

DNSOP Working Group
Internet-Draft
Updates: RFC [1035](#), [RFC 7766](#) (if
approved)
Intended status: Standards Track
Expires: September 6, 2018

R. Bellis
ISC
S. Cheshire
Apple Inc.
J. Dickinson
S. Dickinson
Sinodun
T. Lemon
Barefoot Consulting
T. Pusateri
Unaffiliated
March 05, 2018

DNS Stateful Operations
draft-ietf-dnsop-session-signal-06

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.

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 | 3 |
| 2. | Terminology | 5 |
| 3. | Discussion | 9 |
| 4. | Protocol Details | 10 |
| 4.1. | DSO Session Establishment | 10 |
| 4.1.1. | Connection Sharing | 12 |
| 4.1.2. | Zero Round-Trip Operation | 12 |
| 4.1.3. | Middlebox Considerations | 13 |
| 4.2. | Message Format | 14 |
| 4.2.1. | DNS Header Fields in DSO Messages | 15 |
| 4.2.2. | DSO Data | 17 |
| 4.2.3. | EDNS(0) and TSIG | 22 |
| 4.3. | Message Handling | 23 |
| 4.3.1. | Error Responses | 24 |
| 4.4. | DSO Response Generation | 25 |
| 4.5. | Responder-Initiated Operation Cancellation | 26 |
| 5. | DSO Session Lifecycle and Timers | 27 |
| 5.1. | DSO Session Initiation | 27 |
| 5.2. | DSO Session Timeouts | 27 |
| 5.3. | Inactive DSO Sessions | 28 |
| 5.4. | The Inactivity Timeout | 29 |
| 5.4.1. | Closing Inactive DSO Sessions | 29 |
| 5.4.2. | Values for the Inactivity Timeout | 30 |
| 5.5. | The Keepalive Interval | 31 |
| 5.5.1. | Keepalive Interval Expiry | 31 |
| 5.5.2. | Values for the Keepalive Interval | 31 |
| 5.6. | Server-Initiated Session Termination | 33 |
| 5.6.1. | Server-Initiated Retry Delay Message | 34 |
| 6. | Base TLVs for DNS Stateful Operations | 37 |
| 6.1. | Keepalive TLV | 37 |
| 6.1.1. | Client handling of received Session Timeout values | 39 |
| 6.1.2. | Relation to EDNS(0) TCP Keepalive Option | 41 |
| 6.2. | Retry Delay TLV | 42 |
| 6.2.1. | Retry Delay TLV used as a Primary TLV | 42 |
| 6.2.2. | Retry Delay TLV used as a Response Additional TLV | 43 |
| 6.3. | Encryption Padding TLV | 44 |

| | | |
|-----------------------|-----------------------------------|--------------------|
| 7. | Summary Highlights | 45 |
| 7.1. | QR bit and MESSAGE ID | 45 |
| 7.2. | TLV Usage | 46 |
| 8. | IANA Considerations | 48 |
| 8.1. | DSO OPCODE Registration | 48 |
| 8.2. | DSO RCODE Registration | 48 |
| 8.3. | DSO Type Code Registry | 48 |
| 9. | Security Considerations | 49 |
| 10. | Acknowledgements | 49 |
| 11. | References | 50 |
| 11.1. | Normative References | 50 |
| 11.2. | Informative References | 51 |
| | Authors' Addresses | 52 |

[1.](#) Introduction

The use of transports for DNS other than UDP is being increasingly specified, for example, DNS over TCP [[RFC1035](#)] [[RFC7766](#)] and DNS over TLS [[RFC7858](#)]. Such 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.

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 (the EDNS(0) TCP Keepalive 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 (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.

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

The format for DSO messages ([Section 4.2](#)) 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. 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. It is likely that future updates to these tools will add the ability to recognize, decode, and display the DSO data.

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.

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. 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 "Key words for use in RFCs to Indicate Requirement Levels", when, and only when, they appear in all capitals, as shown here [[RFC2119](#)] [[RFC8174](#)].

"DSO" is used to mean DNS Stateful Operation.

The term "connection" means 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](#)].

The unqualified term "session" in the context of this document means the exchange of DNS messages over a connection where:

- o The connection between client and server is persistent and relatively long-lived (i.e., minutes or hours, rather than seconds).
- o Either end of the connection may initiate messages to the other.

In this document the term "session" is used exclusively as described above, and is in no way related to the anachronistic term "session" as used in the obsolete "seven-layer model" that shaped the design of the failed OSI protocols of the 1980s.

A "DSO Session" is 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](#)].

