to improve latency or load balancing. The query processing rules in Section 6 require to include the ANAME record so that resolvers can use this information (as described in Section 5) to obtain answers that are tailored to the resolver rather than to the zone's primary master.

Resolver support for ANAME is not necessary, since ANAME-oblivious resolvers can get working answers from authoritative servers. It's just an optimization that can be rolled out incrementally, and that will help ANAME to work better the more widely it is deployed.

1.2. Terminology

An "address record" is a DNS resource record whose type is A or AAAA. These are referred to as "address types". "Address query" refers to a DNS query for any address type.

When talking about "address records" we mean the entire RRset, including owner name and TTL. We treat missing address records (i.e. NXDOMAIN or NODATA) the same successfully resolving as a set of zero address records, and distinct from "failure" which covers error responses such as SERVFAIL or REFUSED.

The "sibling address records" of an ANAME record are the address records at the same owner name as the ANAME, which are subject to ANAME substitution.

The "target address records" of an ANAME record are the address records obtained by resolving the ultimate target of the ANAME (see

Section 3).

During the process of looking up the target address records, one or more CNAME or ANAME records may be encountered. These records are not the final target address records, and are referred in this document as "intermediate records". The target name must be replaced with the new name provided in the RDATA and the new target is resolved.

Other DNS-related terminology can be found in [RFC8499].

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. The ANAME resource record

This document defines the "ANAME" DNS resource record type, with RR TYPE value [TBD].

2.1. Presentation and wire format

The ANAME presentation format is identical to that of CNAME [RFC1033]:

```
owner ttl class ANAME target
```

The wire format is also identical to CNAME [RFC1035], except that name compression is not permitted in ANAME RDATA, per [RFC3597].

2.2. Coexistence with other types

Only one ANAME <target> can be defined per <owner>. An ANAME RRset MUST NOT contain more than one resource record.

An ANAME's sibling address records are under the control of ANAME processing (see Section 4) and are not first-class records in their own right. They MAY exist in zone files, but they can subsequently be altered by ANAME processing.

An ANAME record MAY freely coexist at the same owner name with other RR types, except they MUST NOT coexist with CNAME or any other RR type that restricts the types with which it can itself coexist. That means An ANAME record can coexist at the same owner name with A and AAAA records. These are the sibling address records that are updated with the target addresses that are retrieved through the ANAME substitution process Section 3.

Like other types, An ANAME record can coexist with DNAME records at the same owner name; in fact, the two can be used cooperatively to redirect both the owner name address records (via ANAME) and everything under it (via DNAME).

3. Substituting ANAME sibling address records

This process is used by both primary masters (see Section 4) and resolvers (see Section 5), though they vary in how they apply the edit described in the final step. However, this process is not exclusively used by primary masters and resolvers: it may be executed as a bump in the wire, as part of the query lookup, or at any other point during query resolution.

The following steps MUST be performed for each address type:

1. Starting at the ANAME owner, follow the chain of ANAME and/or CNAME records as far as possible to find the ultimate target.

2. If a loop is detected, continue with an empty RRset, otherwise get the ultimate target's address records. (Ignore any sibling address records of intermediate ANAMES.)
3. Stop if resolution failed. (Note that NXDOMAIN and NODATA count as successfully resolving an empty RRset.)
4. If one or more address records are found, replace the owner of the target address records with the owner of the ANAME record. Set the TTL to the minimum of the ANAME TTL, the TTL of each intermediate record, and the TTL of the target address records. Drop any RRSIG records.
5. Stop if this modified RRset is the same as the sibling RRset (ignoring any RRSIG records). The comparison MAY treat nearly-equal TTLs as the same.
6. Delete the sibling address RRset (if any) and replace it with the modified RRset.

At this point, the substituted RRset is not signed. A primary master will proceed to sign the substituted RRset, whereas resolvers can only use the substituted RRset when an unsigned answer is appropriate. This is explained in more detail in the following sections.

4. ANAME processing by primary masters

Each ANAME's sibling address records are kept up-to-date as if by the following process, for each address type:

- o Perform ANAME sibling address record substitution as described in Section 3. Any edit performed in the final step is applied to the ANAME's zone. A primary server MAY use Dynamic Updates (DNS UPDATE) [RFC2136] to update the zone.
- o If resolution failed, wait for a period before trying again. This retry time SHOULD be configurable.
- o Otherwise, wait until the target address RRset TTL has expired or is close to expiring, then repeat.

It may be more efficient to manage the polling per ANAME target rather than per ANAME as specified (for example if the same ANAME target is used by multiple zones).

Sibling address records are committed to the zone and stored in nonvolatile storage. This allows a server to restart without delays due to ANAME processing, use offline DNSSEC signing, and not implement special ANAME processing logic when handling a DNS query.

Appendix D describes how ANAME would fit in different DNS architectures that use online signing or tailored responses.

4.1. Zone transfers

ANAME is no more special than any other RRtype and does not introduce any special processing related to zone transfers.

A zone containing ANAME records that point to frequently-changing targets will itself change frequently, and may see an increased number of zone transfers. Or if a very large number of zones are sharing the same ANAME target, and that changes address, that may cause a great volume of zone transfers. Guidance on dealing with ANAME in large scale implementations is provided Appendix D.

Secondary servers rely on zone transfers to obtain sibling address

records, just like the rest of the zone, and serve them in the usual way (see Section 6). A working DNS NOTIFY [RFC1996] setup is recommended to avoid extra delays propagating updated sibling address records when they change.

4.2. DNSSEC

A zone containing ANAME records that will update address records has to do so before signing the zone with DNSSEC [RFC4033] [RFC4034] [RFC4035]. This means that for traditional DNSSEC signing the substitution of sibling address records must be done before signing and loading the zone into the name server. For servers that support online signing, the substitution may happen as part of the name server process, after loading the zone.

DNSSEC signatures on sibling address records are generated in the same way as for normal (dynamic) updates.

4.3. TTLs

Sibling address records are served from authoritative servers with a fixed TTL. Normally this TTL is expected to be the same as the target address records' TTL; however the exact mechanism for obtaining the target is unspecified, so cache effects, following ANAME and CNAME chains, or deliberate policies might make the sibling TTL smaller.

This means that when adding address records into the zone as a result of ANAME processing, the TTL to use is at most that of the TTL of the address target records. If you use a higher value, this will stretch the TTL which is undesired.

TTL stretching is hard to avoid when implementing ANAME substitution at the primary: The target address records' TTL influences the update rate of the zone, while the sibling address records' TTL determine how long a resolver may cache the address records. Thus, the end-to-end TTL (from the authoritative servers for the target address records to end-user DNS caches) is nearing twice the target address record TTL. There is a more extended discussion of TTL handling in Appendix C.

5. ANAME processing by resolvers

When a resolver makes an address query in the usual way, it might receive a response containing ANAME information in the Answer section, as described in Section 6. This informs the resolver that it MAY resolve the ANAME target address records to get answers that are tailored to the resolver rather than the ANAME's primary master.

In order to provide tailored answers to clients that are ANAME-oblivious, the resolver MAY perform sibling address record substitution in the following situations:

- o The resolver's client queries with DO=0. (As discussed in Section 8, if the resolver finds it would downgrade a secure answer to insecure, it MAY choose not to substitute the sibling address records.)
- o The resolver's client queries with DO=1 and the ANAME and sibling address records are unsigned. (Note that this situation does not apply when the records are signed but insecure: the resolver might not be able to validate them because of a broken chain of trust, but its client could have an extra trust anchor that does allow it to validate them; if the resolver substitutes the sibling address records they will become bogus.)

In these first two cases, the resolver MAY perform ANAME sibling

address record substitution as described in Section 3. Any edit performed in the final step is applied to the Answer section of the response.

If the resolver's client is querying using an API such as "getaddrinfo" [RFC3493] that does not support DNSSEC validation, the resolver MAY perform ANAME sibling address record substitution as described in Section 3. Any edits performed in the final step are applied to the addresses returned by the API. (This case is for validating stub resolvers that query an upstream recursive server with DO=1, so they cannot rely on the recursive server to do ANAME substitution for them.)

6. Query processing

6.1. Authoritative servers

6.1.1. Address queries

When a server receives an address query for a name that has an ANAME record, the response's Answer section MUST contain the ANAME record, in addition to the sibling address queries. The ANAME record indicates to a client that it might wish to resolve the target address records itself.

6.1.2. ANAME queries

When a server receives an query for type ANAME, regardless of whether the ANAME record exists on the queried domain, any sibling address records SHOULD be added to the Additional section. Note that the sibling address records may have been substituted already.

When adding address records to the Additional section, if not all address types are present and the zone is signed, the server SHOULD include a DNSSEC proof of nonexistence for the missing address types.

6.2. Resolvers

6.2.1. Address queries

When a server receives an address query for a name that has an ANAME record, the response's Answer section MUST contain the ANAME record, in addition to the sibling address queries.

The Additional section MAY contain the target address records that match the query type (or the corresponding proof of nonexistence), if they are available in the cache and the target address RDATA fields differ from the sibling address RRset.

An ANAME target MAY resolve to address records via a chain of CNAME and/or ANAME records; any CNAME/ANAME chain MUST be included when adding target address records to a response's Additional section.

6.2.2. ANAME queries

When a resolver receives an query for type ANAME, any sibling address records SHOULD be added to the Additional section. Just like with an authoritative server, when adding address records to the Additional section, if not all address types are present and the zone is signed, the resolver SHOULD include a DNSSEC proof of nonexistence for the missing address types.

7. IANA considerations

IANA is requested to assign a DNS RR TYPE value for ANAME resource records under the "Resource Record (RR) TYPEs" subregistry under the "Domain Name System (DNS) Parameters" registry.

IANA might wish to consider the creation of a registry of address types; addition of new types to such a registry would then implicitly update this specification.

8. Security considerations

When a primary master updates an ANAME's sibling address records to match its target address records, it uses its own best information as to the correct answer. The primary master might sign the updated records, but that is not a guarantee of the actual correctness of the answer. This signing can have the effect of promoting an insecure response from the ANAME <target> to a signed response from the <owner>, which can then appear to clients to be more trustworthy than it should. DNSSEC validation SHOULD be used when resolving the ANAME <target> to mitigate this possible harm. Primary masters MAY refuse to substitute ANAME sibling address records unless the <target> node is both signed and validated.

When a resolver substitutes an ANAME's sibling address records, it can find that the sibling address records are secure but the target address records are insecure. Going ahead with the substitution will downgrade a secure answer to an insecure one. However this is likely to be the counterpart of the situation described in the previous paragraph, so the resolver is downgrading an answer that the ANAME's primary master upgraded. A resolver will only downgrade an answer in this way when its client is security-oblivious; however the client's path to the resolver is likely to be practically safer than the resolver's path to the ANAME target's servers. Resolvers MAY choose not to substitute sibling address records when they are more secure than the target address records.

9. Acknowledgments

Thanks to Mark Andrews, Ray Bellis, Stefan Buehler, Paul Ebersman, Richard Gibson, Tatuya JINMEI, Hakan Lindqvist, Mattijs Mekking, Stephen Morris, Bjorn Mott, Richard Salts, Mukund Sivaraman, Job Snijders, Jan Vcelak, Paul Vixie, Duane Wessels, and Paul Wouters, Olli Vanhoja, Brian Dickson for discussion and feedback.

10. Changes since the last revision

[This section is to be removed before publication as an RFC.]

The full history of this draft and its issue tracker can be found at <https://github.com/each/draft-aname> [1]

10.1. Version -04

- o Split up section about Additional Section processing.
- o Update Additional Section processing requirements.
- o Clarify when ANAME resolution may happen [#43].
- o Revisit TTL considerations [#30, #34].
- o ANAME goes into the Answer section when QTYPE=A|AAAA [#62].
- o Update alternative setups section with concerns (Brian Dickson) [#68].
- o Add section on ANAME loops (open issue [#45]).

10.2. Version -03

- o Grammar improvements (Olli Vanhoja)

- o Split up Implications section, clarify text on zone transfers and dynamic updates [#39].
- o Rewrite Alternative setup section and move to Appendix, add text on zone transfer scalability concerns and GeoIP.

10.3. Version -02

Major revamp, so authoritative servers (other than primary masters) now do not do any special ANAME processing, just Additional section processing.

11. References

11.1. Normative References

- [RFC1033] Lottor, M., "Domain Administrators Operations Guide", RFC 1033, DOI 10.17487/RFC1033, November 1987, <<https://www.rfc-editor.org/info/rfc1033>>.
- [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>>.
- [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>>.
- [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>>.
- [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>>.
- [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>>.
- [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>>.

11.2. Informative References

- [RFC0882] Mockapetris, P., "Domain names: Concepts and facilities", RFC 882, DOI 10.17487/RFC0882, November 1983,

<<https://www.rfc-editor.org/info/rfc882>>.

- [RFC0973] Mockapetris, P., "Domain system changes and observations", RFC 973, DOI 10.17487/RFC0973, January 1986, <<https://www.rfc-editor.org/info/rfc973>>.
- [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>>.
- [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>>.
- [RFC2065] Eastlake 3rd, D. and C. Kaufman, "Domain Name System Security Extensions", RFC 2065, DOI 10.17487/RFC2065, January 1997, <<https://www.rfc-editor.org/info/rfc2065>>.
- [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>>.
- [RFC3493] Gilligan, R., Thomson, S., Bound, J., McCann, J., and W. Stevens, "Basic Socket Interface Extensions for IPv6", RFC 3493, DOI 10.17487/RFC3493, February 2003, <<https://www.rfc-editor.org/info/rfc3493>>.

11.3. URIs

[1] <https://github.com/each/draft-aname>

[2] <https://github.com/each/draft-aname/issues/45>

Appendix A. Implementation status

PowerDNS currently implements a similar authoritative-only feature using "ALIAS" records, which are expanded by the primary server and transferred as address records to secondaries.

[TODO: Add discussion of DNSimple, DNS Made Easy, EasyDNS, Cloudflare, Amazon, Dyn, and Akamai.]

Appendix B. Historical note

In the early DNS [RFC0882], CNAME records were allowed to coexist with other records. However this led to coherency problems: if a resolver had no cache entries for a given name, it would resolve queries for un-cached records at that name in the usual way; once it had cached a CNAME record for a name, it would resolve queries for un-cached records using CNAME target instead.

For example, given the zone contents below, the original CNAME behaviour meant that if you asked for "alias.example.com TXT" first, you would get the answer "owner", but if you asked for "alias.example.com A" then "alias.example.com TXT" you would get the answer "target".

alias.example.com.	TXT	"owner"
alias.example.com.	CNAME	canonical.example.com.
canonical.example.com.	TXT	"target"
canonical.example.com.	A	192.0.2.1

This coherency problem was fixed in [RFC0973] which introduced the inconvenient rule that a CNAME acts as an alias for all other RR types at a name, which prevents the coexistence of CNAME with other records.

A better fix might have been to improve the cache's awareness of which records do and do not coexist with a CNAME record. However that would have required a negative cache mechanism which was not added to the DNS until later [RFC1034] [RFC2308].

While [RFC2065] relaxed the restriction by allowing coexistence of CNAME with DNSSEC records, this exception is still not applicable to other resource records. RRSIG and NSEC exist to prove the integrity of the CNAME record; they are not intended to associate arbitrary data with the domain name. DNSSEC records avoid interoperability problems by being largely invisible to security-oblivious resolvers.

Now that the DNS has negative caching, it is tempting to amend the algorithm for resolving with CNAME records to allow them to coexist with other types. Although an amended resolver will be compatible with the rest of the DNS, it will not be of much practical use because authoritative servers which rely on coexisting CNAMEs will not interoperate well with older resolvers. Practical experiments show that the problems are particularly acute when CNAME and MX try to coexist.

Appendix C. On preserving TTLs

An ANAME's sibling address records are in an unusual situation: they are authoritative data in the owner's zone, so from that point of view the owner has the last say over what their TTL should be; on the other hand, ANAMES are supposed to act as aliases, in which case the target should control the address record TTLs.

However there are some technical constraints that make it difficult to preserve the target address record TTLs.

The following subsections conclude that the end-to-end TTL (from the authoritative servers for the target address records to end-user DNS caches) is nearing twice the target address record TTL.

C.1. Query bunching

If the times of end-user queries for a domain name are well distributed, then (typically) queries received by the authoritative servers for that domain are also well distributed. If the domain is popular, a recursive server will re-query for it once every TTL seconds, but the periodic queries from all the various recursive servers will not be aligned, so the queries remain well distributed.

However, imagine that the TTLs of an ANAME's sibling address records are decremented in the same way as cache entries in recursive servers. Then all the recursive servers querying for the name would try to refresh their caches at the same time when the TTL reaches zero. They would become synchronized, and all the queries for the domain would be bunched into periodic spikes.

This specification says that ANAME sibling address records have a normal fixed TTL derived from (e.g. equal or nearly equal to) the target address records' original TTL. There is no cache-like decrementing TTL, so there is no bunching of queries.

C.2. Upstream caches

There are two straightforward ways to get an RRset's original TTL:

- o by directly querying an authoritative server;
- o using the original TTL field from the RRset's RRSIG record(s).

However, not all zones are signed, and a primary master might not be able to query other authoritative servers directly (e.g. if it is a

hidden primary behind a strict firewall). Instead it might have to obtain an ANAME's target address records via some other recursive server.

Querying via a separate recursive server means the primary master cannot trivially obtain the target address records' original TTLs. Fortunately this is likely to be a self-correcting problem for similar reasons to the query-bunching discussed in the previous subsection. The primary master can inspect the target address records just after the TTL expires when its upstream cache has just refreshed them, so the TTL will be nearly equal to the original TTL.

A related consideration is that the primary master cannot in general refresh its copies of an ANAME's target address records more frequently than their TTL, without privileged control over its resolver cache.

Combined with the requirement that sibling address records are served with a fixed TTL, this means that the end-to-end TTL will be the target address record TTL (which determines when the sibling address records are updated) plus the sibling address record TTL (which determines when end-user caches are updated). Since the sibling address record TTL is derived from the target address records' original TTL, the end-to-end TTL will be nearing twice the target address record TTL.

C.3. ANAME chains

ANAME sibling address record substitution is made slightly more complicated by the requirement to follow chains of ANAME and/or CNAME records. The TTL of the substituted address records is the minimum of TTLs of the ANAME, all the intermediate records, and target records. This stops the end-to-end TTL from being inflated by each ANAME in the chain.

With CNAME records, repeat queries for "cname.example. CNAME target.example." must not be fully answered from cache after its TTL expires, but must instead be sent to name servers authoritative for "cname.example" in case the CNAME has been updated or removed. Similarly, an ANAME at "aname.example" means that repeat queries for "aname.example" must not be fully answered from cache after its TTL expire, but must instead be sent to name servers authoritative for aname.example in case the ANAME has been updated or removed.

C.4. ANAME substitution inside the name server

When ANAME substitution is performed inside the authoritative name server (as described in #alternatives) or in the resolver (as described in #resolver) the end-to-end TTL will actually be just the target address record TTL.

An authoritative server that has control over its resolver can use a cached target address RRset and decremented TTL in the response to the client rather than using the original target address records' TTL. It SHOULD however not use TTLs in the response that are nearing zero to avoid query bunching Appendix C.1.

A resolver that performs ANAME substitution is able to get the original TTL from the authoritative name server and use its own cache to store the substituted address records with the appropriate TTL, thereby honoring the TTL of target address records.

C.5. TTLs and zone transfers

When things are working properly (with secondary name servers responding to NOTIFY messages promptly) the authoritative servers will follow changes to ANAME target address records according to

their TTLs. As a result the end-to-end TTL is unchanged from the previous subsection.

If NOTIFY doesn't work, the TTLs can be stretched by the zone's SOA refresh timer. More serious breakage can stretch them up to the zone expiry time.

Appendix D. Alternative setups

If you are a large scale DNS provider, ANAME may introduce some operational concerns.

D.1. Reducing query volume

When doing ANAME target lookups, an authoritative server might want to use longer TTLs to reduce query volume, for ANAME values that do not change frequently. This is the same concern a recursive resolver may be exposed to when receiving answers with short TTLs. An authoritative server doing ANAME target lookups therefor could use the same mitigation as a recursive nameserver, that is set a configured minimum TTL usage. This may however contribute to TTL stretching as described in Section 4.3 so the configured minimum should not be too low.

D.2. Zone transfer scalability

A frequently changing ANAME target, or a ANAME target that changes its address and is used for many zones, can lead to an increased number of zone transfers. Such DNS architectures may want to consider a zone transfer mechanism outside the DNS.

Another way to deal with zone transfer scalability is to move the ANAME processing (Section 3) inside the name server daemon. This is not a requirement for ANAME to work, but may be a better solution in large scale implementations. These implementations usually already rely on online DNSSEC signing for similar reasons. If ANAME processing occurs inside the name server daemon, it MUST be done before any DNSSEC online signing happens.

For example, some existing ANAME-like implementations are based on a DNS server architecture, in which a zone's published authoritative servers all perform the duties of a primary master in a distributed manner: provisioning records from a non-DNS back-end store, refreshing DNSSEC signatures, and so forth. They don't use standard zone transfers, and already implement their ANAME-like processing inside the name server daemon, substituting ANAME sibling address records on demand.

D.3. Tailored responses

Some DNS providers will tailor responses based on information in the client request. Such implementations will use the source IP address or EDNS Client Subnet [RFC7871] information and use geographical data (GeoIP) or network latency measurements to decide what the best answer is for a given query. Such setups won't work with traditional DNSSEC and provide DNSSEC support usually through online signing. Similar such setups should provide ANAME support through substituting ANAME sibling records on demand.

Also, an authoritative server that uses the client address to tailor the response should obviously not use its own address when looking up ANAME targets, or it could direct clients to a suboptimal server (e.g. a wrong language, or regional restricted content). Instead the authoritative server should look up the ANAME targets on behalf of the client address. It could use for example EDNS Client Subnet for this.

In short, the exact mechanism for obtaining the target address records in such setups is unspecified; typically they will be resolved in the DNS in the usual way, but if an ANAME implementation has special knowledge of the target it can short-cut the substitution process, or it can use clever tricks such as client-dependant answers to make the answer more optimal.

Appendix E. ANAME loops

The ANAME sibling address substitution algorithm in Section 3 poses a challenge of detecting a loop between two or more ANAME records. Imagine this setup: two authoritative servers X and Y performing ANAME sibling address substitution on the fly (i.e. they attempt to resolve the ANAME target when the client query arrives). If server X gets a query for FOO.TEST which is an ANAME to BAR.TEST, it will send a query to server Y for BAR.TEST which is an ANAME to FOO.TEST. Server Y will then start a new query to server X, which has no way to know that it is regarding the original FOO.TEST lookup.

The only indicator of the presence of the loop in the described setup is the network timeout. Ideally we would recognize the loop explicitly based on the exchanged DNS messages.

On-the-fly ANAME substitution is allowed and it's just the most obvious scenario where the problem can be demonstrated, but this loop can also be encountered in other situations. The root cause is that when the server gets a query it doesn't know why and that the server always attempts to fully resolve the ANAME target before sending the response.

TODO: Solve this issue [<https://github.com/each/draft-aname/issues/45> [2]]

Authors' Addresses

Tony Finch
University of Cambridge
University Information Services
Roger Needham Building
7 JJ Thomson Avenue
Cambridge CB3 0RB
England

Email: dot@dotat.at

Evan Hunt
ISC
950 Charter St
Redwood City, CA 94063
USA

Email: each@isc.org

Peter van Dijk
PowerDNS.COM B.V.
Den Haag
The Netherlands

Email: peter.van.dijk@powerdns.com

Anthony Eden
DNSimple
Boston, MA USA

Email: anthony.eden@dnsimple.com
URI: <https://dnsimple.com/>

Matthijs Mekking
ISC
950 Charter St
Redwood City, CA 94063
USA

Email: matthijs@isc.org

dnsop
Internet-Draft
Intended status: Standards Track
Expires: June 15, 2019

J. Dickinson
J. Hague
S. Dickinson
Sinodun IT
T. Manderson
J. Bond
ICANN
December 12, 2018

C-DNS: A DNS Packet Capture Format
draft-ietf-dnsop-dns-capture-format-10

Abstract

This document describes a data representation for collections of DNS messages. The format is designed for efficient storage and transmission 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 metadata. It is intended to assist with the development of DNS traffic monitoring applications.

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 15, 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. Terminology	5
3. Data collection use cases	5
4. Design considerations	7
5. Choice of CBOR	9
6. C-DNS format conceptual overview	9
6.1. Block Parameters	13
6.2. Storage Parameters	13
6.2.1. Optional data items	14
6.2.2. Optional RRs and OPCODES	15
6.2.3. Storage flags	15
6.2.4. IP Address storage	16
7. C-DNS format detailed description	16
7.1. Map quantities and indexes	16
7.2. Tabular representation	17
7.3. "File"	18
7.4. "FilePreamble"	18
7.4.1. "BlockParameters"	19
7.4.2. "CollectionParameters"	22
7.5. "Block"	24
7.5.1. "BlockPreamble"	24
7.5.2. "BlockStatistics"	25
7.5.3. "BlockTables"	26
7.6. "QueryResponse"	32
7.6.1. "ResponseProcessingData"	34
7.6.2. "QueryResponseExtended"	34
7.7. "AddressEventCount"	35
7.8. "MalformedMessage"	36
8. Versioning	37
9. C-DNS to PCAP	37
9.1. Name compression	38
10. Data collection	39
10.1. Matching algorithm	40
10.2. Message identifiers	42
10.2.1. Primary ID (required)	42
10.2.2. Secondary ID (optional)	43
10.3. Algorithm parameters	43
10.4. Algorithm requirements	43
10.5. Algorithm limitations	43

10.6.	Workspace	44
10.7.	Output	44
10.8.	Post processing	44
11.	Implementation guidance	44
11.1.	Optional data	45
11.2.	Trailing bytes	45
11.3.	Limiting collection of RDATA	45
11.4.	Timestamps	45
12.	Implementation status	46
12.1.	DNS-STATS Compactor	46
13.	IANA considerations	47
13.1.	Transport types	47
13.2.	Data storage flags	48
13.3.	Response processing flags	48
13.4.	AddressEvent types	49
14.	Security considerations	49
15.	Privacy considerations	50
16.	Acknowledgements	50
17.	Changelog	51
18.	References	54
18.1.	Normative References	54
18.2.	Informative References	55
18.3.	URIs	57
Appendix A.	CDDL	58
Appendix B.	DNS Name compression example	68
B.1.	NSD compression algorithm	69
B.2.	Knot Authoritative compression algorithm	70
B.3.	Observed differences	70
Appendix C.	Comparison of Binary Formats	70
C.1.	Comparison with full PCAP files	73
C.2.	Simple versus block coding	74
C.3.	Binary versus text formats	74
C.4.	Performance	74
C.5.	Conclusions	75
C.6.	Block size choice	75
Authors' Addresses		76

1. Introduction

There has long been a need for server operators 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 "day in the life" (DITL) [ditl] analysis.

A wide variety of tools already exist that facilitate the collection of DNS traffic data, such as DSC [dsc], packetq [packetq], dnscap [dnscap] and dnstap [dnstap]. However, there is no standard exchange

format for large DNS packet captures. The PCAP [pcap] or PCAP-NG [pcapng] formats are typically used in practice for packet captures, but these file formats can contain a great deal of additional information that is not directly pertinent to DNS traffic analysis and thus unnecessarily increases the capture file size. Additionally these tools and formats typically have no filter mechanism to selectively record only certain fields at capture time, requiring post-processing for anonymization or pseudonymization of data to protect user privacy.

There has also been work on using text based formats to describe DNS packets such as [I-D.daley-dnsxml], [RFC8427], but these are largely aimed at producing convenient representations of single messages.

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

The format described in this document, C-DNS (Compacted-DNS), focusses on the problem of capturing and storing large packet capture files of DNS traffic with the following goals in mind:

- o Minimize the file size for storage and transmission.
- o Minimize the overhead of producing the packet capture file and the cost of any further (general purpose) compression of the file.

This document contains:

- o A discussion of some common use cases in which 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 description of why CBOR [RFC7049] was chosen for this format, see Section 5.
- o A conceptual overview of the C-DNS format, see Section 6.
- o The definition of the C-DNS format for the collection of DNS messages, see Section 7.
- o Notes on converting C-DNS data to PCAP format, see Section 9.

- o Some high level implementation considerations for applications designed to produce C-DNS, see Section 10.

2. 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.

"Packet" refers to an individual IPv4 or IPv6 packet. Typically packets are UDP datagrams, but may also be part of a TCP data stream. "Message", unless otherwise qualified, refers to a DNS payload extracted from a UDP datagram or a TCP data stream.

The parts of DNS messages are named as they are in [RFC1035]. Specifically, the DNS message has five sections: Header, Question, Answer, Authority, and Additional.

Pairs of DNS messages are called a Query and a Response.

3. Data collection use cases

From a purely server operator perspective, collecting full packet captures of all packets going in or out of a name server provides the most comprehensive picture of network activity. However, there are several design choices or other limitations that are common to many DNS installations and operators.

- o DNS servers are hosted in a variety of situations:
 - * 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 DoS attack

- * On servers that are unwitting intermediaries in DoS attacks
- o Traffic can be collected via a variety of mechanisms:
 - * Within the name server implementation itself
 - * On the same hardware as the name server itself
 - * Using a network tap on an adjacent host to listen to DNS traffic
 - * Using port mirroring to listen from another host
- 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 being on the in-band network

Thus, 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 infeasible to reliably collect full packet captures, especially if the server is under attack. In the latter case, collection of full packet captures may be reasonable.

As a result of these restrictions, the C-DNS data format is 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 can be multiple third party servers involved

Because of these considerations, a major factor in the design of the format is minimal storage size of the capture files.

Another significant 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 are the most important jobs of a name server; capturing data is not. Any data collection system co-located with the name server needs to be intelligent enough to carefully manage its CPU, disk, memory and network utilization. This leads to designing a format that requires a relatively low overhead to produce and minimizes the requirement for further potentially costly compression.

However, it is also essential that interoperability with less restricted infrastructure is maintained. In particular, it is highly desirable that the collection format should facilitate the re-creation of common formats (such as PCAP) 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.

1. 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 Question section from the response, if present, as an identifying QNAME).

- * Rationale: A Query and Response represents the basic level of a client's interaction with the server. Also, combining the Query and Response into one item often reduces storage requirements due to commonality in the data of the two messages.

In the context of generating a C-DNS file it is assumed that only those DNS payloads which can be parsed to produce a well-formed DNS message are stored in the C-DNS format and that all other messages will be (optionally) recorded as malformed messages. Parsing a well-formed message means as a minimum:

- * The packet has a well-formed 12 byte DNS Header with a recognised OPCODE.
- * The section counts are consistent with the section contents.

- * All of the resource records can be fully parsed.
2. All top level fields in each Q/R data item will be optional.
 - * Rationale: Different operators will have different requirements for data to be available for analysis. Operators with minimal requirements should not have to pay the cost of recording full data, though this will limit the ability to perform certain kinds of data analysis and also to reconstruct packet captures. For example, omitting the resource records from a Response will reduce the C-DNS file size; in principle responses can be synthesized if there is enough context. Operators may have different policies for collecting user data and can choose to omit or anonymize certain fields at capture time e.g. client address.
 3. Multiple Q/R data items will be collected into blocks in the format. Common data in a block will be abstracted and referenced from individual Q/R data items by indexing. The maximum number of Q/R data items in a block will be configurable.
 - * Rationale: This blocking and indexing provides a significant reduction in the volume of file data generated. Although this introduces complexity, it provides compression of the data that makes use of knowledge of the DNS message structure.
 - * It is anticipated that the files produced can 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 C.2.
 - * Examples of commonality between DNS messages are that in most cases the QUESTION RR is the same in the query and response, and that there is a finite set of query signatures (based on a subset of attributes). For many authoritative servers there is very likely to be a finite set of responses that are generated, of which a large number are NXDOMAIN.
 4. Traffic metadata can optionally be included in each block. Specifically, counts of some types of non-DNS packets (e.g. ICMP, TCP resets) sent to the server may be of interest.
 5. The wire format content of malformed DNS messages may optionally be recorded.
 - * Rationale: Any structured capture format that does not capture the DNS payload byte for byte will be limited to some extent

in that it cannot represent malformed DNS messages. Only those messages that can be fully parsed and transformed into the structured format can be fully represented. Note, however, this can result in rather misleading statistics. For example, a malformed query which cannot be represented in the C-DNS format will lead to the (well formed) DNS responses with error code FORMERR appearing as 'unmatched'. Therefore it can greatly aid downstream analysis to have the wire format of the malformed DNS messages available directly in the C-DNS file.

5. 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 thus is economical in storage space.
- o Other binary representations were investigated, and whilst all had attractive features, none had a significant advantage over CBOR. See Appendix C for some discussion of this.
- o CBOR is an IETF specification and familiar to IETF participants. It is based on the now-common ideas of lists and objects, and thus requires very little familiarization for those in the wider industry.
- o CBOR is a simple format, and can easily be implemented from scratch if necessary. More complex formats require library support which may present problems on unusual platforms.
- o CBOR can also be easily converted to text formats such as JSON ([RFC8259]) for debugging and other human inspection requirements.
- o CBOR data schemas can be described using CDDL [I-D.ietf-cbor-cddl].

6. C-DNS format 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 1 shows the C-DNS format at the top level including the file header and data blocks. The Query/Response data items, Address/Event

Count data items and Malformed Message data items link to various Block tables.

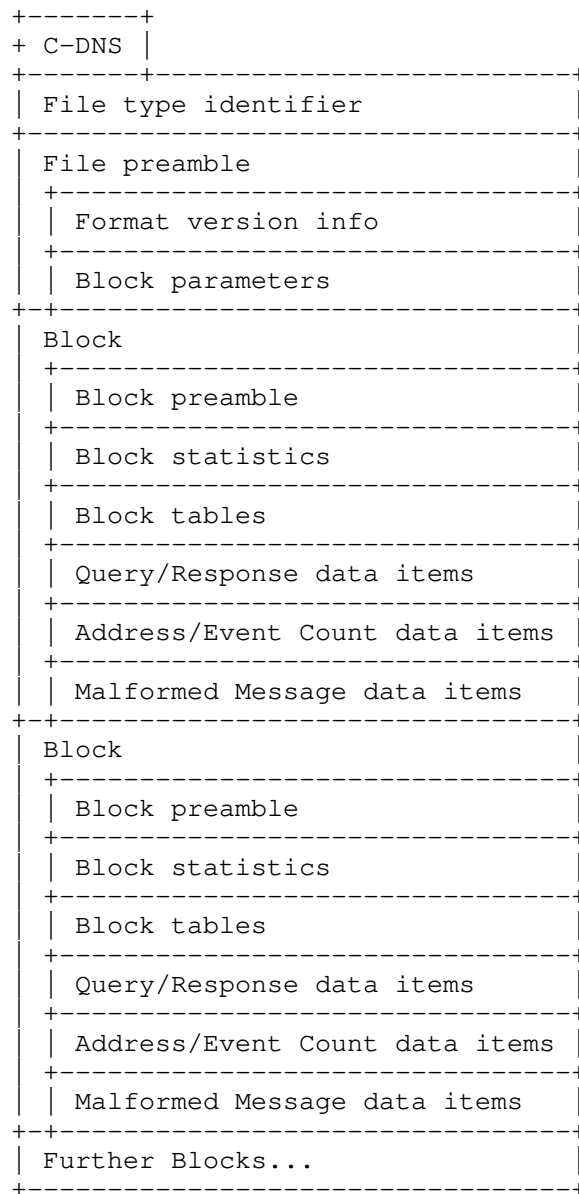
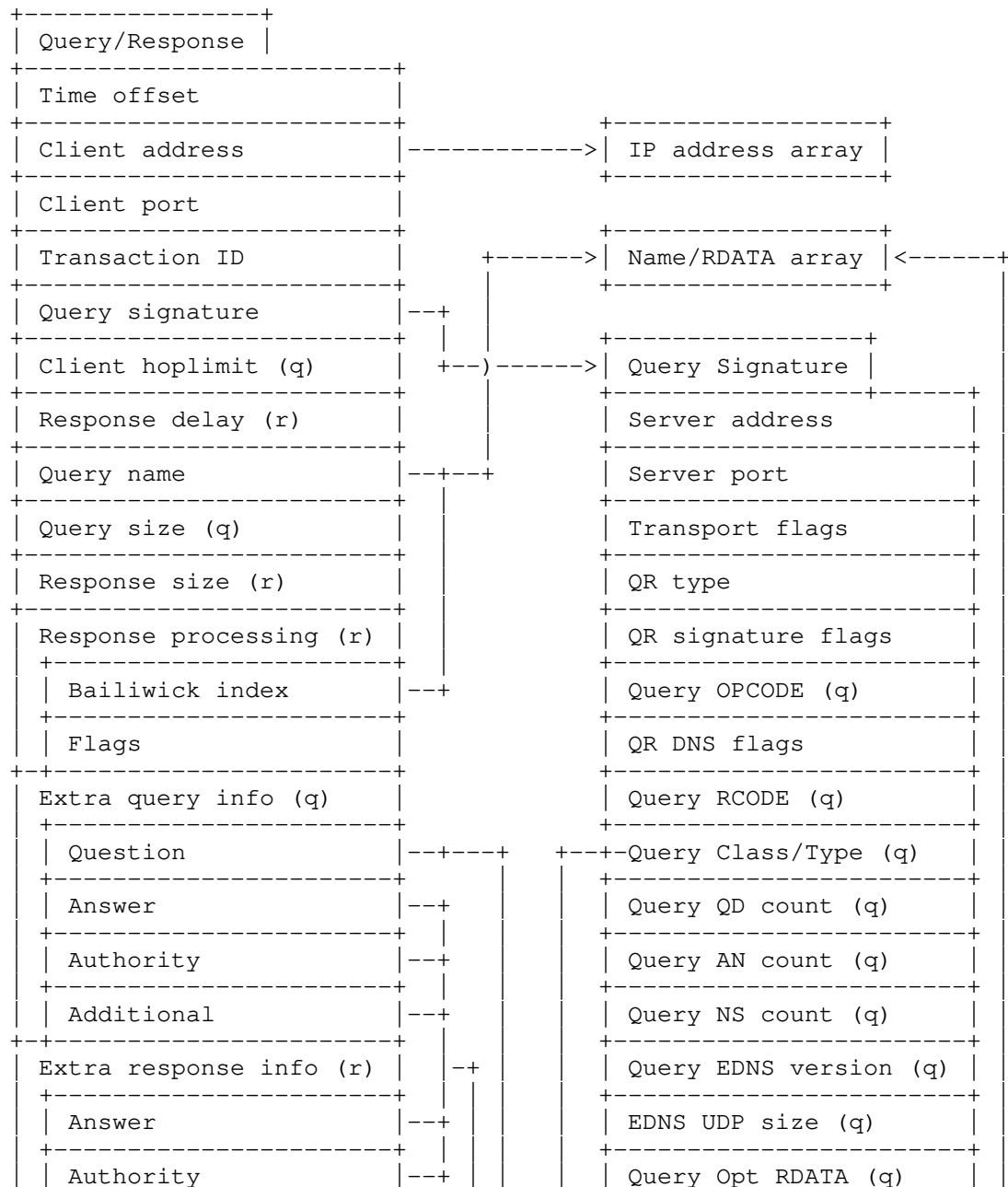


Figure 1: The C-DNS format.

Figure 2 shows some more detailed relationships within each block, specifically those between the Query/Response data item and the relevant Block tables.



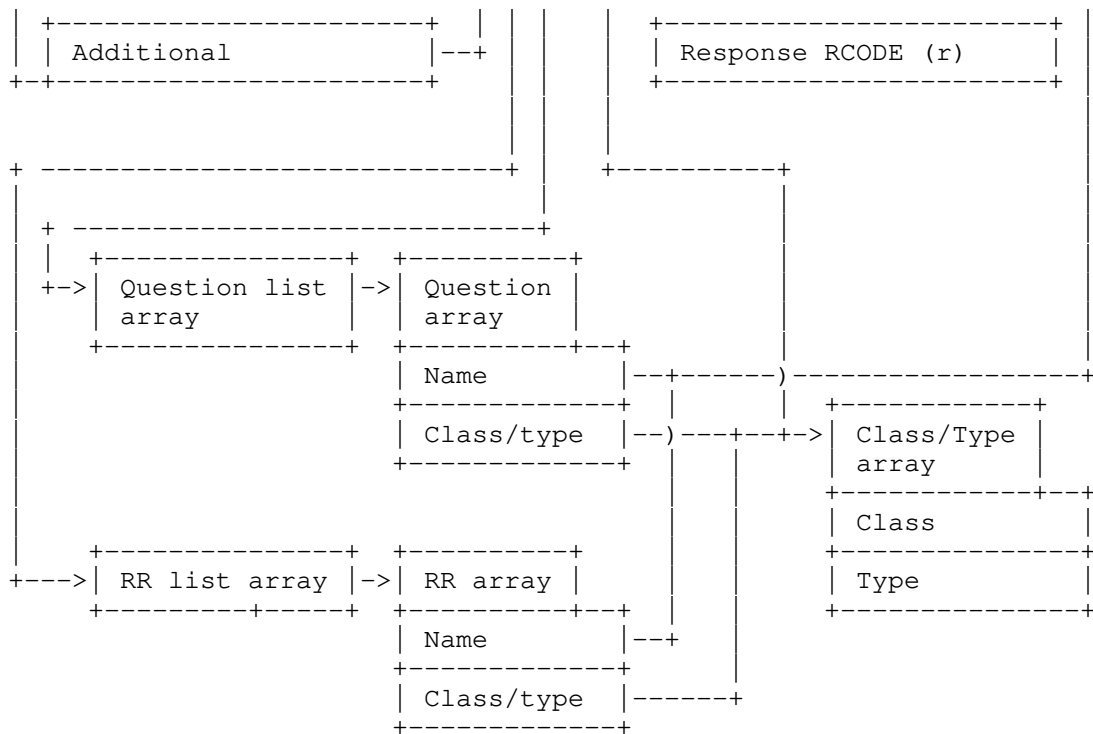


Figure 2: The Query/Response data item and subsidiary tables.

In Figure 2 data items annotated (q) are only present when a query/response has a query, and those annotated (r) are only present when a query/response response is present.

A C-DNS file begins with a file header containing a File Type Identifier and a File Preamble. The File Preamble contains information on the file Format Version and an array of Block Parameters items (the contents of which include Collection and Storage Parameters used for one or more blocks).

The file header is followed by a series of data Blocks.

A Block consists of a Block Preamble item, some Block Statistics for the traffic stored within the Block and then various arrays of common data collectively called the Block Tables. This is then followed by an array of the Query/Response data items detailing the queries and responses stored within the Block. The array of Query/Response data items is in turn followed by the Address/Event Counts data items (an array of per-client counts of particular IP events) and then

Malformed Message data items (an array of malformed messages that stored in the Block).

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 up to 10,000 Q/R data items give good results. See Appendix C.6 for more details.

This design exploits data commonality and block based storage to minimise the C-DNS file size. As a result C-DNS cannot be streamed below the level of a block.

6.1. Block Parameters

The details of the Block Parameters items are not shown in the diagrams but are discussed here for context.

An array of Block Parameters items is stored in the File Preamble (with a minimum of one item at index 0); a Block Parameters item consists of a collection of Storage and Collection Parameters that applies to any given Block. An array is used in order to support use cases such as wanting to merge C-DNS files from different sources. The Block Preamble item then contains an optional index for the Block Parameters item that applies for that Block; if not present the index defaults to 0. Hence, in effect, a global Block Parameters item is defined which can then be overridden per Block.

6.2. Storage Parameters

The Block Parameters item includes a Storage Parameters item - this contains information about the specific data fields stored in the C-DNS file.

These parameters include:

- o The sub-second timing resolution used by the data.
- o Information (hints) on which optional data are omitted. See Section 6.2.1.
- o Recorded OPCODES [opcodes] and RR types [rrtypes]. See Section 6.2.2.
- o Flags indicating, for example, whether the data is sampled or anonymized. See Section 6.2.3 and Section 15.
- o Client and server IPv4 and IPv6 address prefixes. See Section 6.2.4

6.2.1. Optional data items

To enable implementations to store data to their precise requirements in as space-efficient manner as possible, all fields in the following arrays are optional:

- o Query/Response
- o Query Signature
- o Malformed messages

In other words, an implementation can choose to omit any data item that is not required for its use case. In addition, implementations may be configured to not record all RRs, or only record messages with certain OPCODES.

This does, however, mean that a consumer of a C-DNS file faces two problems:

1. How can it quickly determine if a file definitely does not contain the data items it requires to complete a particular task (e.g. reconstructing query traffic or performing a specific piece of data analysis)?
2. How can it determine if a data item is not present because it was:
 - * explicitly not recorded or
 - * the data item was not available/present.

For example, capturing C-DNS data from within a nameserver implementation makes it unlikely that the Client Hoplimit can be recorded. Or, if there is no query ARCount recorded and no query OPT RDATA [RFC6891] recorded, is that because no query contained an OPT RR, or because that data was not stored?

The Storage Parameters therefore also contains a Storage Hints item which specifies which items the encoder of the file omits from the stored data and will therefore never be present. (This approach is taken because a flag that indicated which items were included for collection would not guarantee that the item was present, only that it might be.) An implementation decoding that file can then use these to quickly determine whether the input data is rich enough for its needs.

6.2.2. Optional RRs and OPCODEs

Also included in the Storage Parameters are explicit arrays listing the RR types and the OPCODEs to be recorded. These remove any ambiguity over whether messages containing particular OPCODEs or are not present because they did not occur, or because the implementation is not configured to record them.

In the case of OPCODEs, for a message to be fully parsable, the OPCODE must be known to the collecting implementation. Any message with an OPCODE unknown to the collecting implementation cannot be validated as correctly formed, and so must be treated as malformed. Messages with OPCODEs known to the recording application but not listed in the Storage Parameters are discarded by the recording application during C-DNS capture (regardless of whether they are malformed or not).

In the case of RR records, each record in a message must be fully parsable, including parsing the record RDATA, as otherwise the message cannot be validated as correctly formed. Any RR record with an RR type not known to the collecting implementation cannot be validated as correctly formed, and so must be treated as malformed.

Once a message is correctly parsed, an implementation is free to record only a subset of the RR records present.

6.2.3. Storage flags

The Storage Parameters contains flags that can be used to indicate if:

- o the data is anonymized,
- o the data is produced from sample data, or
- o names in the data have been normalized (converted to uniform case).

The Storage Parameters also contains optional fields holding details of the sampling method used and the anonymization method used. It is RECOMMENDED these fields contain URIs [RFC3986] pointing to resources describing the methods used. See Section 15 for further discussion of anonymization and normalization.

6.2.4. IP Address storage

The format can store either full IP addresses or just IP prefixes, the Storage Parameters contains fields to indicate if only IP prefixes were stored.

If the IP address prefixes are absent, then full addresses are stored. In this case the IP version can be directly inferred from the stored address length and the fields "qr-transport-flags" in QueryResponseSignature and "mm-transport-flags" in MalformedMessageData (which contain the IP version bit) are optional.

If IP address prefixes are given, only the prefix bits of addresses are stored. In this case the fields "qr-transport-flags" in QueryResponseSignature and "mm-transport-flags" in MalformedMessageData MUST be present, so that the IP version can be determined. See Section 7.5.3.2 and Section 7.5.3.5.

As an example of storing only IP prefixes, if a client IPv6 prefix of 48 is specified, a client address of 2001:db8:85a3::8a2e:370:7334 will be stored as 0x20010db885a3, reducing address storage space requirements. Similarly, if a client IPv4 prefix of 16 is specified, a client address of 192.0.2.1 will be stored as 0xc000 (192.0).

7. C-DNS format detailed description

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

7.1. Map quantities and indexes

All map keys are integers with values specified in the CDDL. String keys would significantly bloat the file size.

All key values specified are positive integers under 24, so their CBOR representation is a single byte. Positive integer values not currently used as keys in a map are reserved for use in future standard extensions.

Implementations may choose to add additional implementation-specific entries to any map. Negative integer map keys are reserved for these values. Key values from -1 to -24 also have a single byte CBOR representation, so such implementation-specific extensions are not at any space efficiency disadvantage.

An item described as an index is the index of the data item in the referenced array. Indexes are 0-based.

7.2. Tabular representation

The following sections present the C-DNS specification in tabular format with a detailed description of each item.

In all quantities that contain bit flags, bit 0 indicates the least significant bit, i.e. flag "n" in quantity "q" is on if "(q & (1 << n)) != 0".

For the sake of readability, all type and field names defined in the CDDL definition are shown in double quotes. Type names are by convention camel case (e.g. "BlockTable"), field names are lower-case with hyphens (e.g. "block-tables").

For the sake of brevity, the following conventions are used in the tables:

- o The column M marks whether items in a map are mandatory.
 - * X - Mandatory items.
 - * C - Conditionally mandatory item. Such items are usually optional but may be mandatory in some configurations.
 - * If the column is empty, the item is optional.
- o The column T gives the CBOR data type of the item.
 - * U - Unsigned integer
 - * I - Signed integer (i.e. CBOR unsigned or negative integer)
 - * B - Boolean
 - * S - Byte string
 - * T - Text string
 - * M - Map
 - * A - Array

In the case of maps and arrays, more information on the type of each value, include the CDDL definition name if applicable, is given in the description.

7.3. "File"

A C-DNS file has an outer structure "File", a map that contains the following:

Field	M	T	Description
file-type-id	X	T	String "C-DNS" identifying the file type.
file-preamble	X	M	Version and parameter information for the whole file. Map of type "FilePreamble", see Section 7.4.
file-blocks	X	A	Array of items of type "Block", see Section 7.5. The array may be empty if the file contains no data.

7.4. "FilePreamble"

Information about data in the file. A map containing the following:

Field	M	T	Description
major-format-version	X	U	Unsigned integer '1'. The major version of format used in file. See Section 8.
minor-format-version	X	U	Unsigned integer '0'. The minor version of format used in file. See Section 8.
private-version		U	Version indicator available for private use by implementations.
block-parameters	X	A	Array of items of type "BlockParameters", see Section 7.4.1. The array must contain at least one entry. (The "block-parameters-index" item in each "BlockPreamble" indicates which array entry applies to that "Block".)

7.4.1. "BlockParameters"

Parameters relating to data storage and collection which apply to one or more items of type "Block". A map containing the following:

Field	M	T	Description
storage-parameters	X	M	Parameters relating to data storage in a "Block" item. Map of type "StorageParameters", see Section 7.4.1.1.
collection-parameters		M	Parameters relating to collection of the data in a "Block" item. Map of type "CollectionParameters", see Section 7.4.2.

7.4.1.1. "StorageParameters"

Parameters relating to how data is stored in the items of type "Block". A map containing the following:

Field	M	T	Description
ticks-per-second	X	U	Sub-second timing is recorded in ticks. This specifies the number of ticks in a second.
max-block-items	X	U	The maximum number of items stored in any of the arrays in a "Block" item (Q/R items, address event counts or malformed messages). An indication to a decoder of the resources needed to process the file.
storage-hints	X	M	Collection of hints as to which fields are omitted in the arrays that have optional fields. Map of type "StorageHints", see Section 7.4.1.1.1.
opcodes	X	A	Array of OPCODES [opcodes] (unsigned integers, each in the range 0 to 15 inclusive) recorded by the collection implementation. See Section 6.2.2.

Field	M	T	Description
query-response-hints	X	U	<p>Hints indicating which "QueryResponse" fields are candidates for capture or omitted, see section Section 7.6. If a bit is unset, the field is omitted from the capture.</p> <ul style="list-style-type: none"> Bit 0. time-offset Bit 1. client-address-index Bit 2. client-port Bit 3. transaction-id Bit 4. qr-signature-index Bit 5. client-hoplimit Bit 6. response-delay Bit 7. query-name-index Bit 8. query-size Bit 9. response-size Bit 10. response-processing-data Bit 11. query-question-sections Bit 12. query-answer-sections Bit 13. query-authority-sections Bit 14. query-additional-sections Bit 15. response-answer-sections Bit 16. response-authority-sections Bit 17. response-additional-sections
query-response-signature-hints	X	U	<p>Hints indicating which "QueryResponseSignature" fields are candidates for capture or omitted, see section Section 7.5.3.2. If a bit is unset, the field is omitted from the capture.</p> <ul style="list-style-type: none"> Bit 0. server-address Bit 1. server-port Bit 2. qr-transport-flags Bit 3. qr-type Bit 4. qr-sig-flags Bit 5. query-opcode Bit 6. dns-flags Bit 7. query-rcode Bit 8. query-class-type Bit 9. query-qdcount Bit 10. query-ancount Bit 11. query-nscount Bit 12. query-arcount Bit 13. query-edns-version Bit 14. query-udp-size Bit 15. query-opt-rdata

			Bit 16. response-rcode
rr-hints	X	U	Hints indicating which optional "RR" fields are candidates for capture or omitted, see Section 7.5.3.4. If a bit is unset, the field is omitted from the capture.
other-data-hints	X	U	Bit 0. ttl Bit 1. rdata-index Hints indicating which other data types are omitted. If a bit is unset, the the data type is omitted from the capture. Bit 0. malformed-messages Bit 1. address-event-counts

7.4.2. "CollectionParameters"

Parameters providing information to how data in the file was collected (applicable for some, but not all collection environments). The values are informational only and serve as hints to downstream analysers as to the configuration of a collecting implementation. They can provide context when interpreting what data is present/absent from the capture but cannot necessarily be validated against the data captured.