A "DSO Session" is terminated when the underlying connection is closed. The underlying connection can be closed in two ways:

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.

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.

The term "server" means the software with a listening socket, awaiting incoming connection requests.

The term "client" means the software which initiates a connection to the server's listening socket.

The terms "initiator" and "responder" correspond respectively to the initial sender and subsequent receiver of a DSO request message or unacknowledged message, regardless of which was the "client" and "server" in the usual DNS sense.

The term "sender" may apply to either an initiator (when sending a DSO request message or unacknowledged message) or a responder (when sending a DSO response message).

Likewise, the term "receiver" may apply to either a responder (when receiving a DSO request message or unacknowledged message) or an initiator (when receiving a DSO response message).

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 correctly-implemented peer. 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". Given that these fatal error conditions signify defective software, and given that defective software is likely to remain defective for some time until it is fixed, after forcibly aborting a connection, a client SHOULD refrain from automatically reconnecting to that same server instance for at least one hour.

This document uses the term "same server instance" as follows:

- o In cases where a server is specified or configured using an IP address and TCP port number, two different configurations are referring to the same server instance if they contain the same IP address and TCP port number.

- o In cases where a server is specified or configured using a hostname and TCP port number, such as in the content of a DNS SRV record [[RFC2782](#)], two different configurations (or DNS SRV records) are considered to be referring to the same server instance if they contain the same hostname (subject to the usual case insensitive DNS name matching rules [[RFC1034](#)] [[RFC1035](#)]) and TCP port number. In these cases, configurations with different hostnames are considered to be referring to different server instances, even if those different hostnames happen to be aliases, or happen to resolve to the same IP address(es). Implementations are not expected to resolve hostnames and then perform matching of IP address(es) in order to evaluate whether two entities should be determined to be the "same server instance".

The term "long-lived operations" refers to operations such as Push Notification subscriptions [[I-D.ietf-dnssd-push](#)], Discovery Relay interface subscriptions [[I-D.sctl-dnssd-mdns-relay](#)], and other future long-lived DNS operations that choose to use DSO as their basis, that establish state that persists beyond the lifetime of a traditional brief request/response transaction. This document, the base specification for DNS Stateful Operations, defines a framework for supporting long-lived operations, but does not itself define any long-lived operations. Nonetheless, to appreciate the design rationale behind DNS Stateful Operations, it is helpful to understand the kind of long-lived operations that it is intended to support.

DNS Stateful Operations uses three kinds of message: "DSO request messages", "DSO response messages", and "DSO unacknowledged messages". A DSO request message elicits a DSO response message. DSO unacknowledged messages are unidirectional messages and do not generate any response.

Both DSO request messages and DSO unacknowledged messages are formatted as DNS request messages (the header QR bit is set to zero, as described in [Section 4.2](#)). One difference is that in DSO request messages the MESSAGE ID field is nonzero; in DSO unacknowledged messages it is zero.

The content of DSO messages is expressed using type-length-value (TLV) syntax.

In a DSO request message or DSO unacknowledged message the first TLV is referred to as the "Primary TLV" and determines the nature of the operation being performed, including whether it is an acknowledged or unacknowledged operation; any other TLVs in a DSO request message or unacknowledged message are referred to as "Additional TLVs" and serve additional non-primary purposes, which may be related to the primary purpose, or not, as in the case of the encryption padding TLV.

A DSO response message may contain no TLVs, or it may contain one or more TLVs as appropriate to the information being communicated. In the context of DSO response messages, one or more TLVs with the same DSO-TYPE as the Primary TLV in the corresponding DSO request message are referred to as "Response Primary TLVs". Any other TLVs with different DSO-TYPES are referred to as "Response Additional TLVs". The Response Primary TLV(s), if present, MUST occur first in the response message, before any Response Additional TLVs.

Two timers (elapsed time since an event) are defined in this document:

- o an inactivity timer (see [Section 5.4](#) and [Section 6.1](#))
- o a keepalive timer (see [Section 5.5](#) and [Section 6.1](#))

The timeouts associated with these timers are called the inactivity timeout and the keepalive interval, respectively. The term "Session Timeouts" is used to refer to this pair of timeout values.

Resetting a timer means resetting the timer value to zero and starting the timer again. Clearing a timer means resetting the timer value to zero but NOT starting the timer again.

3. Discussion

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

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(0) TCP Keepalive option [[RFC7828](#)]. The simple set of TLVs defined in this document is sufficient to greatly enhance connection management for this use case.

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. Protocol Details

4.1. DSO Session Establishment

DSO messages MUST be carried in only protocols and in environments where a session may be established according to the definition given above in the Terminology section ([Section 2](#)).