These parameters have no default. If they do not appear, nothing can be inferred about their value.

A map containing the following items:

Field	M	T	Description
query-timeout		U	To be matched with a query, a response must arrive within this number of seconds.
skew-timeout		U	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.
snaplen		U	Collect up to this many bytes per packet.
promisc		B	"true" if promiscuous mode [pcap-options] was enabled on the interface, "false" otherwise.
interfaces		A	Array of identifiers (of type text string) of the interfaces used for collection.
server-addresses		A	Array of server collection IP addresses (of type byte string). Hint for downstream analysers; does not affect collection.
vlan-ids		A	Array of identifiers (of type unsigned integer, each in the range 1 to 4094 inclusive) of VLANs [IEEE802.1Q] selected for collection. VLAN IDs are unique only within an administrative domain.
filter		T	"tcpdump" [pcap-filter] style filter for input.
generator-id		T	Implementation specific human-readable string identifying the collection method.
host-id		T	String identifying the collecting host. Empty if converting an existing packet capture file.

7.5. "Block"

Container for data with common collection and storage parameters. A map containing the following:

Field	M	T	Description
block-preamble	X	M	Overall information for the "Block" item. Map of type "BlockPreamble", see Section 7.5.1.
block-statistics		M	Statistics about the "Block" item. Map of type "BlockStatistics", see Section 7.5.2.
block-tables		M	The arrays containing data referenced by individual "QueryResponse" or "MalformedMessage" items. Map of type "BlockTables", see Section 7.5.3.
query-responses		A	Details of individual DNS Q/R data items. Array of items of type "QueryResponse", see Section 7.6. If present, the array must not be empty.
address-event-counts		A	Per client counts of ICMP messages and TCP resets. Array of items of type "AddressEventCount", see Section 7.7. If present, the array must not be empty.
malformed-messages		A	Details of malformed DNS messages. Array of items of type "MalformedMessage", see Section 7.8. If present, the array must not be empty.

7.5.1. "BlockPreamble"

Overall information for a "Block" item. A map containing the following:

Field	M	T	Description
earliest-time	C	A	A timestamp (2 unsigned integers, "Timestamp") for the earliest record in the "Block" item. The first integer is the number of seconds since the POSIX epoch [posix-time] ("time_t"), excluding leap seconds. The second integer is the number of ticks (see Section 7.4.1.1) since the start of the second. This field is mandatory unless all block items containing a time offset from the start of the block also omit that time offset.
block-parameters -index		U	The index of the item in the "block-parameters" array (in the "file-premable" item) applicable to this block. If not present, index 0 is used. See Section 7.4.1.

7.5.2. "BlockStatistics"

Basic statistical information about a "Block" item. A map containing the following:

Field	M	T	Description
processed-messages		U	Total number of DNS messages processed from the input traffic stream during collection of data in this "Block" item.
qr-data-items		U	Total number of Q/R data items in this "Block" item.
unmatched-queries		U	Number of unmatched queries in this "Block" item.
unmatched-responses		U	Number of unmatched responses in this "Block" item.
discarded-opcode		U	Number of DNS messages processed from the input traffic stream during collection of data in this "Block" item but not recorded because their OPCODE is not in the list to be collected.
malformed-items		U	Number of malformed messages found in input for this "Block" item.

7.5.3. "BlockTables"

Map of arrays containing data referenced by individual "QueryResponse" or "MalformedMessage" items in this "Block". Each element is an array which, if present, must not be empty.

An item in the "qlist" array contains indexes to values in the "qrr" array. Therefore, if "qlist" is present, "qrr" must also be present. Similarly, if "rrlist" is present, "rr" must also be present.

The map contains the following items:

Field	M	T	Description
ip-address		A	Array of IP addresses, in network byte order (of type byte string). If client or server address prefixes are set, only the address prefix bits are stored. Each string is therefore up

			to 4 bytes long for an IPv4 address, or up to 16 bytes long for an IPv6 address. See Section 7.4.1.1.
classtype	A		Array of RR class and type information. Type is "ClassType", see Section 7.5.3.1.
name-rdata	A		Array where each entry is the contents of a single NAME or RDATA in wire format (of type byte string). Note that NAMES, and labels within RDATA contents, are full domain names or labels; no [RFC1035] name compression is used on the individual names/labels within the format.
qr-sig	A		Array Q/R data item signatures. Type is "QueryResponseSignature", see Section 7.5.3.2.
qlist	A		Array of type "QuestionList". A "QuestionList" is an array of unsigned integers, indexes to "Question" items in the "qrr" array.
qrr	A		Array of type "Question". Each entry is the contents of a single question, where a question is the second or subsequent question in a query. See Section 7.5.3.3.
rulist	A		Array of type "RRList". An "RRList" is an array of unsigned integers, indexes to "RR" items in the "rr" array.
rr	A		Array of type "RR". Each entry is the contents of a single RR. See Section 7.5.3.4.
malformed-message-data	A		Array of the contents of malformed messages. Array of type "MalformedMessageData", see Section 7.5.3.5.

7.5.3.1. "ClassType"

RR class and type information. A map containing the following:

Field	M	T	Description
type	X	U	TYPE value [rrtypes].
class	X	U	CLASS value [rrclasses].

7.5.3.2. "QueryResponseSignature"

Elements of a Q/R data item that are often common between multiple individual Q/R data items. A map containing the following:

Field	M	T	Description
server-address-index		U	The index in the item in the "ip-address" array of the server IP address. See Section 7.5.3.
server-port		U	The server port.
qr-transport-flags	C	U	Bit flags describing the transport used to service the query. Same definition as "mm-transport-flags" in Section 7.5.3.5, with an additional indicator for trailing bytes, see Appendix A. Bit 0. IP version. 0 if IPv4, 1 if IPv6. See Section 6.2.4. Bit 1-4. Transport. 4 bit unsigned value where 0 = UDP, 1 = TCP, 2 = TLS, 3 = DTLS [RFC7858], 4 = DoH [RFC8484]. Values 5-15 are reserved for future use. Bit 5. 1 if trailing bytes in query packet. See Section 11.2.
qr-type		U	Type of Query/Response transaction. 0 = Stub. A query from a stub resolver. 1 = Client. An incoming query to a recursive resolver. 2 = Resolver. A query sent from a

			<p>recursive resolver to an authoritative resolver.</p> <p>3 = Authorative. A query to an authoritative resolver.</p> <p>4 = Forwarder. A query sent from a recursive resolver to an upstream recursive resolver.</p> <p>5 = Tool. A query sent to a server by a server tool.</p>
qr-sig-flags	U		<p>Bit flags explicitly indicating attributes of the message pair represented by this Q/R data item (not all attributes may be recorded or deducible).</p> <p>Bit 0. 1 if a Query was present.</p> <p>Bit 1. 1 if a Response was present.</p> <p>Bit 2. 1 if a Query was present and it had an OPT Resource Record.</p> <p>Bit 3. 1 if a Response was present and it had an OPT Resource Record.</p> <p>Bit 4. 1 if a Query was present but had no Question.</p> <p>Bit 5. 1 if a Response was present but had no Question (only one query-name-index is stored per Q/R item).</p>
query-opcode	U		Query OPCODE.
qr-dns-flags	U		<p>Bit flags with values from the Query and Response DNS flags. Flag values are 0 if the Query or Response is not present.</p> <p>Bit 0. Query Checking Disabled (CD).</p> <p>Bit 1. Query Authenticated Data (AD).</p> <p>Bit 2. Query reserved (Z).</p> <p>Bit 3. Query Recursion Available (RA).</p> <p>Bit 4. Query Recursion Desired (RD).</p> <p>Bit 5. Query TrunCation (TC).</p> <p>Bit 6. Query Authoritative Answer (AA).</p> <p>Bit 7. Query DNSSEC answer OK (DO).</p> <p>Bit 8. Response Checking Disabled (CD).</p> <p>Bit 9. Response Authenticated Data (AD).</p>

			Bit 10. Response reserved (Z). Bit 11. Response Recursion Available (RA). Bit 12. Response Recursion Desired (RD). Bit 13. Response TrunCation (TC). Bit 14. Response Authoritative Answer (AA).
query-rcode	U		Query RCODE. If the Query contains OPT [RFC6891], this value incorporates any EXTENDED_RCODE_VALUE [rcodes].
query-classtype-index	U		The index to the item in the the "classtype" array of the CLASS and TYPE of the first Question. See Section 7.5.3.
query-qd-count	U		The QDCOUNT in the Query, or Response if no Query present.
query-an-count	U		Query ANCOUNT.
query-ns-count	U		Query NSCOUNT.
query-ar-count	U		Query ARCOUNT.
edns-version	U		The Query EDNS version.
udp-buf-size	U		The Query EDNS sender's UDP payload size.
opt-rdata-index	U		The index in the "name-rdata" array of the OPT RDATA. See Section 7.5.3.
response-rcode	U		Response RCODE. If the Response contains OPT [RFC6891], this value incorporates any EXTENDED_RCODE_VALUE [rcodes].

7.5.3.3. "Question"

Details on individual Questions in a Question section. A map containing the following:

Field	M	T	Description
name-index	X	U	The index in the "name-rdata" array of the QNAME. See Section 7.5.3.
classtype-index	X	U	The index in the "classtype" array of the CLASS and TYPE of the Question. See Section 7.5.3.

7.5.3.4. "RR"

Details on individual Resource Records in RR sections. A map containing the following:

Field	M	T	Description
name-index	X	U	The index in the "name-rdata" array of the NAME. See Section 7.5.3.
classtype-index	X	U	The index in the "classtype" array of the CLASS and TYPE of the RR. See Section 7.5.3.
ttl		U	The RR Time to Live.
rdata-index		U	The index in the "name-rdata" array of the RR RDATA. See Section 7.5.3.

7.5.3.5. "MalformedMessageData"

Details on malformed message items in this "Block" item. A map containing the following:

Field	M	T	Description
server-address-index		U	The index in the "ip-address" array of the server IP address. See Section 7.5.3.
server-port		U	The server port.
mm-transport-flags	C	U	Bit flags describing the transport used to service the query, see Section 6.2.4. Bit 0. IP version. 0 if IPv4, 1 if IPv6 Bit 1-4. Transport. 4 bit unsigned value where 0 = UDP, 1 = TCP, 2 = TLS, 3 = DTLS [RFC7858], 4 = DoH [RFC8484]. Values 5-15 are reserved for future use.
mm-payload		S	The payload (raw bytes) of the DNS message.

7.6. "QueryResponse"

Details on individual Q/R data items.

Note that there is no requirement that the elements of the "query-responses" array are presented in strict chronological order.

A map containing the following items:

Field	M	T	Description
time-offset		U	Q/R timestamp as an offset in ticks (see Section 7.4.1.1) from "earliest-time". The timestamp is the timestamp of the Query, or the Response if there is no Query.
client-address-index		U	The index in the "ip-address" array of the client IP address. See Section 7.5.3.
client-port		U	The client port.

transaction-id	U	DNS transaction identifier.
qr-signature-index	U	The index in the "qr-sig" array of the "QueryResponseSignature" item. See Section 7.5.3.
client-hoplimit	U	The IPv4 TTL or IPv6 Hoplimit from the Query packet.
response-delay	I	The time difference between Query and Response, in ticks (see Section 7.4.1.1). Only present if there is a query and a response. The delay can be negative if the network stack/capture library returns packets out of order.
query-name-index	U	The index in the "name-rdata" array of the item containing the QNAME for the first Question. See Section 7.5.3.
query-size	U	DNS query message size (see below).
response-size	U	DNS response message size (see below).
response-processing-data	M	Data on response processing. Map of type "ResponseProcessingData", see Section 7.6.1.
query-extended	M	Extended Query data. Map of type "QueryResponseExtended", see Section 7.6.2.
response-extended	M	Extended Response data. Map of type "QueryResponseExtended", see Section 7.6.2.

The "query-size" and "response-size" fields hold the DNS message size. For UDP this is the size of the UDP payload that contained the DNS message. For TCP it is the size of the DNS message as specified in the two-byte message length header. Trailing bytes in UDP queries are routinely observed in traffic to authoritative servers and this value allows a calculation of how many trailing bytes were present.

7.6.1. "ResponseProcessingData"

Information on the server processing that produced the response. A map containing the following:

Field	M	T	Description
bailiwick-index		U	The index in the "name-rdata" array of the owner name for the response bailiwick. See Section 7.5.3.
processing-flags		U	Flags relating to response processing. Bit 0. 1 if the response came from cache.

7.6.2. "QueryResponseExtended"

Extended data on the Q/R data item.

Each item in the map is present only if collection of the relevant details is configured.

A map containing the following items:

Field	M	T	Description
question-index		U	The index in the "qlist" array of the entry listing any second and subsequent Questions in the Question section for the Query or Response. See Section 7.5.3.
answer-index		U	The index in the "rrlist" array of the entry listing the Answer Resource Record sections for the Query or Response. See Section 7.5.3.
authority-index		U	The index in the "rrlist" array of the entry listing the Authority Resource Record sections for the Query or Response. See Section 7.5.3.
additional-index		U	The index in the "rrlist" array of the entry listing the Additional Resource Record sections for the Query or Response. See Section 7.5.3. Note that Query OPT RR data can be optionally stored in the QuerySignature.

7.7. "AddressEventCount"

Counts of various IP related events relating to traffic with individual client addresses. A map containing the following:

Field	M	T	Description
ae-type	X	U	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.
ae-code		U	A code relating to the event. For ICMP or ICMPv6 events, this MUST be the ICMP [RFC0792] or ICMPv6 [RFC4443] code. For other events the contents are undefined.
ae-address-index	X	U	The index in the "ip-address" array of the client address. See Section 7.5.3.
ae-count	X	U	The number of occurrences of this event during the block collection period.

7.8. "MalformedMessage"

Details of malformed messages. A map containing the following:

Field	M	T	Description
time-offset		U	Message timestamp as an offset in ticks (see Section 7.4.1.1) from "earliest-time".
client-address-index		U	The index in the "ip-address" array of the client IP address. See Section 7.5.3.
client-port		U	The client port.
message-data-index		U	The index in the "malformed-message-data" array of the message data for this message. See Section 7.5.3.

8. Versioning

The C-DNS file preamble includes a file format version; a major and minor version number are required fields. The document defines version 1.0 of the C-DNS specification. This section describes the intended use of these version numbers in future specifications.

It is noted that version 1.0 includes many optional fields and therefore consumers of version 1.0 should be inherently robust to parsing files with variable data content.

Within a major version, a new minor version MUST be a strict superset of the previous minor version, with no semantic changes to existing fields. New keys MAY be added to existing maps, and new maps MAY be added. A consumer capable of reading a particular major.minor version MUST also be capable of reading all previous minor versions of the same major version. It SHOULD also be capable of parsing all subsequent minor versions ignoring any keys or maps that it does not recognise.

A new major version indicates changes to the format that are not backwards compatible with previous major versions. A consumer capable of only reading a particular major version (greater than 1) is not required to and has no expectation to be capable of reading a previous major version.

9. C-DNS to PCAP

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

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 will be imperfect.

Even if all sections of the response were captured, one cannot reconstruct the DNS response payload exactly due to the fact that some DNS names in the message on the wire may have been compressed. Section 9.1 discusses this in more detail.

Some transport information is not 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 been split across multiple TCP segments
 - * Multiple DNS messages may have been sent on a single TCP session
- o Malformed DNS messages if the wire format is not recorded
- o Any Non-DNS messages that were in the original packet stream e.g. ICMP

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 malformed messages and Non-DNS packets are captured separately, they can be merged with packet captures reconstructed from C-DNS to produce a more complete packet stream.

9.1. Name compression

All the names stored in the C-DNS format are full domain names; no [RFC1035] name compression is used on the individual names within the format. Therefore when reconstructing a packet, name compression must be used in order to reproduce 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 message. 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.

Therefore, it is not possible to ensure that the DNS response payload is reconstructed byte-for-byte from C-DNS data. 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-DNS data, this is the best that can be achieved.

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

10. Data collection

This section describes a non-normative proposed algorithm for the processing of a captured stream of DNS queries and responses and production of a stream of query/response items, 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, and so on, has already been performed.
2. Each message is associated with transport metadata required to generate the Primary ID (see Section 10.2.1).
3. Each message has a well-formed DNS header of 12 bytes and (if present) the first Question in the Question 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 implementation.

It should be noted that packet capture libraries do not necessarily provide packets in strict chronological order. This can, for example, arise on multi-core platforms where packets arriving at a network device are processed by different cores. On systems where this behaviour has been observed, the timestamps associated with each packet are consistent; queries always have a timestamp prior to the response timestamp. However, the order in which these packets appear in the packet capture stream is not necessarily strictly chronological; a response can appear in the capture stream before the

query that provoked the response. For this discussion, this non-chronological delivery is termed "skew".

In the presence of skew, a response packets can arrive for matching before the corresponding query. To avoid generating false instances of responses without a matching query, and queries without a matching response, the matching algorithm must take account of the possibility of skew.

10.1. Matching algorithm

A schematic representation of the algorithm for matching Q/R data items is shown in Figure 3. It takes individual DNS query or response messages as input, and outputs matched Q/R items. The numbers in the figure identify matching operations listed in Table 1. Specific details of the algorithm, for example queues, timers and identifiers, are given in the following sections.

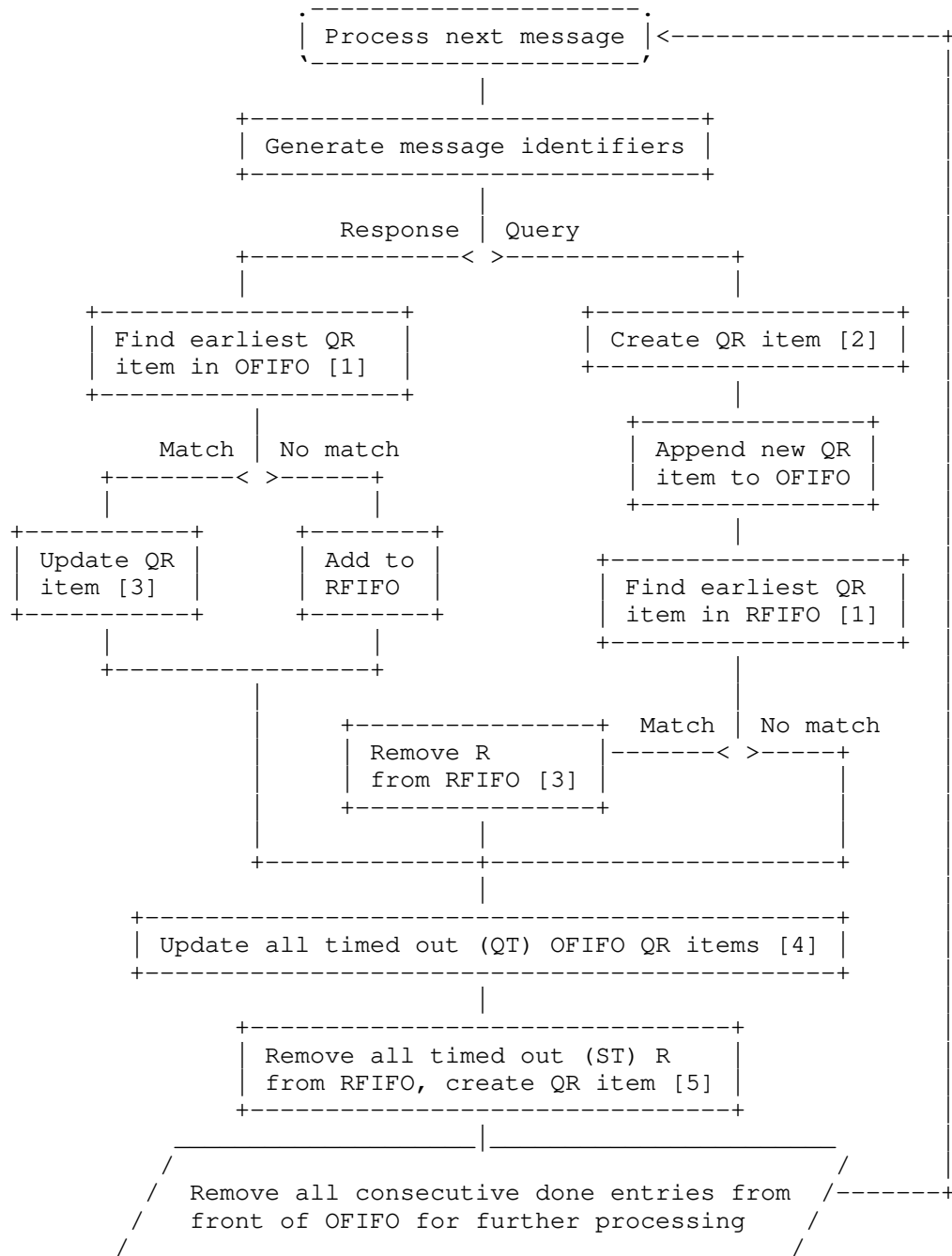


Figure 3: Query/Response matching algorithm

Ref	Operation
[1]	Find earliest QR item in FIFO where: * QR.done = false * QR.Q.PrimaryID == R.PrimaryID and, if both QR.Q and R have SecondaryID: * QR.Q.SecondaryID == R.SecondaryID
[2]	Set: QR.Q := Q QR.R := nil QR.done := false
[3]	Set: QR.R := R QR.done := true
[4]	Set: QR.done := true
[5]	Set: QR.Q := nil QR.R := R QR.done := true

Table 1: Operations used in the matching algorithm

10.2. Message identifiers

10.2.1. Primary ID (required)

A Primary ID is constructed for each message. It 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

10.2.2. 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, responses with RCODE=FORMERR or NOTIMP). In this case the secondary ID is not used in matching.

10.3. Algorithm parameters

1. Query timeout, QT. A query arrives with timestamp t_1 . If no response matching that query has arrived before other input arrives timestamped later than $(t_1 + QT)$, a query/response item containing only a query item is recorded. The query timeout value is typically of the order of 5 seconds.
2. Skew timeout, ST. A response arrives with timestamp t_2 . If a response has not been matched by a query before input arrives timestamped later than $(t_2 + ST)$, a query/response item containing only a response is recorded. The skew timeout value is typically a few microseconds.

10.4. 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.

10.5. Algorithm limitations

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

10.6. Workspace

The algorithm employs two FIFO queues:

- o OFIFO, an output FIFO containing Q/R items in chronological order,
- o RFIFO, a FIFO holding responses without a matching query in order of arrival.

10.7. 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 data items have one of three types of content:

1. A matched pair of query and response messages
2. A query message with no response
3. A response message with 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.

10.8. Post processing

When ending capture, all items in the responses FIFO are timed out immediately, generating response-only entries to the Q/R data item FIFO. These and all other remaining entries in the Q/R data item FIFO should be treated as timed out queries.

11. Implementation guidance

Whilst this document makes no specific recommendations with respect to Canonical CBOR (see Section 3.9 of [RFC7049]) the following guidance may be of use to implementors.

Adherence to the first two rules given in Section 3.9 of [RFC7049] will minimise file sizes.

Adherence to the last two rules given in Section 3.9 of [RFC7049] for all maps and arrays would unacceptably constrain implementations, for example, in the use case of real-time data collection in constrained environments where outputting block tables after query/response data and allowing indefinite length maps and arrays could reduce memory requirements.

11.1. Optional data

When decoding C-DNS data some of the items required for a particular function that the consumer wishes to perform may be missing. Consumers should consider providing configurable default values to be used in place of the missing values in their output.

11.2. Trailing bytes

A DNS query message in a UDP or TCP payload can be followed by some additional (spurious) bytes, which are not stored in C-DNS.

When DNS traffic is sent over TCP, each message is prefixed with a two byte length field which gives the message length, excluding the two byte length field. In this context, trailing bytes can occur in two circumstances with different results:

1. The number of bytes consumed by fully parsing the message is less than the number of bytes given in the length field (i.e. the length field is incorrect and too large). In this case, the surplus bytes are considered trailing bytes in an analogous manner to UDP and recorded as such. If only this case occurs it is possible to process a packet containing multiple DNS messages where one or more has trailing bytes.
2. There are surplus bytes between the end of a well-formed message and the start of the length field for the next message. In this case the first of the surplus bytes will be processed as the first byte of the next length field, and parsing will proceed from there, almost certainly leading to the next and any subsequent messages in the packet being considered malformed. This will not generate a trailing bytes record for the processed well-formed message.

11.3. Limiting collection of RDATA

Implementations should consider providing a configurable maximum RDATA size for capture, for example, to avoid memory issues when confronted with large XFR records.

11.4. Timestamps

The preamble to each block includes a timestamp of the earliest record in the block. As described in Section 7.5.1, the timestamp is an array of 2 unsigned integers. The first is a POSIX "time_t" [posix-time]. Consumers of C-DNS should be aware of this as it excludes leap seconds and therefore may cause minor anomalies in the data e.g. when calculating query throughput.

12. Implementation status

[Note to RFC Editor: please remove this section and reference to [RFC7942] prior to publication.]

This section records the status of known implementations of the protocol defined by this specification at the time of posting of this Internet-Draft, and is based on a proposal described in [RFC7942]. The description of implementations in this section is intended to assist the IETF in its decision processes in progressing drafts to RFCs. Please note that the listing of any individual implementation here does not imply endorsement by the IETF. Furthermore, no effort has been spent to verify the information presented here that was supplied by IETF contributors. This is not intended as, and must not be construed to be, a catalog of available implementations or their features. Readers are advised to note that other implementations may exist.

According to [RFC7942], "this will allow reviewers and working groups to assign due consideration to documents that have the benefit of running code, which may serve as evidence of valuable experimentation and feedback that have made the implemented protocols more mature. It is up to the individual working groups to use this information as they see fit".

12.1. DNS-STATS Compactor

ICANN/Sinodun IT have developed an open source implementation called DNS-STATS Compactor. The Compactor is a suite of tools which can capture DNS traffic (from either a network interface or a PCAP file) and store it in the Compacted-DNS (C-DNS) file format. PCAP files for the captured traffic can also be reconstructed. See Compactor [1].

This implementation:

- o covers the whole of the specification described in the -03 draft with the exception of support for malformed messages and pico second time resolution. (Note: this implementation does allow malformed messages to be recorded separately in a PCAP file).
- o is released under the Mozilla Public License Version 2.0.
- o has a users mailing list available, see dns-stats-users [2].

There is also some discussion of issues encountered during development available at Compressing Pcap Files [3] and Packet Capture [4].

This information was last updated on 3rd of May 2018.

13. IANA considerations

IANA is requested to create a registry "C-DNS DNS Capture Format" containing the subregistries defined in sections Section 13.1 to Section 13.4 inclusive.

In all cases, new entries may be added to the subregistries by Expert Review as defined in [RFC8126]. Experts are expected to exercise their own expert judgement, and should consider the following general guidelines in addition to any guidelines given particular to a subregistry.

- o There should be a real and compelling use for any new value.
- o Values assigned should be carefully chosen to minimise storage requirements for common cases.

13.1. Transport types

IANA is requested to create a registry "C-DNS Transports" of C-DNS transport type identifiers. The primary purpose of this registry is to provide unique identifiers for all transports used for DNS queries.

The following note is included in this registry: "In version 1.0 of C-DNS [[this RFC]], there is a field to identify the type of DNS transport. This field is 4 bits in size."

The initial contents of the registry are as follows - see sections Section 7.5.3.2 and Section 7.5.3.5 of [[this RFC]]:

Identifier	Name	Reference
0	UDP	[[this RFC]]
1	TCP	[[this RFC]]
2	TLS	[[this RFC]]
3	DTLS	[[this RFC]]
4	DoH	[[this RFC]]
5-15	Unassigned	

Expert reviewers should take the following points into consideration:

- o Is the requested DNS transport described by a Standards Track RFC?

13.2. Data storage flags

IANA is requested to create a registry "C-DNS Storage Flags" of C-DNS data storage flags. The primary purpose of this registry is to provide indicators giving hints on processing of the data stored.

The following note is included in this registry: "In version 1.0 of C-DNS [[this RFC]], there is a field describing attributes of the data recorded. The field is a CBOR [RFC7049] unsigned integer holding bit flags."

The initial contents of the registry are as follows - see section Section 7.4.1.1 of [[this RFC]]:

Bit	Name	Description	Reference
0	anonymised-data	The data has been anonymised.	[[this RFC]]
1	sampled-data	The data is sampled data.	[[this RFC]]
2	normalized-names	Names in the data have been normalized.	[[this RFC]]
3-63	Unassigned		

13.3. Response processing flags

IANA is requested to create a registry "C-DNS Response Flags" of C-DNS response processing flags. The primary purpose of this registry is to provide indicators giving hints on the generation of a particular response.

The following note is included in this registry: "In version 1.0 of C-DNS [[this RFC]], there is a field describing attributes of the responses recorded. The field is a CBOR [RFC7049] unsigned integer holding bit flags."

The initial contents of the registry are as follows - see section Section 7.6.1 of [[this RFC]]:

Bit	Name	Description	Reference
0	from-cache	The response came from cache.	[[this RFC]]
1-63	Unassigned		

13.4. AddressEvent types

IANA is requested to create a registry "C-DNS Address Event Types" of C-DNS AddressEvent types. The primary purpose of this registry is to provide unique identifiers of different types of C-DNS address events, and so specify the contents of the optional companion field "ae-code" for each type.

The following note is included in this registry: "In version 1.0 of C-DNS [[this RFC]], there is a field identify types of the events related to client addresses. This field is a CBOR [RFC7049] unsigned integer. There is a related optional field "ae-code", which, if present, holds an additional CBOR unsigned integer giving additional information specific to the event type."

The initial contents of the registry are as follows - see section Section 7.7:

Identifier	Event Type	ae-code contents	Reference
0	TCP reset	None	[[this RFC]]
1	ICMP time exceeded	ICMP code [icmpcodes]	[[this RFC]]
2	ICMP destination unreachable	ICMP code [icmpcodes]	[[this RFC]]
3	ICMPv6 time exceeded	ICMPv6 code [icmp6codes]	[[this RFC]]
4	ICMPv6 destination unreachable	ICMPv6 code [icmp6codes]	[[this RFC]]
5	ICMPv6 packet too big	ICMPv6 code [icmp6codes]	[[this RFC]]
>5	Unassigned		

Expert reviewers should take the following points into consideration:

- o "ae-code" contents must be defined for a type, or if not appropriate specified as "None". A specification of "None" requires less storage, and is therefore preferred.

14. Security considerations

Any control interface MUST perform authentication and encryption.

Any data upload MUST be authenticated and encrypted.

15. Privacy considerations

Storage of DNS traffic by operators in PCAP and other formats is a long standing and widespread practice. Section 2.5 of [I-D.bortzmeyer-dprive-rfc7626-bis] is an analysis of the risks to Internet users of the storage of DNS traffic data in servers (recursive resolvers, authoritative and rogue servers).

Section 5.2 of [I-D.dickinson-dprive-bcp-op] describes mitigations for those risks for data stored on recursive resolvers (but which could by extension apply to authoritative servers). These include data handling practices and methods for data minimization, IP address pseudonymization and anonymization. Appendix B of that document presents an analysis of 7 published anonymization processes. In addition, RSSAC have recently published RSSAC04: [5] "Recommendations on Anonymization Processes for Source IP Addresses Submitted for Future Analysis".

The above analyses consider full data capture (e.g using PCAP) as a baseline for privacy considerations and therefore this format specification introduces no new user privacy issues beyond those of full data capture (which are quite severe). It does provide mechanisms to selectively record only certain fields at the time of data capture to improve user privacy and to explicitly indicate that data is sampled and or anonymized. It also provide flags to indicate if data normalization has been performed; data normalization increases user privacy by reducing the potential for fingerprinting individuals, however, a trade-off is potentially reducing the capacity to identify attack traffic via query name signatures. Operators should carefully consider their operational requirements and privacy policies and SHOULD capture at source the minimum user data required to meet their needs.

16. 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 and Paul Hoffman for a detailed review of the document and the C-DNS CDDL.

Thanks also to Robert Edmonds, Jerry Lundstroem, Richard Gibson, Stephane Bortzmeyer and many other members of DNSOP for review.

Also, Miek Gieben for mmark [6]

17. Changelog

draft-ietf-dnsop-dns-capture-format-10

- o Add IANA Considerations

- o Convert graph in C.6 to table

draft-ietf-dnsop-dns-capture-format-09

- o Editorial changes arising from IESG review

- o `*-transport-flags` and may be mandatory in some configurations

- o Mark fields that are conditionally mandatory

- o Change 'promisc' flag CDDL data type to boolean

- o Add ranges to configuration quantities where appropriate

draft-ietf-dnsop-dns-capture-format-08

- o Convert diagrams to ASCII

- o Describe versioning

- o Fix unused group warning in CDDL

draft-ietf-dnsop-dns-capture-format-07

- o Resolve outstanding questions and TODOs

- o Make RR RDATA optional

- o Update matching diagram and explain skew

- o Add count of discarded messages to block statistics

- o Editorial clarifications and improvements

draft-ietf-dnsop-dns-capture-format-06

- o Correct BlockParameters type to map

- o Make RR ttl optional

- o Add storage flag indicating name normalization

- o Add storage parameter fields for sampling and anonymization methods
 - o Editorial clarifications and improvements
- draft-ietf-dnsop-dns-capture-format-05
- o Make all data items in Q/R, QuerySignature and Malformed Message arrays optional
 - o Re-structure the FilePreamble and ConfigurationParameters into BlockParameters
 - o BlockParameters has separate Storage and Collection Parameters
 - o Storage Parameters includes information on what optional fields are present, and flags specifying anonymization or sampling
 - o Addresses can now be stored as prefixes.
 - o Switch to using a variable sub-second timing granularity
 - o Add response bailiwick and query response type
 - o Add specifics of how to record malformed messages
 - o Add implementation guidance
 - o Improve terminology and naming consistency
- draft-ietf-dnsop-dns-capture-format-04
- o Correct query-d0 to query-do in CDDL
 - o Clarify that map keys are unsigned integers
 - o Add Type to Class/Type table
 - o Clarify storage format in section 7.12
- draft-ietf-dnsop-dns-capture-format-03
- o Added an Implementation Status section
- draft-ietf-dnsop-dns-capture-format-02
- o Update qr_data_format.png to match CDDL

- o Editorial clarifications and improvements
- draft-ietf-dnsop-dns-capture-format-01
- o Many editorial improvements by Paul Hoffman
 - o Included discussion of malformed message handling
 - o Improved Appendix C on Comparison of Binary Formats
 - o Now using C-DNS field names in the tables in section 8
 - o A handful of new fields included (CDDL updated)
 - o Timestamps now include optional picoseconds
 - o Added details of block statistics
- draft-ietf-dnsop-dns-capture-format-00
- o Changed dnstap.io to dnstap.info
 - o qr_data_format.png was cut off at the bottom
 - o Update authors address
 - o Improve wording in Abstract
 - o Changed DNS-STAT to C-DNS in CDDL
 - o Set the format version in the CDDL
 - o Added a TODO: Add block statistics
 - o Added a TODO: Add extend to support pico/nano. Also do this for Time offset and Response delay
 - o Added a TODO: Need to develop optional representation of malformed messages within C-DNS and what this means for packet matching. This may influence which fields are optional in the rest of the representation.
 - o Added section on design goals to Introduction
 - o Added a TODO: Can Class be optimised? Should a class of IN be inferred if not present?
- draft-dickinson-dnsop-dns-capture-format-00

- o Initial commit

18. References

18.1. Normative References

[I-D.ietf-cbor-cddl]

Birkholz, H., Vigano, C., and C. Bormann, "Concise data definition language (CDDL): a notational convention to express CBOR and JSON data structures", draft-ietf-cbor-cddl-06 (work in progress), November 2018.

[pcap-filter]

tcpdump.org, "Manpage of PCAP-FILTER", 2017,
<<http://www.tcpdump.org/manpages/pcap-filter.7.html>>.

[pcap-options]

tcpdump.org, "Manpage of PCAP", 2018,
<<http://www.tcpdump.org/manpages/pcap.3pcap.html>>.

[posix-time]

The Open Group, "Section 4.16, Base Definitions, Standard for Information Technology - Portable Operating System Interface (POSIX(R)) Base Specifications, Issue 7", IEEE Standard 1003.1 2017 Edition,
DOI 10.1109/IEEESTD.2018.8277153, 2017.

[RFC0792] Postel, J., "Internet Control Message Protocol", STD 5, RFC 792, DOI 10.17487/RFC0792, September 1981,
<<https://www.rfc-editor.org/info/rfc792>>.

[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>>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005,
<<https://www.rfc-editor.org/info/rfc3986>>.

- [RFC4443] Conta, A., Deering, S., and M. Gupta, Ed., "Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification", STD 89, RFC 4443, DOI 10.17487/RFC4443, March 2006, <<https://www.rfc-editor.org/info/rfc4443>>.
- [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>>.
- [RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <<https://www.rfc-editor.org/info/rfc7049>>.
- [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>>.
- [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>>.
- [RFC8484] Hoffman, P. and P. McManus, "DNS Queries over HTTPS (DoH)", RFC 8484, DOI 10.17487/RFC8484, October 2018, <<https://www.rfc-editor.org/info/rfc8484>>.

18.2. 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.info, "dnstap", 2016, <<http://dnstap.info/>>.
- [dsc] Wessels, D. and J. Lundstrom, "DSC", 2016, <<https://www.dns-oarc.net/tools/dsc>>.

- [I-D.bortzmeyer-dprive-rfc7626-bis]
Bortzmeyer, S. and S. Dickinson, "DNS Privacy Considerations", draft-bortzmeyer-dprive-rfc7626-bis-01 (work in progress), July 2018.
- [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.dickinson-dprive-bcp-op]
Dickinson, S., Overeinder, B., Rijswijk-Deij, R., and A. Mankin, "Recommendations for DNS Privacy Service Operators", draft-dickinson-dprive-bcp-op-01 (work in progress), July 2018.
- [icmp6codes]
IANA, "ICMPv6 "Code" Fields", 2018,
<<https://www.iana.org/assignments/icmpv6-parameters/icmpv6-parameters.xhtml#icmpv6-parameters-3>>.
- [icmpcodes]
IANA, "Code Fields", 2018,
<<https://www.iana.org/assignments/icmp-parameters/icmp-parameters.xhtml#icmp-parameters-codes>>.
- [IEEE802.1Q]
IEEE, "IEEE Standard for Local and metropolitan area networks -- Bridges and Bridged Networks", DOI 10.1109/IEEESTD.2014.6991462, 2014.
- [opcodes] IANA, "DNS OpCodes", 2018,
<<http://www.iana.org/assignments/dns-parameters/dns-parameters.xhtml#dns-parameters-5>>.
- [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>>.
- [rcodes] IANA, "DNS RCODEs", 2018,
<<http://www.iana.org/assignments/dns-parameters/dns-parameters.xhtml#dns-parameters-6>>.

- [RFC7942] Sheffer, Y. and A. Farrel, "Improving Awareness of Running Code: The Implementation Status Section", BCP 205, RFC 7942, DOI 10.17487/RFC7942, July 2016, <<https://www.rfc-editor.org/info/rfc7942>>.
- [RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <<https://www.rfc-editor.org/info/rfc8259>>.
- [RFC8427] Hoffman, P., "Representing DNS Messages in JSON", RFC 8427, DOI 10.17487/RFC8427, July 2018, <<https://www.rfc-editor.org/info/rfc8427>>.
- [rrclasses] IANA, "DNS CLASSEs", 2018, <<http://www.iana.org/assignments/dns-parameters/dns-parameters.xhtml#dns-parameters-2>>.
- [rrtypes] IANA, "Resource Record (RR) TYPEs", 2018, <<http://www.iana.org/assignments/dns-parameters/dns-parameters.xhtml#dns-parameters-4>>.

18.3. URIs

- [1] <https://github.com/dns-stats/compactor/wiki>
- [2] <https://mm.dns-stats.org/mailman/listinfo/dns-stats-users>
- [3] <https://www.sinodun.com/2017/06/compressing-pcap-files/>
- [4] <https://www.sinodun.com/2017/06/more-on-debian-jessiebuntu-trusty-packet-capture-woes/>
- [5] <https://www.icann.org/en/system/files/files/rssac-040-07aug18-en.pdf>
- [6] <https://github.com/miekg/mmark>
- [7] <https://www.nlnetlabs.nl/projects/nsd/>
- [8] <https://www.knot-dns.cz/>
- [9] <https://avro.apache.org/>
- [10] <https://developers.google.com/protocol-buffers/>
- [11] <http://cbor.io>

- [12] <https://github.com/kubo/snzip>
- [13] <http://google.github.io/snappy/>
- [14] <http://lz4.github.io/lz4/>
- [15] <http://www.gzip.org/>
- [16] <http://facebook.github.io/zstd/>
- [17] <http://tukaani.org/xz/>

Appendix A. CDDL

This appendix gives a CDDL [I-D.ietf-cbor-cddl] specification for C-DNS.

CDDL does not permit a range of allowed values to be specified for a bitfield. Where necessary, those values are given as a CDDL group, but the group definition is commented out to prevent CDDL tooling from warning that the group is unused.

```
; CDDL specification of the file format for C-DNS,  
; which describes a collection of DNS messages and  
; traffic meta-data.
```

```
;  
; The overall structure of a file.  
;  
File = [  
    file-type-id : "C-DNS",  
    file-preamble : FilePreamble,  
    file-blocks : [* Block],  
]  
  
;  
; The file preamble.  
;  
FilePreamble = {  
    major-format-version => 1,  
    minor-format-version => 0,  
    ? private-version => uint,  
    block-parameters => [+ BlockParameters],  
}  
major-format-version = 0  
minor-format-version = 1  
private-version = 2  
block-parameters = 3
```

```

BlockParameters = {
    storage-parameters      => StorageParameters,
    ? collection-parameters => CollectionParameters,
}
storage-parameters      = 0
collection-parameters   = 1

IPv6PrefixLength = 1..128
IPv4PrefixLength = 1..32
OpcodeRange = 0..15
RRTypeRange = 0..65535

StorageParameters = {
    ticks-per-second          => uint,
    max-block-items          => uint,
    storage-hints             => StorageHints,
    opcodes                  => [+ OpcodeRange],
    rr-types                 => [+ RRTypeRange],
    ? storage-flags          => StorageFlags,
    ? client-address-prefix-ipv4 => IPv4PrefixLength,
    ? client-address-prefix-ipv6 => IPv6PrefixLength,
    ? server-address-prefix-ipv4 => IPv4PrefixLength,
    ? server-address-prefix-ipv6 => IPv6PrefixLength,
    ? sampling-method        => tstr,
    ? anonymisation-method   => tstr,
}
ticks-per-second          = 0
max-block-items          = 1
storage-hints             = 2
opcodes                  = 3
rr-types                 = 4
storage-flags            = 5
client-address-prefix-ipv4 = 6
client-address-prefix-ipv6 = 7
server-address-prefix-ipv4 = 8
server-address-prefix-ipv6 = 9
sampling-method          = 10
anonymisation-method     = 11

; A hint indicates if the collection method will output the
; item or will ignore the item if present.
StorageHints = {
    query-response-hints          => QueryResponseHints,
    query-response-signature-hints =>
        QueryResponseSignatureHints,
    rr-hints                      => RRHints,
    other-data-hints              => OtherDataHints,
}

```

```
query-response-hints          = 0
query-response-signature-hints = 1
rr-hints                      = 2
other-data-hints              = 3

QueryResponseHintValues = &(amp;
    time-offset                : 0,
    client-address-index       : 1,
    client-port                 : 2,
    transaction-id             : 3,
    qr-signature-index         : 4,
    client-hoplimit            : 5,
    response-delay              : 6,
    query-name-index           : 7,
    query-size                  : 8,
    response-size               : 9,
    response-processing-data    : 10,
    query-question-sections     : 11,      ; Second & subsequent
                                         ; questions
    query-answer-sections      : 12,
    query-authority-sections    : 13,
    query-additional-sections  : 14,
    response-answer-sections    : 15,
    response-authority-sections : 16,
    response-additional-sections : 17,
)
QueryResponseHints = uint .bits QueryResponseHintValues

QueryResponseSignatureHintValues = &(amp;
    server-address             : 0,
    server-port                : 1,
    qr-transport-flags        : 2,
    qr-type                    : 3,
    qr-sig-flags               : 4,
    query-opcode               : 5,
    dns-flags                   : 6,
    query-rcode                 : 7,
    query-class-type           : 8,
    query-qdcount               : 9,
    query-ancount               : 10,
    query-arcount               : 11,
    query-nscount               : 12,
    query-edns-version         : 13,
    query-udp-size              : 14,
    query-opt-rdata             : 15,
    response-rcode              : 16,
)
QueryResponseSignatureHints =
```

```
uint .bits QueryResponseSignatureHintValues

RRHintValues = &(amp;
    ttl      : 0,
    rdata-index : 1,
)
RRHints = uint .bits RRHintValues

OtherDataHintValues = &(amp;
    malformed-messages : 0,
    address-event-counts : 1,
)
OtherDataHints = uint .bits OtherDataHintValues

StorageFlagValues = &(amp;
    anonymised-data : 0,
    sampled-data : 1,
    normalized-names : 2,
)
StorageFlags = uint .bits StorageFlagValues

; Hints for later analysis.
VLANIdRange = 1..4094

CollectionParameters = {
    ? query-timeout      => uint,
    ? skew-timeout      => uint,
    ? snaplen           => uint,
    ? promisc           => bool,
    ? interfaces        => [+ tstr],
    ? server-addresses  => [+ IPAddress],
    ? vlan-ids          => [+ VLANIdRange],
    ? filter            => tstr,
    ? generator-id      => tstr,
    ? host-id           => tstr,
}
query-timeout      = 0
skew-timeout      = 1
snaplen           = 2
promisc           = 3
interfaces        = 4
server-addresses  = 5
vlan-ids          = 6
filter            = 7
generator-id      = 8
host-id           = 9

;
```

```

; Data in the file is stored in Blocks.
;
Block = {
    block-preamble          => BlockPreamble,
    ? block-statistics      => BlockStatistics, ; Much of this
                                                ; could be derived
    ? block-tables         => BlockTables,
    ? query-responses       => [+ QueryResponse],
    ? address-event-counts => [+ AddressEventCount],
    ? malformed-messages   => [+ MalformedMessage],
}
block-preamble          = 0
block-statistics        = 1
block-tables            = 2
query-responses         = 3
address-event-counts    = 4
malformed-messages     = 5

;
; The (mandatory) preamble to a block.
;
BlockPreamble = {
    ? earliest-time          => Timestamp,
    ? block-parameters-index => uint .default 0,
}
earliest-time            = 0
block-parameters-index  = 1

; Ticks are subsecond intervals. The number of ticks in a second is
; file/block metadata. Signed and unsigned tick types are defined.
ticks = int
uticks = uint

Timestamp = [
    timestamp-secs : uint,
    timestamp-uticks : uticks,
]

;
; Statistics about the block contents.
;
BlockStatistics = {
    ? processed-messages => uint,
    ? qr-data-items     => uint,
    ? unmatched-queries => uint,
    ? unmatched-responses => uint,
    ? discarded-opcode  => uint,
    ? malformed-items   => uint,
}

```

```

}
processed-messages = 0
qr-data-items      = 1
unmatched-queries  = 2
unmatched-responses = 3
discarded-opcode   = 4
malformed-items    = 5

;
; Tables of common data referenced from records in a block.
;
BlockTables = {
    ? ip-address           => [+ IPAddress],
    ? classtype           => [+ ClassType],
    ? name-rdata          => [+ bstr],      ; Holds both Names
                                   ; and RDATA
    ? qr-sig              => [+ QueryResponseSignature],
    ? QuestionTables,
    ? RRTables,
    ? malformed-message-data => [+ MalformedMessageData],
}
ip-address          = 0
classtype           = 1
name-rdata          = 2
qr-sig              = 3
qlist               = 4
qrr                 = 5
rrlist              = 6
rr                  = 7
malformed-message-data = 8

IPv4Address = bstr .size 4
IPv6Address = bstr .size 16
IPAddress = IPv4Address / IPv6Address

ClassType = {
    type => uint,
    class => uint,
}
type = 0
class = 1

QueryResponseSignature = {
    ? server-address-index => uint,
    ? server-port          => uint,
    ? qr-transport-flags  => QueryResponseTransportFlags,
    ? qr-type              => QueryResponseType,
    ? qr-sig-flags         => QueryResponseFlags,
}

```

```

    ? query-opcode           => uint,
    ? qr-dns-flags           => DNSFlags,
    ? query-rcode            => uint,
    ? query-classtype-index  => uint,
    ? query-qd-count         => uint,
    ? query-an-count         => uint,
    ? query-ns-count         => uint,
    ? query-ar-count         => uint,
    ? edns-version           => uint,
    ? udp-buf-size           => uint,
    ? opt-rdata-index        => uint,
    ? response-rcode         => uint,
}
server-address-index = 0
server-port          = 1
qr-transport-flags  = 2
qr-type              = 3
qr-sig-flags        = 4
query-opcode         = 5
qr-dns-flags        = 6
query-rcode          = 7
query-classtype-index = 8
query-qd-count       = 9
query-an-count       = 10
query-ns-count       = 12
query-ar-count       = 12
edns-version         = 13
udp-buf-size         = 14
opt-rdata-index      = 15
response-rcode       = 16

```

```

; Transport gives the values that may appear in bits 1..4 of
; TransportFlags. There is currently no way to express this in
; CDDL, so Transport is unused. To avoid confusion when used
; with CDDL tools, it is commented out.
;

```

```

; Transport = &(amp;
;     udp           : 0,
;     tcp           : 1,
;     tls           : 2,
;     dtls          : 3,
;     doh           : 4,
; )

```

```

TransportFlagValues = &(amp;
    ip-version      : 0,      ; 0=IPv4, 1=IPv6
) / (1..4)
TransportFlags = uint .bits TransportFlagValues

```

```
QueryResponseTransportFlagValues = &(amp;
    query-trailingdata : 5,
) / TransportFlagValues
QueryResponseTransportFlags =
    uint .bits QueryResponseTransportFlagValues

QueryResponseType = &(amp;
    stub      : 0,
    client    : 1,
    resolver  : 2,
    auth      : 3,
    forwarder : 4,
    tool      : 5,
)

QueryResponseFlagValues = &(amp;
    has-query      : 0,
    has-reponse    : 1,
    query-has-opt  : 2,
    response-has-opt : 3,
    query-has-no-question : 4,
    response-has-no-question: 5,
)
QueryResponseFlags = uint .bits QueryResponseFlagValues

DNSFlagValues = &(amp;
    query-cd      : 0,
    query-ad      : 1,
    query-z       : 2,
    query-ra      : 3,
    query-rd      : 4,
    query-tc      : 5,
    query-aa      : 6,
    query-do      : 7,
    response-cd   : 8,
    response-ad   : 9,
    response-z    : 10,
    response-ra   : 11,
    response-rd   : 12,
    response-tc   : 13,
    response-aa   : 14,
)
DNSFlags = uint .bits DNSFlagValues

QuestionTables = (
    qlist => [+ QuestionList],
    qrr   => [+ Question]
)
```



```

    QuestionList = [+ uint]                ; Index of Question

    Question = {                          ; Second and subsequent questions
        name-index      => uint,          ; Index to a name in the
                                        ; name-rdata table
        classtype-index => uint,
    }
    name-index      = 0
    classtype-index = 1

    RRTables = (
        rrlist => [+ RRList],
        rr    => [+ RR]
    )

    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
    }
    ; Other map key values already defined above.
    ttl           = 2
    rdata-index  = 3

    MalformedMessageData = {
        ? server-address-index => uint,
        ? server-port         => uint,
        ? mm-transport-flags  => TransportFlags,
        ? mm-payload          => bstr,
    }
    ; Other map key values already defined above.
    mm-transport-flags  = 2
    mm-payload          = 3

    ;
    ; A single query/response pair.
    ;
    QueryResponse = {
        ? time-offset      => uticks,    ; Time offset from
                                        ; start of block
        ? client-address-index => uint,
        ? client-port      => uint,
        ? transaction-id   => uint,
    }

```

```

    ? qr-signature-index      => uint,
    ? client-hoplimit         => uint,
    ? response-delay          => ticks,
    ? query-name-index        => uint,
    ? query-size              => uint,          ; DNS size of query
    ? response-size           => uint,          ; DNS size of response
    ? response-processing-data => ResponseProcessingData,
    ? query-extended          => QueryResponseExtended,
    ? response-extended       => QueryResponseExtended,
}
time-offset                  = 0
client-address-index         = 1
client-port                  = 2
transaction-id               = 3
qr-signature-index          = 4
client-hoplimit              = 5
response-delay               = 6
query-name-index             = 7
query-size                   = 8
response-size                 = 9
response-processing-data     = 10
query-extended               = 11
response-extended            = 12

ResponseProcessingData = {
    ? bailiwick-index => uint,
    ? processing-flags => ResponseProcessingFlags,
}
bailiwick-index = 0
processing-flags = 1

ResponseProcessingFlagValues = &(amp;
    from-cache : 0,
)
ResponseProcessingFlags = uint .bits ResponseProcessingFlagValues

QueryResponseExtended = {
    ? question-index => uint,          ; Index of QuestionList
    ? answer-index   => uint,          ; Index of RRLList
    ? authority-index => uint,
    ? additional-index => uint,
}
question-index   = 0
answer-index     = 1
authority-index  = 2
additional-index = 3

;

```

```
; Address event data.
;
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,
)

;
; Malformed messages.
;
MalformedMessage = {
    ? time-offset      => uticks,      ; Time offset from
                                       ; start of block

    ? client-address-index => uint,
    ? client-port         => uint,
    ? message-data-index  => uint,
}
; Other map key values already defined above.
message-data-index = 3
```

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 foo.example.