DNS over plain UDP [[RFC0768](#)] is not appropriate since it fails on the requirement for in-order message delivery, and, in the presence of NAT gateways and firewalls with short UDP timeouts, it fails to provide a persistent bi-directional communication channel unless an excessive amount of keepalive traffic is used.

At the time of publication, DSO is specified only for DNS over TCP [[RFC1035](#)] [[RFC7766](#)], and for DNS over TLS over TCP [[RFC7858](#)]. Any use of DSO over some other connection technology needs to be specified in an appropriate future document.

Determining whether a given connection is using DNS over TCP, or DNS over TLS over TCP, is outside the scope of this specification, and must be determined using some out-of-band configuration information. There is no provision within the DSO specification to turn TLS on or off during the lifetime of a connection. For service types where the service instance is discovered using a DNS SRV record [[RFC2782](#)], the specification for that service type SRV name [[RFC6335](#)] will state whether the connection uses plain TCP, or TLS over TCP. For example, the specification for the "_dns-push-tls._tcp" service [[I-D.ietf-dnssd-push](#)], states that it uses TLS. It is a common convention that protocols specified to run over TLS are given IANA service type names ending in "-tls".

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.

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 unacknowledged 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. Similarly, unless it is known in advance by other means that a server does support DSO, a client MUST NOT initiate DSO unacknowledged messages until after a DSO Session has been mutually established.

A DSO Session is established over a connection by the client sending a DSO request message, such as a DSO Keepalive request message ([Section 6.1](#)), and receiving a response, with matching MESSAGE ID, and RCODE set to NOERROR (0), indicating that the DSO request was successful.

If the RCODE in the response is set to DSOTYPENI ("DSO-TYPE Not Implemented", 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, so all implementations of DSO are required to implement the Keepalive TLV. 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.

If the RCODE is set to any value other than NOERROR (0) or DSOTYPENI (tentatively 11), 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 SHOULD NOT issue further DSO messages on that connection.

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.

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](#)].

Note that for clients that implement only the DSO-TYPEs defined in this base specification, sending a DSO 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.

4.1.1. 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 target hostname and port, the client SHOULD use a single shared DSO Session for all those operations. A client SHOULD NOT open multiple connections to the same target host and port just because the names being operated on are different or happen to fall within different zones. This requirement is to reduce unnecessary connection load on the DNS 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.

4.1.2. Zero Round-Trip Operation

There is increased awareness today of the performance benefits of eliminating round trips in session establishment. Technologies like TCP Fast Open [[RFC7413](#)] and TLS 1.3 [[I-D.ietf-tls-tls13](#)] provide mechanisms to reduce or eliminate round trips in session establishment.

Similarly, DSO supports zero round-trip operation.

Having initiated a connection to a server, possibly using zero round-trip TCP Fast Open and/or zero round-trip TLS 1.3, a client MAY send multiple response-requiring DSO request messages to the server in succession without having to wait for a response to the first request message to confirm successful establishment of a DSO session.

However, a client MUST NOT send non-response-requiring DSO request messages until after a DSO Session has been mutually established.

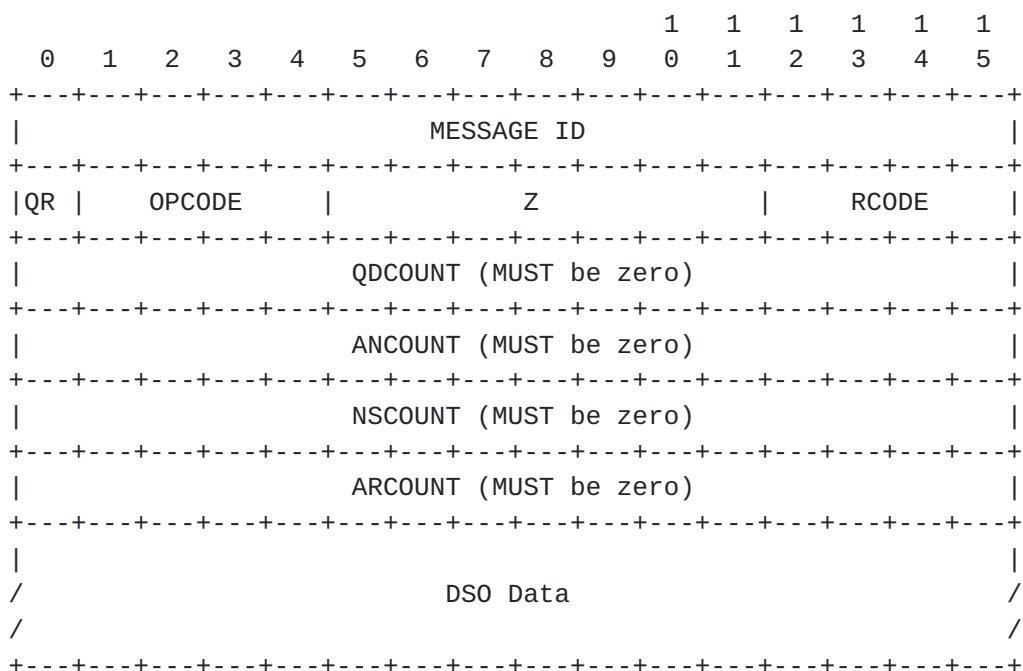
Similarly, a server MUST NOT send DSO request messages until it has received a response-requiring DSO request message from a client and transmitted a successful NOERROR response for that request.