N	Name	Uncompressed	Compression Target
1	foo.example		

The next name added is bar.example. This is matched against foo.example. The example 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	foo.example		
2	bar.example	bar	1 + offset to example

The third name added is www.bar.example. This is first matched against foo.example, 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	foo.example		
2	bar.example	bar	1 + offset to example
3	www.bar.example	www	2

As an optimization, if a name is already perfectly compressed (in other words, the uncompressed part of the name is empty), then 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 [7] 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 [8] 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:
 - * 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.
 - * The compression target name MUST be in RDATA.
 - * The name section TYPE must match the compression target name section TYPE.
 - * 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 serialisation formats were considered, and for completeness were also compared to JSON.

- o Apache Avro [9]. Data is stored according to a pre-defined schema. The schema itself is always included in the data file.

Data can therefore be stored untagged, for a smaller serialisation size, and be written and read by an Avro library.

- * At the time of writing, Avro libraries are available for C, C++, C#, Java, Python, Ruby and PHP. Optionally tools are available for C++, Java and C# to generate code for encoding and decoding.
- o Google Protocol Buffers [10]. Data is stored according to a pre-defined schema. The schema is used by a generator to generate code for encoding and decoding the data. Data can therefore be stored untagged, for a smaller serialisation size. The schema is not stored with the data, so unlike Avro cannot be read with a generic library.
 - * Code must be generated for a particular data schema to read and write data using that schema. At the time of writing, the Google code generator can currently generate code for encoding and decoding a schema for C++, Go, Java, Python, Ruby, C#, Objective-C, Javascript and PHP.
- o CBOR [11]. Defined in [RFC7049], this serialisation format is comparable to JSON but with a binary representation. It does not use a pre-defined schema, so data is always stored tagged. However, CBOR data schemas can be described using CDDL [I-D.ietf-cbor-cddl] and tools exist to verify data files conform to the schema.
 - * CBOR is a simple format, and simple to implement. At the time of writing, the CBOR website lists implementations for 16 languages.

Avro and Protocol Buffers both allow storage of untagged data, but because they rely on the data schema for this, their implementation is considerably more complex than CBOR. Using Avro or Protocol Buffers in an unsupported environment would require notably greater development effort compared to CBOR.

A test program was written which reads input from a PCAP file and writes output using one of two basic structures; either a simple structure, where each query/response pair is represented in a single record entry, or the C-DNS block structure.

The resulting output files were then compressed using a variety of common general-purpose lossless compression tools to explore the compressibility of the formats. The compression tools employed were:

- o snzip [12]. A command line compression tool based on the Google Snappy [13] library.
- o lz4 [14]. The command line compression tool from the reference C LZ4 implementation.
- o gzip [15]. The ubiquitous GNU zip tool.
- o zstd [16]. Compression using the Zstandard algorithm.
- o xz [17]. A popular compression tool noted for high compression.

In all cases the compression tools were run using their default settings.

Note that this draft does not mandate the use of compression, nor any particular compression scheme, but it anticipates that in practice output data will be subject to general-purpose compression, and so this should be taken into consideration.

"test.pcap", a 662Mb capture of sample data from a root instance was used for the comparison. The following table shows the formatted size and size after compression (abbreviated to Comp. in the table headers), together with the task resident set size (RSS) and the user time taken by the compression. File sizes are in Mb, RSS in kb and user time in seconds.

Format	File size	Comp.	Comp. size	RSS	User time
PCAP	661.87	snzip	212.48	2696	1.26
		lz4	181.58	6336	1.35
		gzip	153.46	1428	18.20
		zstd	87.07	3544	4.27
		xz	49.09	97416	160.79
JSON simple	4113.92	snzip	603.78	2656	5.72
		lz4	386.42	5636	5.25
		gzip	271.11	1492	73.00
		zstd	133.43	3284	8.68
		xz	51.98	97412	600.74
Avro simple	640.45	snzip	148.98	2656	0.90
		lz4	111.92	5828	0.99
		gzip	103.07	1540	11.52
		zstd	49.08	3524	2.50
		xz	22.87	97308	90.34

CBOR simple	764.82	snzip	164.57	2664	1.11
		lz4	120.98	5892	1.13
		gzip	110.61	1428	12.88
		zstd	54.14	3224	2.77
		xz	23.43	97276	111.48
PBuf simple	749.51	snzip	167.16	2660	1.08
		lz4	123.09	5824	1.14
		gzip	112.05	1424	12.75
		zstd	53.39	3388	2.76
		xz	23.99	97348	106.47
JSON block	519.77	snzip	106.12	2812	0.93
		lz4	104.34	6080	0.97
		gzip	57.97	1604	12.70
		zstd	61.51	3396	3.45
		xz	27.67	97524	169.10
Avro block	60.45	snzip	48.38	2688	0.20
		lz4	48.78	8540	0.22
		gzip	39.62	1576	2.92
		zstd	29.63	3612	1.25
		xz	18.28	97564	25.81
CBOR block	75.25	snzip	53.27	2684	0.24
		lz4	51.88	8008	0.28
		gzip	41.17	1548	4.36
		zstd	30.61	3476	1.48
		xz	18.15	97556	38.78
PBuf block	67.98	snzip	51.10	2636	0.24
		lz4	52.39	8304	0.24
		gzip	40.19	1520	3.63
		zstd	31.61	3576	1.40
		xz	17.94	97440	33.99

The above results are discussed in the following sections.

C.1. Comparison with full PCAP files

An important first consideration is whether moving away from PCAP offers significant benefits.

The simple binary formats are typically larger than PCAP, even though they omit some information such as Ethernet MAC addresses. But not only do they require less CPU to compress than PCAP, the resulting compressed files are smaller than compressed PCAP.

C.2. Simple versus block coding

The intention of the block coding is to perform data de-duplication on query/response records within the block. The simple and block formats above store exactly the same information for each query/response record. This information is parsed from the DNS traffic in the input PCAP file, and in all cases each field has an identifier and the field data is typed.

The data de-duplication on the block formats show an order of magnitude reduction in the size of the format file size against the simple formats. As would be expected, the compression tools are able to find and exploit a lot of this duplication, but as the de-duplication process uses knowledge of DNS traffic, it is able to retain a size advantage. This advantage reduces as stronger compression is applied, as again would be expected, but even with the strongest compression applied the block formatted data remains around 75% of the size of the simple format and its compression requires roughly a third of the CPU time.

C.3. Binary versus text formats

Text data formats offer many advantages over binary formats, particularly in the areas of ad-hoc data inspection and extraction. It was therefore felt worthwhile to carry out a direct comparison, implementing JSON versions of the simple and block formats.

Concentrating on JSON block format, the format files produced are a significant fraction of an order of magnitude larger than binary formats. The impact on file size after compression is as might be expected from that starting point; the stronger compression produces files that are 150% of the size of similarly compressed binary format, and require over 4x more CPU to compress.

C.4. Performance

Concentrating again on the block formats, all three produce format files that are close to an order of magnitude smaller than the original "test.pcap" file. CBOR produces the largest files and Avro the smallest, 20% smaller than CBOR.

However, once compression is taken into account, the size difference narrows. At medium compression (with gzip), the size difference is 4%. Using strong compression (with xz) the difference reduces to 2%, with Avro the largest and Protocol Buffers the smallest, although CBOR and Protocol Buffers require slightly more compression CPU.

The measurements presented above do not include data on the CPU required to generate the format files. Measurements indicate that writing Avro requires 10% more CPU than CBOR or Protocol Buffers. It appears, therefore, that Avro's advantage in compression CPU usage is probably offset by a larger CPU requirement in writing Avro.

C.5. Conclusions

The above assessments lead us to the choice of a binary format file using blocking.

As noted previously, this draft anticipates that output data will be subject to compression. There is no compelling case for one particular binary serialisation format in terms of either final file size or machine resources consumed, so the choice must be largely based on other factors. CBOR was therefore chosen as the binary serialisation format for the reasons listed in Section 5.

C.6. Block size choice

Given the choice of a CBOR format using blocking, the question arises of what an appropriate default value for the maximum number of query/response pairs in a block should be. This has two components; what is the impact on performance of using different block sizes in the format file, and what is the impact on the size of the format file before and after compression.

The following table addresses the performance question, showing the impact on the performance of a C++ program converting "test.pcap" to C-DNS. File size is in Mb, resident set size (RSS) in kb.

Block size	File size	RSS	User time
1000	133.46	612.27	15.25
5000	89.85	676.82	14.99
10000	76.87	752.40	14.53
20000	67.86	750.75	14.49
40000	61.88	736.30	14.29
80000	58.08	694.16	14.28
160000	55.94	733.84	14.44
320000	54.41	799.20	13.97

Increasing block size, therefore, tends to increase maximum RSS a little, with no significant effect (if anything a small reduction) on CPU consumption.

The following table demonstrates the effect of increasing block size on output file size for different compressions.

Block size	None	snzip	lz4	gzip	zstd	xz
1000	133.46	90.52	90.03	74.65	44.78	25.63
5000	89.85	59.69	59.43	46.99	37.33	22.34
10000	76.87	50.39	50.28	38.94	33.62	21.09
20000	67.86	43.91	43.90	33.24	32.62	20.16
40000	61.88	39.63	39.69	29.44	28.72	19.52
80000	58.08	36.93	37.01	27.05	26.25	19.00
160000	55.94	35.10	35.06	25.44	24.56	19.63
320000	54.41	33.87	33.74	24.36	23.44	18.66

There is obviously scope for tuning the default block size to the compression being employed, traffic characteristics, frequency of output file rollover etc. Using a strong compression scheme, block sizes over 10,000 query/response pairs would seem to offer limited improvements.

Authors' Addresses

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

Email: jad@sinodun.com

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

Email: jim@sinodun.com

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

Email: sara@sinodun.com

Terry Manderson
ICANN
12025 Waterfront Drive
Suite 300
Los Angeles CA 90094-2536

Email: terry.manderson@icann.org

John Bond
ICANN
12025 Waterfront Drive
Suite 300
Los Angeles CA 90094-2536

Email: john.bond@icann.org

DNSOP
Internet-Draft
Intended status: Standards Track
Expires: April 23, 2019

G. Huston
J. Damas
APNIC
W. Kumari
Google
October 20, 2018

A Root Key Trust Anchor Sentinel for DNSSEC
draft-ietf-dnsop-kskroll-sentinel-17

Abstract

The DNS Security Extensions (DNSSEC) were developed to provide origin authentication and integrity protection for DNS data by using digital signatures. These digital signatures can be verified by building a chain of trust starting from a trust anchor and proceeding down to a particular node in the DNS. This document specifies a mechanism that will allow an end user and third parties to determine the trusted key state for the root key of the resolvers that handle that user's DNS queries. Note that this method is only applicable for determining which keys are in the trust store for the root key.

[This document is being collaborated on in Github at:
<https://github.com/APNIC-Labs/draft-kskroll-sentinel>. The most recent version of the document, open issues, etc should all be available here. The authors (gratefully) accept pull requests. RFC Editor, please remove text in square brackets before publication.]

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 April 23, 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
1.1. Terminology	4
2. Sentinel Mechanism in Resolvers	4
2.1. Preconditions	5
2.2. Special Processing	5
3. Sentinel Tests for a Single DNS Resolver	6
3.1. Forwarders	9
4. Sentinel Tests for Multiple Resolvers	10
4.1. Test Scenario and Objective	10
4.2. Test Assumptions	10
4.3. Test Procedure	11
5. Security Considerations	12
6. Privacy Considerations	13
7. Implementation Experience	13
8. IANA Considerations	14
9. Acknowledgements	14
10. Change Log	15
11. References	19
11.1. Normative References	19
11.2. Informative References	19
Appendix A. Protocol Walkthrough Example	20
Authors' Addresses	23

1. Introduction

The DNS Security Extensions (DNSSEC) [RFC4033], [RFC4034] and [RFC4035] were developed to provide origin authentication and integrity protection for DNS data by using digital signatures. DNSSEC uses Key Tags to efficiently match signatures to the keys from which they are generated. The Key Tag is a 16-bit value computed from the RDATA of a DNSKEY RR as described in Appendix B of

[RFC4034]. RRSIG RRs contain a Key Tag field whose value is equal to the Key Tag of the DNSKEY RR that was used to generate the corresponding signature.

This document specifies how security-aware DNS resolvers that perform validation of their responses can respond to certain queries in a manner that allows an agent performing the queries to deduce whether a particular key for the root has been loaded into that resolver's trusted key store. This document also describes a procedure where a collection of resolvers can be tested to determine if at least one of these resolvers has loaded a given key into its trusted key store. These tests can be used to determine whether a certain root zone Key Signing Key (KSK) is ready to be used as a trusted key, within the context of a planned root zone KSK key roll.

There are two primary use cases for this mechanism:

- o Users may wish to ascertain whether their DNS resolution environment's resolver is ready for an upcoming root KSK rollover.
- o Researchers want to perform Internet-wide studies about the proportion of users who will be negatively impacted by an upcoming root KSK rollover.

The mechanism described in this document satisfy the requirements of both these use-cases. This mechanism is OPTIONAL to implement and use. If implemented, this mechanism SHOULD be enabled by default to facilitate Internet-wide measurement. Configuration options MAY be provided to disable the mechanism for reasons of local policy.

The KSK sentinel tests described in this document use a test comprising of a set of DNS queries to domain names that have special values for the left-most label. The test relies on recursive resolvers supporting a mechanism that recognises this special pattern in queries, and under certain defined circumstances will return a DNS SERVFAIL response code (RCODE 2), mimicking the response code that is returned by security-aware resolvers when DNSSEC validation fails.

If a browser or operating system is configured with multiple resolvers, and those resolvers have different properties (for example, one performs DNSSEC validation and one does not), the sentinel test described in this document can still be used. The sentinel test makes a number of assumptions about DNS resolution behaviour that may not necessarily hold in all environments; if these assumptions do not hold (such as, for example, requiring the stub resolver to query the next recursive resolver in the locally configured set upon receipt of a SERVFAIL response code) then this

test may produce indeterminate or inconsistent results. In some cases where these assumptions do not hold, repeating the same test query set may generate different results.

Note that the measurements facilitated by the mechanism described in this document are different from those of [RFC8145]. RFC 8145 relies on resolvers reporting towards the root servers a list of locally cached trust anchors for the root zone. Those reports can be used to infer how many resolvers may be impacted by a KSK roll, but not what the user impact of the KSK roll will be.

1.1. 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.

This document contains a number of terms related to the DNS. The current definitions of these terms can be found in [RFC7719].

2. Sentinel Mechanism in Resolvers

DNSSEC-Validating resolvers that implement this mechanism MUST perform validation of responses in accordance with the DNSSEC response validation specification [RFC4035].

This sentinel mechanism makes use of two special labels:

- o root-key-sentinel-is-ta-<key-tag>
- o root-key-sentinel-not-ta-<key-tag>

These labels trigger special processing in the validating DNS resolver when responses from authoritative servers are received. Labels containing "root-key-sentinel-is-ta-<key-tag>" is used to answer the question "Is this the Key Tag of a key which the validating DNS resolver is currently trusting as a trust anchor?" Labels containing "root-key-sentinel-not-ta-<key-tag>" is used to answer the question "Is this the Key Tag of a key which the validating DNS resolver is **not** currently trusting as a trust anchor?".

The special labels defined here came after extensive IETF evaluation of alternative patterns and approaches in light of the desired behaviour (sections 2.1, 2.2) within the resolver and the applied testing methodology (section 4.3). As one example, underscore

prefixed names were rejected because some browsers and operating systems would not fetch them because they domain names but not valid hostnames (see [RFC7719] for these definitions). Attention was paid to the consideration of local collisions and the reservation of leftmost labels of a domain name, and the impact upon zone operators who might desire to use a similarly constructed hostname for a purpose other than as documented here. Therefore, it is important to note that the reservation of the labels in this manner is definitely not considered "best practice".

2.1. Preconditions

All of the following conditions must be met to trigger special processing inside resolver code:

- o The DNS response is DNSSEC validated.
- o The result of validation is "Secure".
- o The EDNS(0) Checking Disabled (CD) bit in the query is not set.
- o The QTYPE is either A or AAAA (Query Type value 1 or 28).
- o The OPCODE is QUERY.
- o The leftmost label of the original QNAME (the name sent in the Question Section in the original query) is either "root-key-sentinel-is-ta-<key-tag>" or "root-key-sentinel-not-ta-<key-tag>".

If any one of the preconditions is not met, the resolver MUST NOT alter the DNS response based on the mechanism in this document.

Note that the <key-tag> is specified in the DNS label as unsigned decimal integer (as described in [RFC4034], section 5.3), but zero-padded to five digits (for example, a Key Tag value of 42 would be represented in the label as 00042). The precise specification of the special labels above should be followed exactly. For example, a label that does not include a Key Tag zero-padded to five digits does not match this specification, and should not be processed as if they did -- in other words, such queries should be handled as any other label and not according to Section 2.2.

2.2. Special Processing

Responses which fulfil all of the preconditions in Section 2.1 require special processing, depending on leftmost label in the QNAME.

First, the resolver determines if the numerical value of <key-tag> is equal to any of the Key Tag values of an active root zone KSK which is currently trusted by the local resolver and is stored in its store of trusted keys. An active root zone KSK is one which could currently be used for validation (that is, a key that is not in either the AddPend or Revoked state as described in [RFC5011]).

Second, the resolver alters the response being sent to the original query based on both the left-most label and the presence of a key with given Key Tag in the trust anchor store. Two labels and two possible states of the corresponding key generate four possible combinations summarized in the table:

Label	Key is trusted	Key is not trusted
is-ta	return original answer	return SERVFAIL
not-ta	return SERVFAIL	return original answer

Instruction "return SERVFAIL" means that the resolver MUST set RCODE=SERVFAIL (value 2) and the ANSWER section of the DNS response MUST be empty, ignoring all other documents which specify content of the ANSWER section.

Instruction "return original answer" means that the resolver MUST process the query without any further special processing; that is, exactly as if the mechanism described in this document was not implemented or was disabled. The answer for the A or AAAA query is sent on to the client.

3. Sentinel Tests for a Single DNS Resolver

This section describes the use of the sentinel detection mechanism against a single DNS recursive resolver in order to determine whether this resolver is using a particular trust anchor to validate DNSSEC-signed responses.

Note that the test in this section applies to a single DNS resolver. The test described in Section 4 applies instead to a collection of DNS resolvers, as might be found in the DNS configuration of an end-user environment.

The critical aspect of the DNS names used in this mechanism is that they contain the specified label for either the positive and negative test as the left-most label in the query name.

The sentinel detection procedure can test a DNS resolver using three queries:

- o A query name containing the left-most label "root-key-sentinel-is-ta-<key-tag>". This corresponds to a validly-signed name in the parent zone, so that responses associated with this query name can be authenticated by a DNSSEC-validating resolver. Any validly-signed DNS zone can be used as the parent zone for this test.
- o A query name containing the left-most label "root-key-sentinel-not-ta-<key-tag>". This also corresponds to a validly-signed name. Any validly-signed DNS zone can be used as the parent zone for this test.
- o A query name that is signed with a DNSSEC signature that cannot be validated (described as a "bogus" RRset in Section 5 of [RFC4033], when, for example, an RRset associated with a zone that is not signed with a valid RRSIG record).

The responses received from queries to resolve each of these query names can be evaluated to infer a trust key state of the DNS resolver.

An essential assumption here is that this technique relies on security-aware (DNSSEC validating) resolvers responding with a SERVFAIL response code to queries where DNSSEC checking is requested and the response cannot be validated. Note that other issues can also cause a resolver to return SERVFAIL responses, and so the sentinel processing may sometimes result in incorrect or indeterminate conclusions.

To describe this process of classification, DNS resolvers are classified by five distinct behavior types using the labels: "Vnew", "Vold", "Vind", "nonV", and "other". These labels correspond to resolver system behaviour types as follows:

Vnew: A DNS resolver that is configured to implement this mechanism and has loaded the nominated key into their local trusted key stores will respond with an A or AAAA RRset response for the associated "root-key-sentinel-is-ta" queries, SERVFAIL for "root-key-sentinel-not-ta" queries and SERVFAIL for the signed name queries that return "bogus" validation status.

Vold: A DNS resolver that is configured to implement this mechanism and has not loaded the nominated key into their local trusted key stores will respond with an SERVFAIL for the associated "root-key-sentinel-is-ta" queries, an A or AAAA RRset response for "root-key-sentinel-not-ta" queries and SERVFAIL for the signed name queries that return "bogus" validation status.

Vind: A DNS resolver that has is not configured to implement this mechanism will respond with an A or AAAA RRset response for "root-key-sentinel-is-ta", an A or AAAA RRset response for "root-key-sentinel-not-ta" and SERVFAIL for the name that returns "bogus" validation status. This set of responses does not give any information about the trust anchors used by this resolver.

nonV: A non-security-aware DNS resolver will respond with an A or AAAA RRset response for "root-key-sentinel-is-ta", an A or AAAA RRset response for "root-key-sentinel-not-ta" and an A or AAAA RRset response for the name that returns "bogus" validation status.

other: There is the potential to admit other combinations of responses to these three queries. While this may appear self-contradictory, there are cases where such an outcome is possible. For example, in DNS resolver farms what appears to be a single DNS resolver that responds to queries passed to a single IP address is in fact constructed as a collection of slave resolvers, and the query is passed to one of these internal resolver engines. If these individual slave resolvers in the farm do not behave identically, then other sets of results can be expected from these three queries. In such a case, no determination about the capabilities of this DNS resolver farm can be made.

Note that SERVFAIL might be cached according to Section 7 of [RFC2308] for up to 5 minutes and a positive answer for up to its TTL.

If a client directs these three queries to a single resolver, the responses should allow the client to determine the capability of the resolver, and if it supports this sentinel mechanism, whether or not it has a particular key in its trust anchor store, as in the following table:

		Query		
		is-ta	not-ta	bogus
Type	Vnew	Y	SERVFAIL	SERVFAIL
	Vold	SERVFAIL	Y	SERVFAIL
	Vind	Y	Y	SERVFAIL
	nonV	Y	Y	Y
	other	*	*	*

In this table, the "Y" response denotes an A or AAAA RRset response (depending on the query type of A or AAAA records), "SERVFAIL"

denotes a DNS SERVFAIL response code (RCODE 2), and "*" denotes either response.

Vnew: The nominated key is trusted by the resolver.

Vold: The nominated key is not yet trusted by the resolver.

Vind: There is no information about the trust anchors of the resolver.

nonV: The resolver does not perform DNSSEC validation.

other: The properties of the resolver cannot be analyzed by this protocol.

3.1. Forwarders

Some resolvers are configured not to answer queries using the recursive algorithm first described in [RFC1034] section 4.3.2, but instead relay queries to one or more other resolvers. Resolvers configured in this manner are referred to in this document as "forwarders".

If the resolver is non-validating, and it has a single forwarder, then the resolver will presumably mirror the capabilities of the forwarder's target resolver.

If the validating resolver has a forwarding configuration, and it sets the EDNS(0) Checking Disabled (CD) bit as described in Section 3.2.2 of [RFC4035] on all forwarded queries, then this resolver is acting in a manner that is identical to a standalone resolver.

A more complex case is where all of the following conditions hold:

- o Both the validating resolver and the forwarder target resolver support this trusted key sentinel mechanism
- o The local resolver's queries do not have the EDNS(0) CD bit set
- o The trusted key state differs between the forwarding resolver and the forwarder's target resolver

In such a case, either the outcome is indeterminate validating ("Vind"), or a case of mixed signals such as SERVFAIL in all three responses, ("other") which is similarly an indeterminate response with respect to the trusted key state.

4. Sentinel Tests for Multiple Resolvers

The description in Section 3 describes a trust anchor test that can be used in the simple situation where the test queries were being passed to a single recursive resolver that directly queried authoritative name servers.

However, the common end-user scenario is where a user's local DNS resolution environment is configured to use more than one recursive resolver. The single resolver test technique will not function reliably in such cases, as a a SERVFAIL response from one resolver may cause the local stub resolver to repeat the query against one of the other configured resolvers and the results may be inconclusive.

In describing a test procedure that can be used in this environment of a set of DNS resolvers there are some necessary changes to the nature of the question that this test can answer, the assumptions about the behaviour of the DNS resolution environment, and some further observations about potential variability in the test outcomes.

4.1. Test Scenario and Objective

This test is not intended to expose which trust anchors are used by any single DNS resolver.

The test scenario is explicitly restricted to that of the KSK environment where a current active KSK (called "KSK-current") is to be replaced with a new KSK (called "KSK-new"). The test is designed to be run between when KSK-new is introduced into the root zone and when the root zone is signed with KSK-new.

The objective of the test is to determine if the user will be negatively impacted by the KSK roll. A "negative impact" for the user is defined such that all the configured resolvers are security-aware resolvers that perform validation of DNSSEC-signed responses, and none of these resolvers have loaded KSK-new into their local trust anchor set. In this situation, it is anticipated that once the KSK is rolled the entire set of the user's resolvers will not be able to validate the contents of the root zone and the user is likely to lose DNS service as a result of this inability to perform successful DNSSEC validation.

4.2. Test Assumptions

There are a number of assumptions about the DNS environment used in this test. Where these assumptions do not hold, the results of the test will be indeterminate.

- o When a recursive resolver returns SERVFAIL to the user's stub resolver, the stub resolver will send the same query to the next resolver in the locally configured resolver set. It will continue to do this until it gets a non-SERVFAIL response or until it runs out of resolvers to try.
- o When the user's stub resolver passes a query to a resolver in the configured resolver set, it will get a consistent answer over the timeframe of the queries. This assumption implies that if the same query is asked by the same stub resolver multiple times in succession to the same recursive resolver, the recursive resolver's response will be the same for each of these queries.
- o All DNSSEC-validating resolvers have KSK-current in their local trust anchor cache.

There is no current published measurement data that indicates to what extent the first two assumptions listed here are valid, and how many end users may be impacted by these assumptions. In particular, the first assumption, that a consistent SERVFAIL response will cause the local stub DNS resolution environment to query all of its configured recursive resolvers before concluding that the name cannot be resolved, is a very critical assumption for this test.

Note that additional precision / determinism may be achievable by bypassing the normal OS behavior and explicitly testing using each configured recursive resolver (e.g using 'dig').

4.3. Test Procedure

The sentinel detection process tests a DNS resolution environment with three query names. Note that these same general categories of query as in Section 3 but the key tag used is different for some queries:

- o A query name that is signed with a DNSSEC signature that cannot be validated (described as a "bogus" RRset in Section 5 of [RFC4033], when, for example, an RRset is not signed with a valid RRSIG record).
- o A query name containing the left-most label "root-key-sentinel-not-ta-<key-tag-of-KSK-current>". This name MUST be a validly-signed name. Any validly-signed DNS zone can be used for this test.
- o A query name containing the left-most label "root-key-sentinel-is-ta-<key-tag-of-KSK-new>". This name MUST be a validly-signed name. Any validly-signed DNS zone can be used for this test.

The responses received from queries to resolve each of these names can be evaluated to infer a trust key state of the user's DNS resolution environment.

The responses to these queries are described using a simplified notation. Each query will either result in a SERVFAIL response (denoted as "S"), indicating that all of the resolvers in the recursive resolver set returned the SERVFAIL response code, or result in a response with the desired RRset value (denoted as "A"). The queries are ordered by the "invalid" name, the "root-key-sentinel-not-ta" label, then the "root-key-sentinel-is-ta" label, and a triplet notation denotes a particular response. For example, the triplet "(S S A)" denotes a SERVFAIL response to the invalid query, a SERVFAIL response to the "root-key-sentinel-not-ta" query and a RRset response to the "root-key-sentinel-is-ta" query.

The set of all possible responses to these three queries are:

- (A * *): If any resolver returns an "A" response for the query for the invalid name, then the resolver set contains at least one non-validating DNS resolver, and the user will not be impacted by the KSK roll.
- (S A *): If any of the resolvers returns an "A" response to the "root-key-sentinel-not-ta" query, then at least one of the resolvers does not recognise the sentinel mechanism, and the behaviour of the collection of resolvers during the KSK roll cannot be reliably determined.
- (S S A): This case implies that all of the resolvers in the set perform DNSSEC-validation, all of the resolvers are aware of the sentinel mechanism, and at least one resolver has loaded KSK-new as a local trust anchor. The user will not be impacted by the KSK roll.
- (S S S): This case implies that all of the resolvers in the set perform DNSSEC-validation, all of the resolvers are aware of the sentinel mechanism, and none of the resolvers has loaded KSK-new as a local trust anchor. The user will be negatively impacted by the KSK roll.

5. Security Considerations

This document describes a mechanism to allow users to determine the trust anchor state of root zone key signing keys in the DNS resolution system that they use. If the user executes third party code, then this information may also be available to the third party.

The mechanism does not require resolvers to set otherwise unauthenticated responses to be marked as authenticated, and does not alter the security properties of DNSSEC with respect to the interpretation of the authenticity of responses that are so marked.

The mechanism does not require any further significant processing of DNS responses, and queries of the form described in this document do not impose any additional load that could be exploited in an attack over the normal DNSSEC validation processing load.

6. Privacy Considerations

The mechanism in this document enables third parties (with either good or bad intentions) to learn something about the security configuration of recursive DNS resolvers. That is, someone who can cause an Internet user to make specific DNS queries (e.g. via web-based advertisements or javascript in web pages), can, under certain specific circumstances that includes additional knowledge of the resolvers that are invoked by the user, determine which trust anchors are configured in these resolvers. Without this additional knowledge, the third party can infer the aggregate capabilities of the user's DNS resolution environment, but cannot necessarily infer the trust configuration of any recursive name server.

7. Implementation Experience

[RFC Editor: Please remove before publication. As this section will be removed, it is more conversational than would appear in a published doc.]

List of known resolver implementations (alphabetical):

BIND Ondrej Sury of ISC reported to the DNSOP Working Group in April 2018 that this technique was peer-reviewed and merged into BIND master branch with the intent to backport the feature into older release branches. The merge request:
https://gitlab.isc.org/isc-projects/bind9/merge_requests/123
Information on configuring this can be found in the BIND 9.13.0 Administrator Reference Manual (ARM), available at
<https://ftp.isc.org/isc/bind9/9.13.0/doc/arm/Bv9ARM.pdf>

Knot resolver Petr Spacek implemented early versions of this technique into the Knot resolver, identified a number of places where it wasn't clear, and provided very helpful text to address these issues and make the document more clear. Petr also identified an embarrassingly large number of typos (and similar) in the ksk-test setup. More information is at <http://knot->

resolver.readthedocs.io/en/stable/modules.html#sentinel-for-detecting-trusted-keys

Unbound Benno Overeinder of NLnet Labs reported to the DNSOP Working Group in April 2018 an intention to support this technique in Unbound in the near future. This is now implemented in Unbound version 1.7.1, available from <http://unbound.nlnetlabs.nl/download.html>. Configuration information is at <http://unbound.nlnetlabs.nl/documentation/unbound.conf.html>

A (partial) list of "client" / user side implementations (the author was keeping a more complete list of implementations, but has misplaced it - apologies, I'm happy to re-add them if you send me a note.):

<http://www.ksk-test.net> An Javascript implementation of the client side of this protocol is available at: <http://www.ksk-test.net>

<http://test.kskroll.dnssec.lab.nic.cl/> Hugo Salgado-Hernandez has created an implementation at <http://test.kskroll.dnssec.lab.nic.cl/>

<http://sentinel.research.icann.org/> The code for this implementation is published at <https://github.com/paulehoffman/sentinel-testbed>

<http://www.bellis.me.uk/sentinel/> Ray Bellis client implementation - <http://www.bellis.me.uk/sentinel/>

8. IANA Considerations

This document has no IANA actions.

9. Acknowledgements

This document has borrowed extensively from [RFC8145] for the introductory text, and the authors would like to acknowledge and thank the authors of that document both for some text excerpts and for the more general stimulation of thoughts about monitoring the progress of a roll of the KSK of the root zone of the DNS.

The authors would like to thank Joe Abley, Mehmet Akcin, Mark Andrews, Richard Barnes, Ray Bellis, Stephane Bortzmeyer, David Conrad, Ralph Dolmans, John Dickinson, Steinar Haug, Bob Harold, Wes Hardaker, Paul Hoffman, Matt Larson, Jinmei Tatuya, Edward Lewis, George Michaelson, Benno Overeinder, Matthew Pounsett, Hugo Salgado-Hernandez, Andreas Schulze, Mukund Sivaraman, Petr Spacek, Job Snijders, Andrew Sullivan, Ondrej Sury, Paul Vixie, Duane Wessels and Paul Wouters for their helpful feedback.

The authors would like to especially call out Paul Hoffman and Duane Wessels for providing comments in the form of pull requests. Joe Abley also helpfully provided extensive review and OLD / NEW text.

Petr Spacek wrote some very early implementations, and provided significant feedback (including pointing out when the test bed didn't match the document!)

10. Change Log

RFC Editor: Please remove this section!

Note that this document is being worked on in GitHub - see Abstract. The below is mainly large changes, and is not authoritative.

From -16 to -17:

- o Thank to Paul Hoffman for "Lots of editorial fixes for post-IESG draft" (<https://github.com/APNIC-Labs/draft-kskroll-sentinel/pull/28>)
- o Repeat after me: Do not drive git while on cold meds...

From -15 to -16:

- o Addressed IESG comments
- o Benjamin Kaduk's Discuss on draft-ietf-dnsop-kskroll-sentinel
- o Also added Terry's "This a bad design pattern, but we decided the benefits outweigh the costs this time." text.
- o Suggestion from Adam to clarify that bypassing e.g gethostbyname() can provide better testing.
- o Nit: Forgot 'name' in 'This name MUST be a validly-signed name.'
- o Clarified that 'bogus.example.com' is intentionally DNSSEC bogus / invalid.

From -14 to -15:

- o Addressed Joe Abley's thorough review, at: <https://mailarchive.ietf.org/arch/msg/dnsop/8ZnN1xj55Yimet2cg-LrdoJafEA>

From -13 to -14:

- o Addressed nits from Bob Harold -
<https://mailarchive.ietf.org/arch/msg/dnsop/j4Serw0z24o470AnlD8ISo8o9k4>
- o Formatting changes (and a bit more text) in the implementation section.
- o Closes PR #21: Clarify indeterminate and resolution systems,
- o Closes PR #22: Updates to -13 describing the test procedure for a set of resolvers
- o Closes PR #23: Fix sundry typos,
- o Closes PR #24: Editorial and clarifications to the new text
- o Closes PR #25: Clarified when the test can be run

From -12 to -13:

- o Merged Paul Hoffmans PR#19, PR#20.
- o Moved toy ksk-test.net to implementation section.
- o Split the test procedures between the test of a single DNS resolvers and the test of a collection of DNS resolvers as would be found in an end user environment.

From -11 to -12:

- o Moved the Walkthrough Example to the end of the document as an appendix.
- o Incorporated changes as proposed by Ondrej Sury, relating to a consistent use of Key Tag and a reference to the definition of a Bogus RRset.
- o Corrected minor typos.
- o Revised the Privacy Considerations.
- o In response to a request from DNSOP Working Group chairs, a section on reported Implementation Experience has been added, based on postings to the DNSOP Working Group mailing list.

From -10 to -11:

- o Clarified the preconditions for this mechanism as per Working Group mailing list discussion.
- o Corrected minor typo.

From -09 to -10:

- o Clarified the precondition list to specify that the resolver had performed DNSSEC-validation by setting the AD bit in the response
- o Clarified the language referring to the operation of RFC8145 signalling.

From -08 to -09:

- o Incorporated Paul Hoffman's PR # 15 (Two issues from the Hackathon) - <https://github.com/APNIC-Labs/draft-kskroll-sentinel/pull/15>
- o Clarifies that the match is on the *original* QNAME.

From -08 to -07:

- o Changed title from "A Sentinel for Detecting Trusted Keys in DNSSEC" to "A Root Key Trust Anchor Sentinel for DNSSEC".
- o Changed magic string from "kskroll-sentinel-" to "root-key-sentinel-" -- this time for sure, Rocky!

From -07 to -06:

- o Addressed GitHub PR #14: Clarifications regarding caching and SERVFAIL responses
- o Addressed GitHub PR #12, #13: Clarify situation with multiple resolvers, Fix editorial nits.

From -05 to -06:

- o Paul improved my merging of Petr's text to make it more readable. Minor change, but this is just before the cut-off, so I wanted it maximally readable.

From -04 to -05:

- o Incorporated Duane's #10

- o Integrated Petr Spacek's Issue - <https://github.com/APNIC-Labs/draft-kskroll-sentinel/issues/9> (note that commit-log incorrectly referred to Duane's PR as number 9, it is actually 10).

From -03 to -04:

- o Addressed GitHub pull requests #4, #5, #6, #7 #8.
- o Added Duane's privacy concerns
- o Makes the use cases clearer
- o Fixed some A/AAAA stuff
- o Changed the example numbers
- o Made it clear that names and addresses must be real

From -02 to -03:

- o Integrated / published comments from Paul in GitHub PR #2 - <https://github.com/APNIC-Labs/draft-kskroll-sentinel/pull/2>
- o Made the Key Tag be decimal, not hex (thread / consensus in https://mailarchive.ietf.org/arch/msg/dnsop/Kg7AtDhFRNw31He8n0_bMr9hBuE)

From -01 to 02:

- o Removed Address Record definition.
- o Clarified that many things can cause SERVFAIL.
- o Made examples FQDN.
- o Fixed a number of typos.
- o Had accidentally said that Charlie was using a non-validating resolver in example.
- o [TODO(WK): Doc says Key Tags are hex, is this really what the WG wants?]
- o And active key is one that can be used *now* (not e.g AddPend)

From -00 to 01:

- o Added a conversational description of how the system is intended to work.
- o Clarification that this is for the root.
- o Changed the label template from `_is-ta-<key-tag>` to `kskroll-sentinel-is-ta-<key-tag>`. This is because BIND (at least) will not allow records which start with an underscore to have address records (CNAMEs, yes, A/AAAA no). Some browsers / operating systems also will not fetch resources from names which start with an underscore.

11. References

11.1. 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>>.
- [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>>.
- [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>>.
- [RFC5011] StJohns, M., "Automated Updates of DNS Security (DNSSEC) Trust Anchors", STD 74, RFC 5011, DOI 10.17487/RFC5011, September 2007, <<https://www.rfc-editor.org/info/rfc5011>>.

11.2. Informative References

- [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>>.

[RFC8145] Wessels, D., Kumari, W., and P. Hoffman, "Signaling Trust Anchor Knowledge in DNS Security Extensions (DNSSEC)", RFC 8145, DOI 10.17487/RFC8145, April 2017, <<https://www.rfc-editor.org/info/rfc8145>>.

Appendix A. Protocol Walkthrough Example

This Appendix provides a non-normative example of how the sentinel mechanism could be used, and what each participant does. It is provided in a conversational tone to be easier to follow. The examples here all assume that each person has just one resolver, or a system of resolvers that have the same properties.

Alice is in charge of the DNS root KSK (Key Signing Key), and would like to roll / replace the key with a new one. She publishes the new KSK, but would like to be able to predict / measure what the impact will be before removing/revoking the old key. The current KSK has a Key Tag of 11112, the new KSK has a Key Tag of 02323. Users want to verify that their resolver will not break after Alice rolls the root KSK key (that is, starts signing with just the KSK whose Key Tag is 02323).

Bob, Charlie, Dave, Ed are all users. They use the DNS recursive resolvers supplied by their ISPs. They would like to confirm that their ISPs have picked up the new KSK. Bob's ISP does not perform validation. Charlie's ISP does validate, but the resolvers have not yet been upgraded to support this mechanism. Dave and Ed's resolvers have been upgraded to support this mechanism; Dave's resolver has the new KSK, Ed's resolver hasn't managed to install the 02323 KSK in its trust store yet.

Geoff is a researcher, and would like to both provide a means for Bob, Charlie, Dave and Ed to be able to perform tests, and also would like to be able to perform Internet-wide measurements of what the impact will be (and report this back to Alice).

Geoff sets an authoritative DNS server for example.com, and also a webserver (www.example.com). He adds three address records to example.com:

```
bogus.example.com. IN AAAA 2001:db8::1
root-key-sentinel-is-ta-02323.example.com. IN AAAA 2001:db8::1
root-key-sentinel-not-ta-11112.example.com. IN AAAA 2001:db8::1
```

Note that the use of "example.com" names and the addresses here are examples, and "bogus" intentionally has invalid DNSSEC signatures.

In a real deployment, the domain names need to be under control of the researcher, and the addresses must be real, reachable addresses.

Geoff then DNSSEC signs the example.com zone, and intentionally makes the bogus.example.com record have bogus validation status (for example, by editing the signed zone and entering garbage for the signature). Geoff also configures his webserver to listen on 2001:db8::1 and serve a resource (for example, a 1x1 GIF, 1x1.gif) for all of these names. The webserver also serves a webpage (www.example.com) which contains links to these 3 resources (http://bogus.example.com/1x1.gif, http://root-key-sentinel-is-ta-02323.example.com/1x1.gif, http://root-key-sentinel-not-ta-11112.example.com/1x1.gif).

Geoff then asks Bob, Charlie, Dave and Ed to browse to www.example.com. Using the methods described in this document, the users can figure out what their fate will be when the 11112 KSK is removed.

Bob is not using a validating resolver. This means that he will be able to resolve bogus.example.com (and fetch the 1x1 GIF) - this tells him that the KSK roll does not affect him, and so he will be OK.

Charlie's resolvers are validating, but they have not been upgraded to support the KSK sentinel mechanism. Charlie will not be able to fetch the http://bogus.example.com/1x1.gif resource (the bogus.example.com record is bogus, and none of his resolvers will resolve it). He is able to fetch both of the other resources - from this he knows (see the logic in the body of this document) that he is using validating resolvers, but at least one of these resolvers is not configured to perform sentinel processing. The KSK sentinel method cannot provide him with a definitive answer to the question of whether he will be impacted by the KSK roll.

Dave's resolvers implement the sentinel method, and have picked up the new KSK. For the same reason as Charlie, he cannot fetch the "bogus" resource. His resolver resolves the root-key-sentinel-is-ta-02323.example.com name normally (it contacts the example.com authoritative servers, etc); as it supports the sentinel mechanism, just before Dave's recursive resolver sends the reply to Dave's stub, it performs the KSK Sentinel check. The QNAME starts with "root-key-sentinel-is-ta-", and the recursive resolver does indeed have a key with the Key Tag of 02323 in its root trust store. This means that that this part of the KSK Sentinel check passes (it is true that Key Tag 02323 is in the trust anchor store), and the recursive resolver replies normally (with the answer provided by the authoritative server). Dave's recursive resolver then resolves the root-key-

sentinel-not-ta-11112.example.com name. Once again, it performs the normal resolution process, but because it implements KSK Sentinel (and the QNAME starts with "root-key-sentinel-not-ta-"), just before sending the reply, it performs the KSK Sentinel check. As it has the key with key-tag 11112 in its trust anchor store, the answer to "is this *not* a trust anchor" is false, and so the recursive resolver does not reply with the answer from the authoritative server - instead, it replies with a SERVFAIL (note that replying with SERVFAIL instead of the original answer is the only mechanism that KSK Sentinel uses). This means that Dave cannot fetch "bogus", he can fetch "root-key-sentinel-is-ta-02323", but he cannot fetch "root-key-sentinel-not-ta-11112". From this, Dave knows that he is behind a collection of resolvers that all validate, all have the key with key tag 11112 loaded and at least one of these resolvers has loaded the key with key-tag 02323 into its local trust anchor cache, Dave will not be impacted by the KSK roll.

Just like Charlie and Dave, Ed cannot fetch the "bogus" record. This tells him that his resolvers are validating. When his (sentinel-aware) resolvers perform the KSK Sentinel check for "root-key-sentinel-is-ta-02323", none of them have loaded the new key with key-tag 02323 in their local trust anchor store. This means check fails, and Ed's recursive resolver converts the (valid) answer into a SERVFAIL error response. It performs the same check for root-key-sentinel-not-ta-11112.example.com, and as all of Ed's resolvers both perform DNSSEC validation and recognise the sentinel label Ed will be unable to fetch the "root-key-sentinel-not-ta-11112" resource. This tells Ed that his resolvers have not installed the new KSK and he will be negatively impacted by the KSK roll..

Geoff would like to do a large scale test and provide the information back to Alice. He uses some mechanism such as causing users to go to a web page to cause a large number of users to attempt to resolve the three resources, and then analyzes the results of the tests to determine what percentage of users will be affected by the KSK rollover event.

This description is a simplified example - it is not anticipated that Bob, Charlie, Dave and Ed will actually look for the absence or presence of web resources; instead, the webpage that they load would likely contain JavaScript (or similar) which displays the result of the tests, sends the results to Geoff, or both. This sentinel mechanism does not rely on the web: it can equally be used by trying to resolve the names (for example, using the common "dig" command) and checking which result in a SERVFAIL.

Authors' Addresses

Geoff Huston

Email: gih@apnic.net

URI: <http://www.apnic.net>

Joao Silva Damas

Email: joao@apnic.net

URI: <http://www.apnic.net>

Warren Kumari

Email: warren@kumari.net

Network Working Group
Internet-Draft
Updates: 1034, 1035 (if approved)
Intended status: Standards Track
Expires: February 15, 2019

J. Abley
Afilias
O. Gudmundsson
M. Majkowski
Cloudflare Inc.
E. Hunt
ISC
August 14, 2018

Providing Minimal-Sized Responses to DNS Queries that have QTYPE=ANY
draft-ietf-dnsop-refuse-any-07

Abstract

The Domain Name System (DNS) specifies a query type (QTYPE) "ANY". The operator of an authoritative DNS server might choose not to respond to such queries for reasons of local policy, motivated by security, performance or other reasons.

The DNS specification does not include specific guidance for the behaviour of DNS servers or clients in this situation. This document aims to provide such guidance.

This document updates RFC 1034 and RFC 1035.

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 February 15, 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
1.1.	Terminology	3
2.	Motivations for Use of ANY Queries	3
3.	General Approach	4
4.	Behaviour of DNS Responders	4
4.1.	Answer with a Subset of Available RRSets	5
4.2.	Answer with a Synthesised HINFO RRSet	5
4.3.	Answer with Best Guess as to Intention	6
4.4.	Behaviour with TCP Transport	6
5.	Behaviour of DNS Initiators	6
6.	HINFO Considerations	7
7.	Updates to RFC 1034 and RFC 1035	7
8.	Implementation Experience	8
9.	Security Considerations	8
10.	IANA Considerations	8
11.	Acknowledgements	8
12.	References	9
12.1.	Normative References	9
12.2.	Informative References	9
12.3.	URIs	9
Appendix A.	Editorial Notes	10
A.1.	Change History	10
A.1.1.	draft-ietf-dnsop-refuse-any-07	10
A.1.2.	draft-ietf-dnsop-refuse-any-06	10
A.1.3.	draft-ietf-dnsop-refuse-any-05	10
A.1.4.	draft-ietf-dnsop-refuse-any-04	10
A.1.5.	draft-ietf-dnsop-refuse-any-03	10
A.1.6.	draft-ietf-dnsop-refuse-any-02	10
A.1.7.	draft-ietf-dnsop-refuse-any-01	11
A.1.8.	draft-ietf-dnsop-refuse-any-00	11
A.1.9.	draft-jabley-dnsop-refuse-any-01	11
A.1.10.	draft-jabley-dnsop-refuse-any-00	11
Authors' Addresses	11

1. Introduction

The Domain Name System (DNS) specifies a query type (QTYPE) "ANY". The operator of an authoritative DNS server might choose not to respond to such queries for reasons of local policy, motivated by security, performance or other reasons.

The DNS specification [RFC1034] [RFC1035] does not include specific guidance for the behaviour of DNS servers or clients in this situation. This document aims to provide such guidance.

1.1. Terminology

This document uses terminology specific to the Domain Name System (DNS), descriptions of which can be found in [RFC7719].

In this document, "ANY Query" refers to a DNS meta-query with QTYPE=ANY. An "ANY Response" is a response to such a query.

In this document, "conventional ANY response" means an ANY response that is constructed in accordance with the algorithm documented in section 4.3.2 of [RFC1034] and specifically without implementing any of the mechanisms described in this document.

In an exchange of DNS messages between two hosts, this document refers to the host sending a DNS request as the initiator, and the host sending a DNS response as the responder.

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. Motivations for Use of ANY Queries

ANY queries are legitimately used for debugging and checking the state of a DNS server for a particular name.

ANY queries are sometimes used as an attempt to reduce the number of queries needed to get information, e.g. to obtain MX, A and AAAA RRSets for a mail domain in a single query. There is no documented guidance available for this use case, however, and some implementations have been observed not to function as perhaps their developers expected. Implementers that assume that an ANY query will ultimately be received by an authoritative server and will fetch all existing RRSets, should include a fallback mechanism to use when that does not happen.

ANY queries are frequently used to exploit the amplification potential of DNS servers/resolvers using spoofed source addresses and UDP transport (see [RFC5358]). Having the ability to return small responses to such queries makes DNS servers less attractive amplifiers.

ANY queries are sometimes used to help mine authoritative-only DNS servers for zone data, since they are expected to return all RRSets for a particular query name. If a DNS operator prefers to reduce the potential for information leaks, they might choose not to send large ANY responses.

Some authoritative-only DNS server implementations require additional processing in order to send a conventional ANY response, and avoiding that processing expense might be desirable.

3. General Approach

This proposal provides a mechanism for an authority server to signal that conventional ANY queries are not supported for a particular QNAME, and to do so in such a way that is both compatible with and triggers desirable behaviour by unmodified clients (e.g. DNS resolvers).

Alternative proposals for dealing with ANY queries have been discussed. One approach proposed using a new RCODE to signal that an authoritative server did not answer ANY queries in the standard way. This approach was found to have an undesirable effect on both resolvers and authoritative-only servers; resolvers receiving an unknown RCODE would re-send the same query to all available authoritative servers, rather than suppress future such ANY queries for the same QNAME.

This proposal avoids that outcome by returning a non-empty RRSet in the ANY response, providing resolvers with something to cache and effectively suppressing repeat queries to the same or different authority servers.

4. Behaviour of DNS Responders

Below are the three different modes of behaviour by DNS responders when processing queries with QNAMEs that exist, QCLASS=IN and QTYPE=ANY. Operators/Implementers are free to choose whichever mechanism best suits their environment.

1. A DNS responder can choose to select one or a larger subset of the available RRSets at the QNAME.

2. A DNS responder can return a synthesised HINFO resource record. See Section 6 for discussion of the use of HINFO.
3. Resolver can try to give out the most likely records the requester wants. This is not always possible and the result might well be a large response.

Except as described below in this section, the DNS responder MUST follow the standard algorithms when constructing a response.

4.1. Answer with a Subset of Available RRSets

A DNS responder which receives an ANY query MAY decline to provide a conventional ANY response, or MAY instead send a response with a single RRSets (or a larger subset of available RRSets) in the answer section.

The RRSets returned in the answer section of the response MAY consist of a single RRSets owned by the name specified in the QNAME. Where multiple RRSets exist, the responder SHOULD choose a small subset of those available to reduce the amplification potential of the response.

If the zone is signed, appropriate RRSIG records MUST be included in the answer.

Note that this mechanism does not provide any signalling to indicate to a client that an incomplete subset of the available RRSets has been returned.

4.2. Answer with a Synthesised HINFO RRSets

If there is no CNAME present at the owner name matching the QNAME, the resource record returned in the response MAY instead be synthesised, in which case a single HINFO resource record SHOULD be returned. The CPU field of the HINFO RDATA SHOULD be set to RFCXXXX [note to RFC Editor, replace with RFC number assigned to this document]. The OS field of the HINFO RDATA SHOULD be set to the null string to minimize the size of the response.

The TTL encoded for the synthesised HINFO RR SHOULD be chosen by the operator of the DNS responder to be large enough to suppress frequent subsequent ANY queries from the same initiator with the same QNAME, understanding that a TTL that is too long might make policy changes relating to ANY queries difficult to change in the future. The specific value used is hence a familiar balance when choosing TTL for any RR in any zone, and be specified according to local policy.