Caution must be taken to ensure that DSO messages sent before the first round-trip is completed 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.

4.1.3. Middlebox Considerations

Where an application-layer middlebox (e.g., a DNS proxy, forwarder, or session multiplexer) is in the path, the middlebox MUST NOT blindly forward DSO messages in either direction, and MUST treat the inbound and outbound connections as separate sessions. This does not preclude the use of DSO messages in the presence of an IP-layer middlebox, such as a NAT that rewrites IP-layer and/or transport-layer headers but otherwise preserves the effect of a single session between the client and the server.

To illustrate the above, consider a network where a middlebox terminates one or more TCP connections from clients and multiplexes the queries therein over a single TCP connection to an upstream server. The DSO messages and any associated state are specific to the individual TCP connections. A DSO-aware middlebox MAY in some circumstances be able to retain associated state and pass it between the client and server (or vice versa) but this would be highly TLV-specific. For example, the middlebox may be able to maintain a list of which clients have made Push Notification subscriptions [[I-D.ietf-dnssd-push](#)] and make its own subscription(s) on their behalf, relaying any subsequent notifications to the client (or clients) that have subscribed to that particular notification.



4.2.1. DNS Header Fields in DSO Messages

In an unacknowledged message the MESSAGE ID field MUST be set to zero. In an acknowledged 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 request 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.sctl-dnssd-mdns-relay](#)].

Whether a message is acknowledged or unacknowledged is determined only by the specification for the Primary TLV. An acknowledgment cannot be requested by including a nonzero message ID in a message the primary TLV of which is specified to be unacknowledged, nor can an acknowledgment be prevented by sending a message ID of zero in a message with a primary TLV that is specified to be acknowledged. A responder that receives either such malformed message MUST treat it as a fatal error and forcibly abort the connection immediately.

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

In a 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 response message (QR=1) the MESSAGE ID field MUST contain a copy of the value of the MESSAGE ID field in the request message being responded to. In a response message (QR=1) the MESSAGE ID field MUST NOT be zero. If a 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 (tentatively 6).

The Z bits are currently unused in DSO messages, and in both DSO requests and DSO responses the Z bits MUST be set to zero (0) on transmission and MUST be silently ignored on reception, unless a future IETF Standard specifies otherwise.

In a DNS request message (QR=0) the RCODE is set according to the definition of the request. For example, in a Retry Delay message ([Section 5.6.1](#)) the RCODE indicates the reason for termination. However, in most cases, except where clearly specified otherwise, in a DNS 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 request due to a problem with the server |
| 3 | NXDOMAIN | Name Error -- Named entity does not exist (TLV-dependent) |
| 4 | NOTIMP | DSO not supported |
| 5 | REFUSED | Operation declined for policy reasons |
| 9 | NOTAUTH | Not Authoritative (TLV-dependent) |
| 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 NXDOMAIN (Name Error) or NOTAUTH (Not Authoritative) then that document MUST specify the specific interpretation of these RCODE values in the context of that new DSO TLV.

4.2.2. 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 [Section 4.2.2.1](#).

A DSO message may be a request message, a response message, or an unacknowledged message.

A DSO request message or DSO unacknowledged message MUST contain at least one TLV. The first TLV in a DSO request message or DSO unacknowledged message is referred to as the "Primary TLV" and determines the nature of the operation being performed, including whether it is an acknowledged or unacknowledged operation. In some cases it may be appropriate to include other TLVs in a request message or unacknowledged message, such as the Encryption Padding TLV ([Section 6.3](#)), and these extra TLVs are referred to as the "Additional TLVs".

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.