If the DNS query includes DO=1 and the QNAME corresponds to a zone that is known by the responder to be signed, a valid RRSIG for the RRSets in the answer (or authority if answer is empty) section MUST be returned. In the case of DO=0, the RRSIG SHOULD be omitted.

A system that receives an HINFO response SHOULD NOT infer that the response was generated according to this specification and apply any special processing of the response, since in general it is not possible to tell with certainty whether the HINFO RRSet received was synthesised. In particular, systems SHOULD NOT rely upon the HINFO RDATA described in this section to distinguish between synthesised and non-synthesised HINFO RRSets.

4.3. Answer with Best Guess as to Intention

In some cases it is possible to guess what the initiator wants in the answer (but not always). Some implementations have implemented the spirit of this document by returning all RRSets of RRTYPE CNAME, MX, A and AAAA that are present at the owner name but suppressing others. This heuristic seems to work well in practice, satisfying the needs of some applications whilst suppressing other RRSets such as TXT and DNSKEY that can often contribute to large responses. Whilst some applications may be satisfied by this behaviour, the resulting responses in the general case are larger than the approaches described in Section 4.1 and Section 4.2.

As before, if the zone is signed and the DO bit is set on the corresponding query, an RRSIG RRSet MUST be included in the response.

4.4. Behaviour with TCP Transport

A DNS responder MAY behave differently when processing ANY queries received over different transport, e.g. by providing a conventional ANY response over TCP whilst using one of the other mechanisms specified in this document in the case where a query was received using UDP.

Implementers SHOULD provide configuration options to allow operators to specify different behaviour over UDP and TCP.

5. Behaviour of DNS Initiators

A DNS initiator which sends a query with QTYPE=ANY and receives a response containing an HINFO resource record or a single RRset, as described in Section 4, MAY cache the response in the normal way. Such cached resource records SHOULD be retained in the cache following normal caching semantics, as it would with any other response received from a DNS responder.

A DNS initiator MAY suppress queries with QTYPE=ANY in the event that the local cache contains a matching HINFO resource record with RDATA.CPU field, as described in Section 4. A DNS initiator MAY instead respond to such queries with the contents of the local cache in the usual way.

6. HINFO Considerations

It is possible that the synthesised HINFO RRSet in an ANY response, once cached by the initiator, might suppress subsequent queries from the same initiator with QTYPE=HINFO. Thus the use of HINFO in this proposal would hence have effectively mask the HINFO RRSet present in the zone.

Authority-server operators who serve zones that rely upon conventional use of the HINFO RRTYPE SHOULD sensibly choose the "single RRset" method described in this document or select another type.

The HINFO RRTYPE is believed to be rarely used in the DNS at the time of writing, based on observations made at recursive servers, authority servers and in passive DNS.

7. Updates to RFC 1034 and RFC 1035

This document extends the specification for processing ANY queries described in section 4.3.2 of [RFC1034].

It is important to note that returning a subset of available RRsets when processing an ANY query is legitimate and consistent with [RFC1035]; it can be argued that ANY does not always mean ALL, as used in section 3.2.3 of [RFC1035]. The main difference here is that the TC bit SHOULD NOT be set on the response indicating that this is not a complete answer.

This document describes optional behaviour for both DNS initiators and responders, and implementation of the guidance provided by this document is OPTIONAL.

RRSIG queries (i.e. queries with QTYPE=RRSIG) are similar to ANY queries in the sense that they have the potential to generate large responses as well as extra work for the responders that process them, e.g. in the case where signatures are generated on-the-fly. RRSIG RRsets are not usually obtained using such explicit queries, but are rather included in the responses for other RRsets that the RRSIGs cover. This document does not specify appropriate behaviour for RRSIG queries, but note that future such advice might well benefit

from consistency with and experience of the approaches for ANY queries described here.

8. Implementation Experience

In October 2015 Cloudflare Authoritative Name server implementation implemented the HINFO response. A few minor problems were reported and have since been resolved.

An implementation of the subset-mode response to ANY queries was implemented in NSD 4.1 in 2016.

An implementation of a single RRSset response to an ANY query was made for BIND9 by Tony Finch, and that functionality was subsequently made available in production releases starting in BIND 9.11.

9. Security Considerations

Queries with QTYPE=ANY are frequently observed as part of reflection attacks, since a relatively small query can be used to elicit a large response; this is a desirable characteristic if the goal is to maximize the amplification potential of a DNS server as part of a volumetric attack. The ability of a DNS operator to suppress such responses on a particular server makes that server a less useful amplifier.

The optional behaviour described in this document to reduce the size of responses to queries with QTYPE=ANY is compatible with the use of DNSSEC by both initiator and responder.

10. IANA Considerations

The IANA is requested to update the Resource Record (RR) TYPEs Registry [1] entry as follows:

Type	Value	Meaning	Reference
*	255	A request for some or all records the server has available	[RFC1035][RFC6895] [This Document]

11. Acknowledgements

David Lawrence provided valuable observations and concrete suggestions. Jeremy Laidman helped make the document better. Tony Finch realized that this document was valuable and implemented it

while under attack. Richard Gibson identified areas where more detail and accuracy was useful. A large number of other people also provided comments and suggestions we thank them all for the feedback.

12. References

12.1. 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>>.
- [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>>.
- [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>>.

12.2. Informative References

- [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>>.
- [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>>.
- [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>>.

12.3. URIs

- [1] <http://www.iana.org/assignments/dns-parameters/dns-parameters.xhtml#dns-parameters-4>

Appendix A. Editorial Notes

This section (and sub-sections) to be removed prior to publication.

A.1. Change History

A.1.1. draft-ietf-dnsop-refuse-any-07

Address AD's concerns: more colour to describe updates to 1034/1035 in the abstract; don't rely upon HINFO RDATA formatting; language cleanup around guess intent. Add Evan as author (originator of the "choose one record" response idea).

A.1.2. draft-ietf-dnsop-refuse-any-06

Update RFC 1034 as well as RFC 1035; define the term "conventional ANY response"; soften and qualify ANY does not mean ALL; note that the subset mode response lacks signalling.

A.1.3. draft-ietf-dnsop-refuse-any-05

Minor editorial changes. Soften advice on RRSIG queries. Version bump.

A.1.4. draft-ietf-dnsop-refuse-any-04

These are the changes requested during WGLC. The title has been updated for readability The behavior section now contains description of three different approaches in order of preference. Text added on behavior over TCP. The document is clear in how it updates from RFC1035. Minor adjustments for readability and remove redundancy.

A.1.5. draft-ietf-dnsop-refuse-any-03

Change section name to "Updates to RFC1034", few minor grammar changes suggested by Matthew Pounsett and Tony Finch.

Text clarifications, reflecting experience, added implementation experience.

A.1.6. draft-ietf-dnsop-refuse-any-02

Added suggestion to call out RRSIG is optional when DO=0.

Number of text suggestions from Jeremy Laidman.

A.1.7. draft-ietf-dnsop-refuse-any-01

Add IANA Considerations

A.1.8. draft-ietf-dnsop-refuse-any-00

Re-submitted with a different name following adoption at the dnsop WG meeting convened at IETF 94.

A.1.9. draft-jabley-dnsop-refuse-any-01

Make signing of RRsets in answers from signed zones mandatory.

Document the option of returning an existing RRSet in place of a synthesised one.

A.1.10. draft-jabley-dnsop-refuse-any-00

Initial draft circulated for comment.

Authors' Addresses

Joe Abley
Afilias
300-184 York Street
London, ON N6A 1B5
Canada

Phone: +1 519 670 9327
Email: jabley@afilias.info

Olafur Gudmundsson
Cloudflare Inc.

Email: olafur+ietf@cloudflare.com

Marek Majkowski
Cloudflare Inc.

Email: marek@cloudflare.com

Evan Hunt
ISC
950 Charter St
Redwood City, CA 94063
USA

Email: each@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.

5.1.1.1. 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

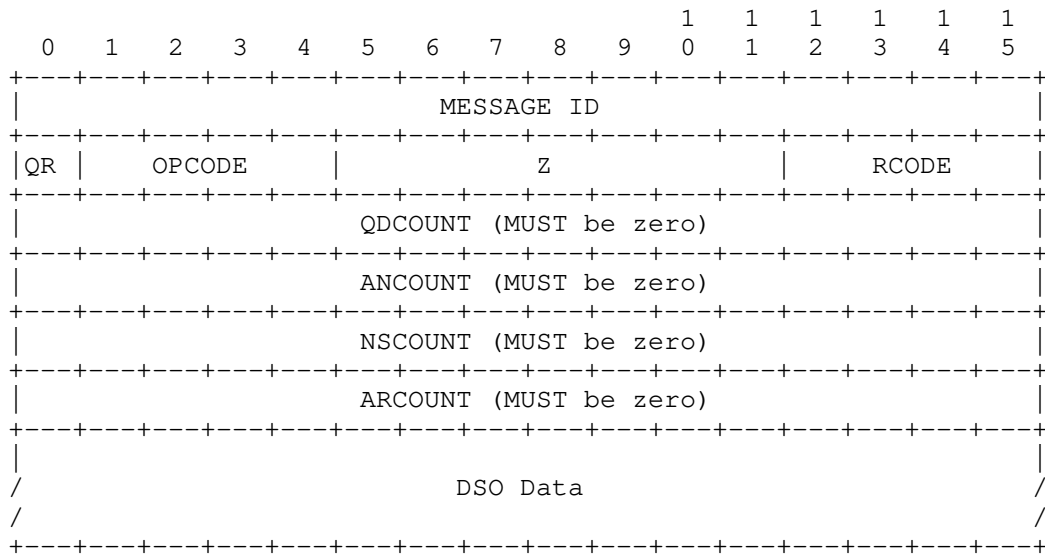
In protocol implementation there are generally two kinds of errors that software writers have to deal with. The first is situations that arise due to factors in the environment, such as temporary loss of connectivity. While undesirable, these situations do not indicate a flaw in the software, and they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

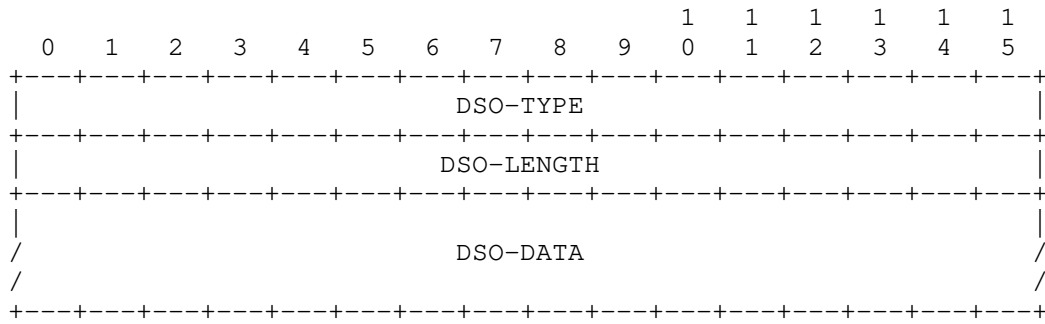
For example, after a client has subscribed for Push Notifications [I-D.ietf-dnssd-push], the subsequent event notifications are then sent as DSO unidirectional messages, and this is appropriate because the client initiated the message stream by virtue of its Push Notification subscription, thereby indicating 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 0xFFFFFFFFE ($2^{32}-2$ 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

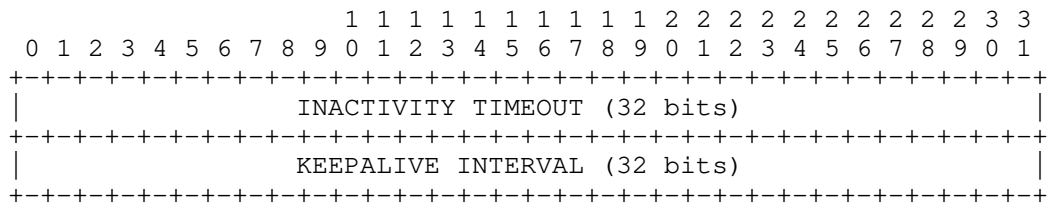
This section describes the three base TLVs for DNS Stateful Operations: Keepalive, Retry Delay, and Encryption Padding.

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:



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.

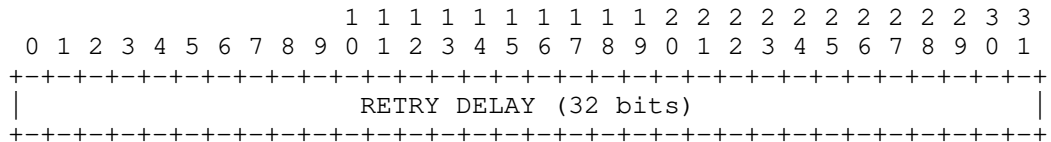
7.1.2. Relationship to edns-tcp-keepalive EDNS0 Option

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:



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

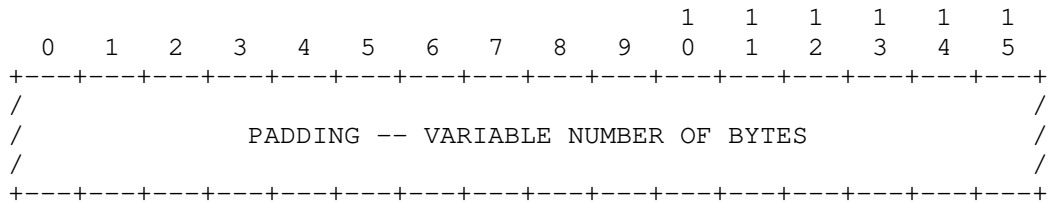
In the case of a DSO request message that results in a nonzero RCODE value, the responder MAY append a Retry Delay TLV to the response, indicating the time interval during which the initiator SHOULD NOT attempt this operation again.

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).



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 send 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 encloser: "The longest ancestor of a name that can be proven to exist. Note that this is only different from the closest encloser 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 encloser 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 encloser, provided that this wildcard domain name exists. 'Immediately descending' means that the source of synthesis has a name of the form: <asterisk label>.<closest encloser>." (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

Network Working Group
Internet-Draft
Intended status: Best Current Practice
Expires: September 20, 2018

J. Abley
Afilias
D. Knight
Neustar
March 19, 2018

Establishing an Appropriate Root Zone DNSSEC Trust Anchor at Startup
draft-jabley-dnsop-bootstrap-validator-00

Abstract

Domain Name System Security Extensions (DNSSEC) allow cryptographic signatures to be used to validate responses received from the Domain Name System (DNS). A DNS client which validates such signatures is known as a validator.

The choice of appropriate root zone trust anchor for a validator is expected to vary over time as the corresponding cryptographic keys used in DNSSEC are changed.

This document provides guidance on how validators might determine an appropriate trust anchor for the root zone to use at start-up, or when other mechanisms intended to allow key rollover to be tolerated gracefully are not available.

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, 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 (<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. Definitions 2
- 2. Introduction 3
- 3. Summary of Approach 4
 - 3.1. Initial State 4
 - 3.2. Trust Anchor Retrieval 4
 - 3.3. Trust Anchor Selection 4
 - 3.4. Full Operation 4
- 4. Timing Considerations 4
- 5. Retrieval of Candidate Trust Anchors 5
 - 5.1. Retrieval of Trust Anchors from Local Sources 5
 - 5.2. Retrieval of Trust Anchors from the DNS 5
 - 5.3. Retrieval of Trust Anchors from the Root Zone
KSK Manager 5
- 6. Establishing Trust in Candidate Trust Anchors 6
- 7. Failure to Locate a Valid Trust Anchor 7
- 8. IANA Considerations 7
- 9. Security Considerations 7
- 10. References 7
 - 10.1. Normative References 7
 - 10.2. Informative References 8
- Appendix A. Acknowledgements 8
- Appendix B. Editorial Notes 8
 - B.1. Discussion 8
 - B.2. Change History 8
- Authors' Addresses 9

1. Definitions

The terms Key Signing Key (KSK) and Trust Anchor are used as defined in [RFC4033].

The term Validator is used in this document to mean a Validating Security-Aware Stub Resolver, as defined in [RFC4033].

2. Introduction

The Domain Name System (DNS) is described in [RFC1034] and [RFC1035]. DNS Security Extensions (DNSSEC) are described in [RFC4033], [RFC4034] and [RFC4035].

The root zone of the DNS was signed using DNSSEC in July 2011, and many top-level domain registries have since signed their zones, installing secure delegations for them in the root zone. A single trust anchor for the root zone is hence increasingly sufficient for validators.

Validators are deployed in a variety of environments, and there is variation in the amount of system administration that might reasonably be expected to be available. For example, embedded devices might never be administered by a human operator, whereas validators deployed on general-purpose operating systems in enterprise networks might have technical staff available to assist with their configuration.

This document includes descriptions of mechanisms for validator bootstrapping, intended to be sufficient for embedded devices. The implementation of those mechanisms might be automatic in the case of unattended devices, or manual, carried out by a systems administrator, depending on local circumstances.

The choice of appropriate trust anchor for a DNSSEC Validator is expected to vary over time as the corresponding KSK used in the root zone is changed. The DNSSEC Policy and Practice Statement (DPS) for the root zone KSK maintainer [KSK-DPS] specifies that scheduled KSK rollover will be undertaken according to the semantics specified in [RFC5011]. Validators which are able to recognise and accommodate those semantics should need no additional support to be able to maintain an appropriate trust anchor over a root zone KSK rollover event.

The possibility remains, however, that [RFC5011] signalling will not be available to a validator: e.g. certain classes of emergency KSK rollover may require a compromised KSK to be discarded more quickly than [RFC5011] specifies, or a validator might be off-line over the whole key-roll event.

This document provides guidance on how DNSSEC Validators might determine an appropriate set of trust anchors to use at start-up, or when other mechanisms intended to allow key rollover to be tolerated gracefully are not available.

The bootstrapping procedures described in this document are also expected to be useful for a deployed, running validator which is not able to accommodate a KSK roll using [RFC5011] signalling.

3. Summary of Approach

A validator that has no valid trust anchor initialises itself as follows.

3.1. Initial State

A validator in its initial state is capable of sending and receiving DNS queries and responses, but is not capable of validating signatures received in responses.

A validator must confirm that its local clock is sufficiently accurate before trust anchors can be established, and before processing of DNSSEC signatures can proceed. Discussion of timing considerations can be found in Section 4.

3.2. Trust Anchor Retrieval

Once the local clock has been synchronised, a validator may proceed to gather candidate trust anchors for consideration. Discussion of trust anchor retrieval can be found in Section 5.

3.3. Trust Anchor Selection

Once a set of candidate trust anchors has been obtained, a validator attempts to find one trust anchor in the set which is appropriate for use. This process involves verification of cryptographic signatures, and is discussed in Section 6.

3.4. Full Operation

The validator now has an accurate trust anchor for the root zone, and is capable of validating signatures on responses from the DNS.

4. Timing Considerations

DNSSEC signatures are valid for particular periods of time, as specified by the administrator of the zone containing the signatures. It follows that any validator must maintain an accurate local clock in order to verify that signatures are accurate.

Trust anchors correspond to KSKs in particular zones. Zone administrators may choose to replace KSKs from time to time, e.g. due to a key compromise or local key management policy, and the

corresponding appropriate choice in trust anchor will change as KSKs are replaced.

Trust anchors for the root zone in particular are published with intended validity periods, as discussed in Section 5. A validator making use of such trust anchors also requires an accurate local clock in order to avoid configuring a local trust anchor which corresponds to an old key.

Validators should take appropriate steps to ensure that their local clocks are set with sufficient accuracy, and in the case where local clocks are set with reference to external time sources over a network [RFC5905] that the time information received from those sources is authentic.

5. Retrieval of Candidate Trust Anchors

Candidate trust anchors may be retrieved using several mechanisms. The process of gaining trust in particular candidate trust anchors before using them is discussed in Section 6.

5.1. Retrieval of Trust Anchors from Local Sources

A trust anchor which is packaged with validator software can never be trusted, since the corresponding root zone KSK may have rolled since the software was packaged, and the trust anchor may be derived from a root zone KSK that was retired due to compromise.

Validators should never use local trust anchors for bootstrapping.

5.2. Retrieval of Trust Anchors from the DNS

The current root zone trust anchor is a hash (in DS RDATA format) of a member of the root zone apex DNSKEY RRSet that has the SEP bit set. Such a trust anchor could be derived from a response to the query ". IN DNSKEY?", but there is no mechanism available to trust the result: without an existing, accurate trust anchor the validator has no means to gauge the authenticity of the response.

Validators should never derive trust anchors from DNSKEY RRsets obtained from the DNS.

5.3. Retrieval of Trust Anchors from the Root Zone KSK Manager

The Root Zone KSK Manager publishes trust anchors corresponding to the root zone KSK as described in [RFC7958].

A full history of previously-published trust anchors, including the trust anchor recommended for immediate use, is made available in an XML document at the following stable URLs:

- o <<http://data.iana.org/root-anchors/root-anchors.xml>>
- o <<https://data.iana.org/root-anchors/root-anchors.xml>>

Validity periods for each trust anchor packaged in the root-anchors.xml document are provided as XML attributes, allowing an appropriate trust anchor for immediate use to be identified (but see Section 4).

Individual trust anchors are also packaged as X.509 identity certificates, signed by various Certificate Authorities (CAs). URLs to allow those certificates to be retrieved are included as optional elements in the XML document.

For automatic bootstrapping, the recommended approach is as follows.

1. Retrieve <<http://data.iana.org/root-anchors/root-anchors.xml>>
 2. Identify the trust anchors which are valid for current use, with reference to the current time and date.
 3. Retrieve the corresponding X.509 identity certificates for the key identified in the previous step, for use in establishing trust in the retrieved trust anchor (see Section 6).
6. Establishing Trust in Candidate Trust Anchors

Once a candidate trust anchor has been retrieved, the validator must establish that it is authentic before it can be used. This document recommends that this be carried out by checking the signatures on each of the X.509 identity certificates retrieved in the previous step until a certificate is found which matches a CA trust anchor.

This verification phase requires that validators ship with a useful set of CA trust anchors, and that corresponding identity certificates are published by the root zone KSK manager. In some cases validator implementors may decide to use commercial CA services, perhaps a subset of the "browser list" that is commonly distributed with web browsers; alternatively a vendor may instantiate its own CA and make arrangements with the root zone KSK manager to have the corresponding identity certificate locations published in root-anchors.xml.

The CA trust anchors packaged with validators should have an expected lifetime in excess of the anticipated life of the validator. As a

protection against CA failure, validators are recommended to ship with more than one CA trust anchor.

7. Failure to Locate a Valid Trust Anchor

A validator that has failed to locate a valid trust anchor may re-try the retrieval and trust establishment phases indefinitely, but must not perform validation on DNS responses until a valid trust anchor has been identified.

8. IANA Considerations

This document has no IANA actions.

9. Security Considerations

This document discusses an approach for automatic configuration of trust anchors in a DNSSEC validator.

10. References

10.1. Normative References

- [KSK-DPS] Ljunggren, F., Okubo, T., Lamb, R., and J. Schlyter, "DNSSEC Practice Statement for the Root Zone KSK Operator", May 2010.
- [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>>.
- [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>>.
- [RFC5011] StJohns, M., "Automated Updates of DNS Security (DNSSEC) Trust Anchors", STD 74, RFC 5011, DOI 10.17487/RFC5011, September 2007, <<https://www.rfc-editor.org/info/rfc5011>>.
- [RFC7958] Abley, J., Schlyter, J., Bailey, G., and P. Hoffman, "DNSSEC Trust Anchor Publication for the Root Zone", RFC 7958, DOI 10.17487/RFC7958, August 2016, <<https://www.rfc-editor.org/info/rfc7958>>.

10.2. Informative References

- [RFC5905] Mills, D., Martin, J., Ed., Burbank, J., and W. Kasch, "Network Time Protocol Version 4: Protocol and Algorithms Specification", RFC 5905, DOI 10.17487/RFC5905, June 2010, <<https://www.rfc-editor.org/info/rfc5905>>.

Appendix A. Acknowledgements

This document contains material first discussed at VeriSign and ICANN during the deployment of DNSSEC in the root zone, and also draws upon subsequent technical discussion from public mailing lists. The contributions of all those who voiced opinions are acknowledged.

Appendix B. Editorial Notes

This section (and sub-sections) to be removed prior to publication.

B.1. Discussion

This is not a working group document. However, the topics discussed in this document are consistent with the general subject area of the DNSOP working group, and discussion of this document could reasonably take place on the dnsop mailing list.

B.2. Change History

draft-jabley-dnsop-validator-bootstrap-00 Initial draft.

draft-jabley-dnsop-bootstrap-validator-00 Resurrected in response to observed root zone KSK rollover hilarity. References updated, slightly. Authors' contact information updated. Document name changed to be arbitrarily different from previous, expired draft in order to avoid zombie jokes.

Authors' Addresses

Joe Abley
Afilias
300-184 York Street
London, ON N6A 1B5
Canada

Phone: +1 519 670 9327
Email: jabley@afilias.info

Dave Knight
Neustar
300-184 York Street
London, ON N6A 1B5
Canada

Email: dave.knight@team.neustar

Network Working Group
Internet-Draft
Updates: 7706 (if approved)
Intended status: Informational
Expires: December 26, 2018

W. Kumari
Google
P. Hoffman
ICANN
June 24, 2018

Decreasing Access Time to Root Servers by Running One On The Same Server
draft-kh-dnsop-7706bis-01

Abstract

Some DNS recursive resolvers have longer-than-desired round-trip times to the closest DNS root server. Some DNS recursive resolver operators want to prevent snooping of requests sent to DNS root servers by third parties. Such resolvers can greatly decrease the round-trip time and prevent observation of requests by running a copy of the full root zone on the same server, such as on a loopback address. This document shows how to start and maintain such a copy of the root zone that does not pose a threat to other users of the DNS, at the cost of adding some operational fragility for the operator.

This draft will update RFC 7706. See Section 1.1 for a list of topics that will be added in the update.

[Ed note: Text inside square brackets ([]) is additional background information, answers to frequently asked questions, general musings, etc. They will be removed before publication.]

[This document is being collaborated on in Github at: <https://github.com/wkumari/draft-kh-dnsop-7706bis>. The most recent version of the document, open issues, and so on should all be available there. The authors gratefully accept pull requests.]

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 December 26, 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 (<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
1.1. Updates from RFC 7706	4
1.2. Requirements Notation	5
2. Requirements	5
3. Operation of the Root Zone on the Local Server	5
4. Using the Root Zone Server on the Same Host	7
5. Security Considerations	7
6. References	7
6.1. Normative References	7
6.2. Informative References	8
Appendix A. Current Sources of the Root Zone	8
Appendix B. Example Configurations of Common Implementations	9
B.1. Example Configuration: BIND 9.9	9
B.2. Example Configuration: Unbound 1.4 and NSD 4	10
B.3. Example Configuration: Microsoft Windows Server 2012	11
Acknowledgements	12
Authors' Addresses	12

1. Introduction

DNS recursive resolvers have to provide answers to all queries from their customers, even those for domain names that do not exist. For each queried name that has a top-level domain (TLD) that is not in the recursive resolver's cache, the resolver must send a query to a root server to get the information for that TLD, or to find out that the TLD does not exist. Research shows that the vast majority of

queries going to the root are for names that do not exist in the root zone, partially because the negative answers are cached for a much shorter period of time. A slow path between the recursive resolver and the closest root server has a negative effect on the resolver's customers.

Many of the queries from recursive resolvers to root servers get answers that are referrals to other servers. Malicious third parties might be able to observe that traffic on the network between the recursive resolver and root servers.

This document describes a method for the operator of a recursive resolver to greatly speed these queries and to hide them from outsiders. The basic idea is to create an up-to-date root zone server on the same host as the recursive server, and use that server when the recursive resolver looks up root information. The recursive resolver validates all responses from the root server on the same host, just as it would all responses from a remote root server.

The primary goals of this design are to provide faster negative responses to stub resolver queries that contain queries that result in NXDOMAIN responses, and to prevent queries and responses from being visible on the network. This design will probably have little effect on getting faster positive responses to stub resolver for good queries on TLDs, because the TTL for most TLDs is usually long-lived (on the order of a day or two) and is thus usually already in the cache of the recursive resolver.

This design explicitly only allows the new root zone server to be run on the same server as the recursive resolver, in order to prevent the server from serving authoritative answers to any other system. Specifically, the root server on the local system **MUST** be configured to only answer queries from the resolvers on the same host, and **MUST NOT** answer queries from any other resolver.

It is important to note that the design described in this document is controversial. There is not consensus on whether this is a "best practice". In fact, many people feel that it is an excessively risky practice because it introduces a new operational piece to local DNS operations where there was not one before. The advantages listed above do not come free: if this new system does not work correctly, users can get bad data, or the entire recursive resolution system might fail in ways that are hard to diagnose.

This design requires the addition of authoritative name server software running on the same machine as the recursive resolver. Thus, recursive resolver software such as BIND will not need to add much new functionality, but recursive resolver software such as

Unbound will need to be able to talk to an authoritative server (such as NSD) running on the same host. However, more recursive resolver software might add the capabilities described in this document in the future.

A different approach to solving the problems discussed in this document is described in [RFC8198].

1.1. Updates from RFC 7706

RFC 7706 explicitly required that the root server instance be run on the loopback interface of the host running the validating resolver. However, RFC 7706 also had examples of how to set up common software that did not use the loopback interface. Thus, this document loosens the restriction on the interface but keeps the requirement that only systems running on that single host be able to query that root server instance.

Removed the prohibition on distribution of recursive DNS servers including configurations for this design because some already do, and others have expressed an interest in doing so.

Added the idea that a recursive resolver using this design might switch to using the normal (remote) root servers if the local root server fails.

[This section will list all the changes from RFC 7706. For this draft, it is also the list of changes that we will make in future versions of the draft.]

[Give a clearer comparison of software that allows slaving the root zone in the software (such as BIND) versus resolver software that requires a local slaved root zone (Unbound).]

[Add examples of other resolvers such as Knot Resolver and PowerDNS Recursor, and maybe Windows Server.]

[Add discussion of BIND slaving the root zone in the same view instead of using different views.]

[Make the use cases explicit. Be clearer that a real use case is folks who are worried that root server unavailability due to DDoS against them is a reason some people would use the mechanisms here.]

[Describe how slaving the root zone from root zone servers does not fully remove the reliance on the root servers being available.]

[Refresh list of where one can get copies of the root zone.]

[Other new topics might go here.]

1.2. Requirements Notation

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. Requirements

In order to implement the mechanism described in this document:

- o The system MUST be able to validate a zone with DNSSEC [RFC4033].
- o The system MUST have an up-to-date copy of the key used to sign the DNS root.
- o The system MUST be able to retrieve a copy of the entire root zone (including all DNSSEC-related records).
- o The system MUST be able to run an authoritative server for the root zone on the same host. The root server instance MUST only respond to queries from the same host. One way to assure not responding to queries from other hosts is to make the address of the authoritative server one of the IPv4 loopback addresses (that is, an address in the range 127/8 for IPv4 or ::1 in IPv6).

A corollary of the above list is that authoritative data in the root zone used on the local authoritative server MUST be identical to the same data in the root zone for the DNS. It is possible to change the unsigned data (the glue records) in the copy of the root zone, but such changes could cause problems for the recursive server that accesses the local root zone, and therefore any changes to the glue records SHOULD NOT be made.

3. Operation of the Root Zone on the Local Server

The operation of an authoritative server for the root in the system described here can be done separately from the operation of the recursive resolver, or it might be part of the configuration of the recursive resolver system.

The steps to set up the root zone are:

1. Retrieve a copy of the root zone. (See Appendix A for some current locations of sources.)

2. Start the authoritative server with the root zone on an address on the host that is not in use. For IPv4, this could be 127.0.0.1, but if that address is in use, any address in 127/8 is acceptable. For IPv6, this would be ::1. It can also be a publicly-visible address on the host, but only if the authoritative server software allows restricting the addresses that can access the authoritative server, and the software is configured to only allow access from addresses on this single host.

The contents of the root zone MUST be refreshed using the timers from the SOA record in the root zone, as described in [RFC1035]. This inherently means that the contents of the local root zone will likely be a little behind those of the global root servers because those servers are updated when triggered by NOTIFY messages.

If the contents of the root zone cannot be refreshed before the expire time in the SOA, the local root server MUST return a SERVFAIL error response for all queries sent to it until the zone can be successfully be set up again. Because this would cause a recursive resolver on the same host that is relying on this root server to also fail, a resolver might be configured to immediately switch to using other (non-local) root servers if the resolver receives a SERVFAIL response from a local root server.

In the event that refreshing the contents of the root zone fails, the results can be disastrous. For example, sometimes all the NS records for a TLD are changed in a short period of time (such as 2 days); if the refreshing of the local root zone is broken during that time, the recursive resolver will have bad data for the entire TLD zone.

An administrator using the procedure in this document SHOULD have an automated method to check that the contents of the local root zone are being refreshed; this might be part of the resolver software. One way to do this is to have a separate process that periodically checks the SOA of the root zone from the local root zone and makes sure that it is changing. At the time that this document is published, the SOA for the root zone is the digital representation of the current date with a two-digit counter appended, and the SOA is changed every day even if the contents of the root zone are unchanged. For example, the SOA of the root zone on January 2, 2018 was 2018010201. A process can use this fact to create a check for the contents of the local root zone (using a program not specified in this document).

4. Using the Root Zone Server on the Same Host

A recursive resolver that wants to use a root zone server operating as described in Section 3 simply specifies the local address as the place to look when it is looking for information from the root. All responses from the root server MUST be validated using DNSSEC.

Note that using this simplistic configuration will cause the recursive resolver to fail if the local root zone server fails. A more robust configuration would cause the resolver to start using the normal remote root servers when the local root server fails (such as if it does not respond or gives SERVFAIL responses).

See Appendix B for more discussion of this for specific software.

To test the proper operation of the recursive resolver with the local root server, use a DNS client to send a query for the SOA of the root to the recursive server. Make sure the response that comes back has the AA bit in the message header set to 0.

5. Security Considerations

A system that does not follow the DNSSEC-related requirements given in Section 2 can be fooled into giving bad responses in the same way as any recursive resolver that does not do DNSSEC validation on responses from a remote root server. Anyone deploying the method described in this document should be familiar with the operational benefits and costs of deploying DNSSEC [RFC4033].

As stated in Section 1, this design explicitly only allows the new root zone server to be run on the same host, answering queries only from resolvers on that host, in order to prevent the server from serving authoritative answers to any system other than the recursive resolver. This has the security property of limiting damage to any other system that might try to rely on an altered copy of the root.

6. References

6.1. Normative References

- [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>>.

[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>>.

6.2. Informative References

[Manning2013] Manning, W., "Client Based Naming", 2013, <http://www.sfc.wide.ad.jp/dissertation/bill_e.html>.

[RFC8198] Fujiwara, K., Kato, A., and W. Kumari, "Aggressive Use of DNSSEC-Validated Cache", RFC 8198, DOI 10.17487/RFC8198, July 2017, <<https://www.rfc-editor.org/info/rfc8198>>.

Appendix A. Current Sources of the Root Zone

The root zone can be retrieved from anywhere as long as it comes with all the DNSSEC records needed for validation. Currently, one can get the root zone from ICANN by zone transfer (AXFR) over TCP from DNS servers at xfr.lax.dns.icann.org and xfr.cjr.dns.icann.org.

Currently, the root can also be retrieved by AXFR over TCP from the following root server operators:

- o b.root-servers.net
- o c.root-servers.net
- o f.root-servers.net
- o g.root-servers.net
- o k.root-servers.net

It is crucial to note that none of the above services are guaranteed to be available. It is possible that ICANN or some of the root server operators will turn off the AXFR capability on the servers listed above. Using AXFR over TCP to addresses that are likely to be anycast (as the ones above are) may conceivably have transfer problems due to anycast, but current practice shows that to be unlikely.

To repeat the requirement from earlier in this document: if the contents of the zone cannot be refreshed before the expire time, the server MUST return a SERVFAIL error response for all queries until the zone can be successfully be set up again.

Appendix B. Example Configurations of Common Implementations

This section shows fragments of configurations for some popular recursive server software that is believed to correctly implement the requirements given in this document.

The IPv4 and IPv6 addresses in this section were checked recently by testing for AXFR over TCP from each address for the known single-letter names in the root-servers.net zone.

The examples here use a loopback address of 127.12.12.12, but typical installations will use 127.0.0.1. The different address is used in order to emphasize that the root server does not need to be on the device at the name "localhost" which is often locally served as 127.0.0.1.

B.1. Example Configuration: BIND 9.9

BIND acts both as a recursive resolver and an authoritative server. Because of this, there is "fate-sharing" between the two servers in the following configuration. That is, if the root server dies, it is likely that all of BIND is dead.

Using this configuration, queries for information in the root zone are returned with the AA bit not set.

When slaving a zone, BIND will treat zone data differently if the zone is slaved into a separate view (or a separate instance of the software) versus slaved into the same view or instance that is also performing the recursion.

Validation: When using separate views or separate instances, the DS records in the slaved zone will be validated as the zone data is accessed by the recursive server. When using the same view, this validation does not occur for the slaved zone.

Caching: When using separate views or instances, the recursive server will cache all of the queries for the slaved zone, just as it would using the traditional "root hints" method. Thus, as the zone in the other view or instance is refreshed or updated, changed information will not appear in the recursive server until the TTL of the old record times out. Currently, the TTL for DS and delegation NS records is two days. When using the same view, all zone data in the recursive server will be updated as soon as it receives its copy of the zone.

```
view root {
    match-destinations { 127.12.12.12; };
    zone "." {
        type slave;
        file "rootzone.db";
        notify no;
        masters {
            192.228.79.201; # b.root-servers.net
            192.33.4.12;    # c.root-servers.net
            192.5.5.241;   # f.root-servers.net
            192.112.36.4;  # g.root-servers.net
            193.0.14.129;  # k.root-servers.net
            192.0.47.132;  # xfr.cjr.dns.icann.org
            192.0.32.132;  # xfr.lax.dns.icann.org
            2001:500:84::b; # b.root-servers.net
            2001:500:2f::f; # f.root-servers.net
            2001:7fd::1;   # k.root-servers.net
            2620:0:2830:202::132; # xfr.cjr.dns.icann.org
            2620:0:2d0:202::132; # xfr.lax.dns.icann.org
        };
    };
};

view recursive {
    dnssec-validation auto;
    allow-recursion { any; };
    recursion yes;
    zone "." {
        type static-stub;
        server-addresses { 127.12.12.12; };
    };
};
```

B.2. Example Configuration: Unbound 1.4 and NSD 4

Unbound and NSD are separate software packages. Because of this, there is no "fate-sharing" between the two servers in the following configurations. That is, if the root server instance (NSD) dies, the recursive resolver instance (Unbound) will probably keep running but will not be able to resolve any queries for the root zone. Therefore, the administrator of this configuration might want to carefully monitor the NSD instance and restart it immediately if it dies.

Using this configuration, queries for information in the root zone are returned with the AA bit not set.

```
# Configuration for Unbound
server:
  do-not-query-localhost: no
stub-zone:
  name: "."
  stub-prime: no
  stub-addr: 127.12.12.12

# Configuration for NSD
server:
  ip-address: 127.12.12.12
zone:
  name: "."
  request-xfr: 192.228.79.201 NOKEY # b.root-servers.net
  request-xfr: 192.33.4.12 NOKEY # c.root-servers.net
  request-xfr: 192.5.5.241 NOKEY # f.root-servers.net
  request-xfr: 192.112.36.4 NOKEY # g.root-servers.net
  request-xfr: 193.0.14.129 NOKEY # k.root-servers.net
  request-xfr: 192.0.47.132 NOKEY # xfr.cjr.dns.icann.org
  request-xfr: 192.0.32.132 NOKEY # xfr.lax.dns.icann.org
  request-xfr: 2001:500:84::b NOKEY # b.root-servers.net
  request-xfr: 2001:500:2f::f NOKEY # f.root-servers.net
  request-xfr: 2001:7fd::1 NOKEY # k.root-servers.net
  request-xfr: 2620:0:2830:202::132 NOKEY # xfr.cjr.dns.icann.org
  request-xfr: 2620:0:2d0:202::132 NOKEY # xfr.lax.dns.icann.org
```

B.3. Example Configuration: Microsoft Windows Server 2012

Windows Server 2012 contains a DNS server in the "DNS Manager" component. When activated, that component acts as a recursive server. DNS Manager can also act as an authoritative server.

Using this configuration, queries for information in the root zone are returned with the AA bit set.

The steps to configure DNS Manager to implement the requirements in this document are:

1. Launch the DNS Manager GUI. This can be done from the command line ("dnsmgmt.msc") or from the Service Manager (the "DNS" command in the "Tools" menu).
2. In the hierarchy under the server on which the service is running, right-click on the "Forward Lookup Zones", and select "New Zone". This brings up a succession of dialog boxes.
3. In the "Zone Type" dialog box, select "Secondary zone".

4. In the "Zone Name" dialog box, enter ".".
5. In the "Master DNS Servers" dialog box, enter "b.root-servers.net". The system validates that it can do a zone transfer from that server. (After this configuration is completed, the DNS Manager will attempt to transfer from all of the root zone servers.)
6. In the "Completing the New Zone Wizard" dialog box, click "Finish".
7. Verify that the DNS Manager is acting as a recursive resolver. Right-click on the server name in the hierarchy, choosing the "Advanced" tab in the dialog box. See that "Disable recursion (also disables forwarders)" is not selected, and that "Enable DNSSEC validation for remote responses" is selected.

Acknowledgements

The authors fully acknowledge that running a copy of the root zone on the loopback address is not a new concept, and that we have chatted with many people about that idea over time. For example, Bill Manning described a similar solution but to a very different problem (intermittent connectivity, instead of constant but slow connectivity) in his doctoral dissertation in 2013 [Manning2013].

Evan Hunt contributed greatly to the logic in the requirements. Other significant contributors include Wouter Wijngaards, Tony Hain, Doug Barton, Greg Lindsay, and Akira Kato. The authors also received many offline comments about making the document clear that this is just a description of a way to operate a root zone on the same host, and not a recommendation to do so.

Authors' Addresses

Warren Kumari
Google

Email: Warren@kumari.net

Paul Hoffman
ICANN

Email: paul.hoffman@icann.org

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 29, 2018

W. Mekking
Oracle Dyn
O. Gudmundsson
CloudFlare
March 28, 2018

Minimal Incremental Zone Transfer in DNS
draft-mekking-mixfr-02

Abstract

This document proposes extensions to the DNS protocol to provide an incremental zone transfer (IXFR) mechanism with dynamic update (UPDATE) capabilities, to keep IXFRs that deal with DNSSEC small.

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 29, 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.	Definitions	3
3.	Syntax	3
3.1.	Implicit RRSIG deletion	3
3.2.	Add an RR	3
3.3.	Delete an RR	3
3.4.	Delete an RRset	3
3.5.	Delete All RRsets on a Name	4
3.6.	Replace an RRset	4
4.	Protocol Description	4
4.1.	Client side	4
4.2.	Server side	5
4.3.	Future zone transfer improvements	5
5.	IANA Considerations	6
6.	Security Considerations	6
7.	Acknowledgements	6
8.	References	6
8.1.	Informative References	6
8.2.	Normative References	7
Appendix A.	Changelog	8
A.1.	Version 02	8
A.2.	Version 01	8
A.3.	Version 00	8
Authors' Addresses		8

1. Introduction

Incremental zone transfer (IXFR, [RFC1995]) was introduced to efficiently transfer changed portions of a zone. However, when a zone is signed with DNSSEC [RFC4033], [RFC4034], [RFC4035], the transfer can still become very large. For example, when many resource record sets (RRsets) need to be re-signed, or when the NSEC3 [RFC5155] salt is changed, an IXFR may become larger than a full zone transfer (AXFR, [RFC5936]). This is because the IXFR includes complete copies of both the deleted and replacement RRSIG records.

To keep the deltas small in zone transfers, we need to have a richer change syntax, for example like in Dynamic Update (DNS UPDATE, [RFC2136]). This document introduces a new query type MIXFR (minimal incremental zone transfer) that is able to express this richer syntax. The goal of this proposal is to allow small changes to be communicated over UDP, and remove as much redundant information from the zone transfer as possible.

An earlier proposal to keep the zone transfers small is IXFR-ONLY [IXFR-ONLY], by giving the client an opportunity to signal the server

that it prefers an error above a fall back to an AXFR in case the server is not able to send an IXFR. However IXFR-ONLY did not reduce the size of an IXFR.

2. Definitions

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. Syntax

The syntax for MIXFR is a superset of IXFR. The richer syntax of MIXFR allows to add or delete multiple records with one resource record (RR). MIXFR is DNSSEC aware thus if there is a change to RRset it knows to delete the covering RRSIG(s), this saves the transmission of old RRSigs.

3.1. Implicit RRSIG deletion

When an RRset is modified, the MIXFR client MUST also remove all existing RRSIG records on that RRset. This is valid for all RRtypes except RRSIG itself.

3.2. Add an RR

This works the same as with IXFR, with implicit RRSIG delete logic added.

3.3. Delete an RR

This works the same as with IXFR, with implicit RRSIG delete logic added.

3.4. Delete an RRset

Similar to DNS UPDATE. To delete an RRset, the MIXFR deletion list includes an RR whose NAME and TYPE are those of the RRset to be deleted. CLASS must be specified as ANY. RDLENGTH must be zero (0) and RDATA must therefore be empty. This also deletes the covering RRSIGs.

Note that a record with its CLASS set to ANY does not mean to delete (or change) the record in all available classes: zone transfers are encapsulated in SOA records that determine the zone name and class (see Figure ()[#fig:a-MIXFR-response]). Only changes in the zone matching that name and class will be made.

3.5. Delete All RRsets on a Name

Similar to DNS UPDATE. To delete all RRsets at a name, the MIXFR deletion list includes an RR at that NAME, whose TYPE must be specified as ANY and CLASS must be specified as ANY. RDLENGTH must be zero (0) and RDATA must therefore be empty.

3.6. Replace an RRset

The MIXFR addition list includes an RR whose NAME and TYPE are those of the RRset to be replaced. CLASS must be specified as ANY. RDLENGTH must be non-zero and the RDATA is that of the first replacement record.

If an RRset is to be replaced with multiple records, the second and subsequent records MUST use the syntax for adding an RR.

The same syntax is used to delete an RRset and to replace an RRset with an RR whose RDLENGTH is zero. This is not ambiguous because the former appears in the deletion list (before the new SOA RR) and the latter appears in the addition list (after the new SOA RR).

4. Protocol Description

4.1. Client side

The client can send a MIXFR request. Just like with IXFR, it places a SOA RR in the authority section to signal the version of the zone it holds now. If the client does not want the server to fall back to AXFR, it MAY add another SOA RR in the additional section. This achieves MIXFR-only behavior, similar to IXFR-ONLY [IXFR-ONLY]. For example:

```
;; ->>HEADER<<- opcode: QUERY, rcode: NOERROR, id: 1337
;; flags: qr ; QUERY: 1, ANSWER: 0, AUTHORITY: 1, ADDITIONAL: 1
;; QUESTION SECTION:
;; example.      IN      MIXFR

;; AUTHORITY SECTION:
example.        IN      SOA      serial=1

;; ADDITIONAL SECTION:
example.        IN      SOA      serial=1
```

Figure 1: A MIXFR request for the "example." zone.

[MM] Adding a whole record is quite some overhead in bits while we only signal one bit of information: to fall back or not to fall back.

[OG] Can we use a bit from header or OPT record? Or can we just use "Class | 0x8000" to signal that?

4.2. Server side

A server receiving a minimal incremental zone transfer (MIXFR) request will reply with a MIXFR. A MIXFR looks exactly like an IXFR, except there may be zero or more of the new introduced syntax RRs that can add or delete more records. For the zone "example.", the following zone transfer can be sent that will replace all signatures in the zone with new signatures for the names "example.", "a.example.", "b.example." and "c.example.":

```
;; ->>HEADER<<- opcode: QUERY, rcode: NOERROR, id: 1337
;; flags: qr ; QUERY: 1, ANSWER: 9, AUTHORITY: 0, ADDITIONAL: 0
;; QUESTION SECTION:
;; example.      IN      MIXFR

;; ANSWER SECTION:
example.        IN      SOA      serial=3
example.        IN      SOA      serial=1
example.        ANY    RRSIG
example.        IN      SOA      serial=3
example.        IN      RRSIG    rdata
a.example.      IN      RRSIG    rdata
b.example.      IN      RRSIG    rdata
c.example.      IN      RRSIG    rdata
example.        IN      SOA      serial=3
```

Figure 2: A MIXFR response for the "example." zone.

The server MAY reply with an IXFR or AXFR instead. If the server does not implement MIXFR it MUST return a response with NOTIMPL rcode. The client MUST fallback to request IXFR or AXFR.

4.3. Future zone transfer improvements

In many cases DNS servers have many zones in common, and there are many changes in the zones each hour, in this case having a long lived TCP connection or an out-of-band protocol where the primary server can push changes to the secondary.

The size of the zone transfer can be reduced even more if the syntax on the wire is changed, i.e. the RR wire format is abandoned. A different grammar may add operators, remove duplicate RRset owner names, and use standard compression algorithms.

These kind of improvements will require more drastic changes, and may be covered in a separate, future document.

5. IANA Considerations

IANA is requested to assign the OPCODE value [TBD] (decimal) for MIXFR, in sub-registry "DNS OpCodes" of registry "Domain Name System (DNS) Parameters".

6. Security Considerations

This document does not introduce additional security considerations. Or does it?

Should we explain what the security implications are, because descriptions from old RFC's are not good enough?

Any MIXFR transactions should use secure channels such as IPSEC or SSH tunnel, and use TSIG for authentication.

7. Acknowledgements

Johan Ihren, Tony Finch, Bob Harold.

8. References

8.1. Informative References

[IXFR-ONLY]

Sury, O. and S. Kerr, "IXFR-ONLY to Prevent IXFR Fallback to AXFR", February 2010, <<https://tools.ietf.org/html/draft-kerr-ixfr-only-01>>.

[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>>.

- [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>>.
- [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>>.

8.2. Normative References

- [RFC1995] Ohta, M., "Incremental Zone Transfer in DNS", RFC 1995, DOI 10.17487/RFC1995, August 1996, <<https://www.rfc-editor.org/info/rfc1995>>.
- [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>>.

Appendix A. Changelog

A.1. Version 02

- o Removed 'Delete All RRsets of a Type' because it had the same syntax as 'Delete an RRset' [Olafur].
- o Clarify ANY CLASS [#5, Bob Harold].
- o Sleep for 3 years.
- o Remove IXFR Gone Wild section.

A.2. Version 01

- o Split document in trivial and 'more wild' ideas.

A.3. Version 00

- o Initial version

Authors' Addresses

W. (Matthijs) Mekking
Oracle Dyn
Hertogswetering 163-167
Utrecht 3543 AS Utrecht
NL

EMail: matthijs.mekking@oracle.com
URI: <https://www.dyn.com>

Olafur Gudmundsson
CloudFlare
San Francisco, CA 94107
USA

EMail: olafur@cloudflare.com

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 Strettman 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
Intended status: Standards Track
Expires: November 30, 2018

P. Spacek
CZ.NIC
O. Gudmundsson
Cloudflare
O. Sury
ISC
May 29, 2018

Minimal EDNS compliance requirements
draft-spacek-edns-camel-diet-01

Abstract

DNS responders must either follow RFC 6891 by fully implementing EDNS or at least respond to queries containing OPT record according to older specifications. Non-compliant implementations which do not respond at all are not worth talking to.

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 November 30, 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 (<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
1.1. Terminology	2
2. The Protocol	2
3. Security Considerations	2
4. Privacy Considerations	2
5. IANA Considerations	3
6. Normative References	3
Authors' Addresses	3

1. Introduction

Neither the original DNS standard RFC 1035 nor its extensions RFC 2671 and RFC 6891 allow not to respond to a DNS query. Many years later non-compliant implementations which drop queries still exist and cause lot of extra queries, latency, and complicated logic in recursive resolvers. The cost of supporting these non-compliant implementations keeps increasing.

1.1. 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.

2. The Protocol

No DNS response message to a repeated DNS query containing EDNS extension implies that the other side is not a DNS responder. The querier MUST NOT retry its query without EDNS.

3. Security Considerations

Instruction to follow EDNS standard does not change security properties beyond what is written in RFC 6891.

4. Privacy Considerations

This has no effect on privacy of DNS.

5. IANA Considerations

[Note to IANA, to be removed prior to publication: there are no IANA considerations stated in this version of the document.]

6. Normative References

[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>>.

[RFC2671] Vixie, P., "Extension Mechanisms for DNS (EDNS0)", RFC 2671, DOI 10.17487/RFC2671, August 1999, <<https://www.rfc-editor.org/info/rfc2671>>.

[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

Petr Spacek

Email: petr.spacek@nic.cz

Olafur Gudmundsson

Email: olafur+ietf@cloudflare.com

Ondrej Sury

Email: ondrej@isc.org