A DSO response message may contain one or more TLVs with DSO-TYPE the same as the Primary TLV from the corresponding DSO request message, in which case those TLV(s) are referred to as "Response Primary TLVs". A DSO response message is not required to carry Response Primary TLVs. 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 request messages, which generate corresponding responses. In some specialized high-traffic use cases, it may be appropriate to specify unacknowledged messages. Unacknowledged messages can be more efficient on the network, because they don't generate a stream of corresponding reply messages. Using unacknowledged 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 DNS request message (i.e., QR=0), that message is to be unacknowledged, the MESSAGE ID field MUST be set to zero and the receiver MUST NOT generate any response message corresponding to this unacknowledged message.

The previous point, that the receiver MUST NOT generate responses to unacknowledged 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.

Unacknowledged 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 of the request message (the request message does not contain a unique MESSAGE ID with which to associate a response with its corresponding request). Unacknowledged messages are only appropriate in cases where the sender already knows that the receiver supports, and wishes to receive, these messages.

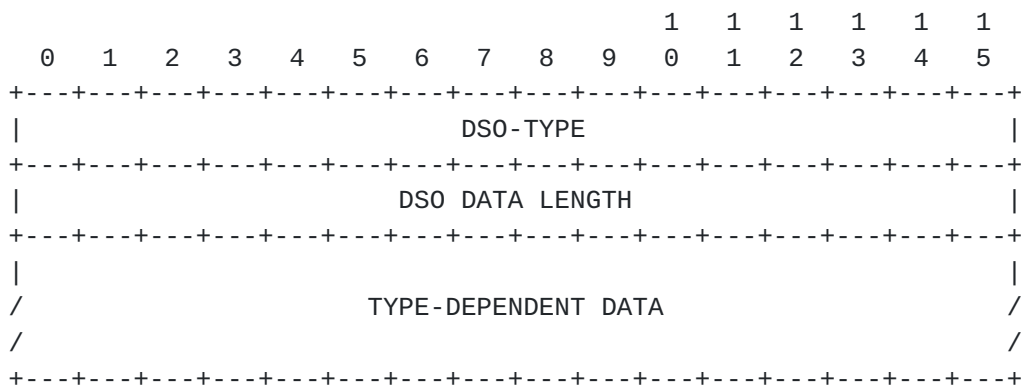
For example, after a client has subscribed for Push Notifications [[I-D.ietf-dnssd-push](#)], the subsequent event notifications are then sent as unacknowledged messages, and this is appropriate because the client initiated the message stream by virtue of its Push Notification subscription, thereby indicating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

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

[4.2.2.1.](#) TLV Syntax

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

The specification for a TLV states whether that DSO-TYPE may be used in "Primary", "Additional", "Response Primary", or "Response Additional" TLVs. The specification for a TLV also states whether, when used as the Primary (i.e., first) TLV in a DNS request message (i.e., QR=0), that DSO message is to be acknowledged. If the DSO message is to be acknowledged, the specification also states 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 stated in the specification for that TLV.



DS0-TYPE: A 16-bit unsigned integer, in network (big endian) byte order, giving the DS0-TYPE of the current DS0 TLV per the IANA DS0 Type Code Registry.

DS0 DATA LENGTH: A 16-bit unsigned integer, in network (big endian) byte order, giving the size in bytes of the TYPE-DEPENDENT DATA.

TYPE-DEPENDENT DATA: Type-code specific format. The generic DS0 machinery treats the TYPE-DEPENDENT DATA as an opaque "blob" without attempting to interpret it. Interpretation of the meaning of the TYPE-DEPENDENT DATA for a particular DS0-TYPE is the responsibility of the software that implements that DS0-TYPE.

4.2.2.2. Request TLVs

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

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

4.2.2.3. Response TLVs

Depending on the operation, a DSO response message **MAY** contain no TLVs, because it is simply a response to a previous request message, and the MESSAGE ID in the header is sufficient to identify the 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 request 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.

4.2.2.4. 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 (tentatively 11). The error response MUST NOT contain a copy of the unrecognized Primary TLV.

If DSO unacknowledged 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 unacknowledged 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.

4.2.3. EDNS(0) and TSIG

Since the ARCOUNT field MUST be zero, a DSO message MUST NOT contain an 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 6.3](#) MUST be used.

Similarly, a DSO message MUST NOT contain a TSIG record. A TSIG record in a conventional DNS message is added as the last record in the additional records section, and carries a signature computed over the preceding message content. Since DSO data appears *after* the additional records section, it would not be included in the signature calculation. If use of signatures with DSO messages becomes necessary in the future, a new DSO TLV needs 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(0) TCP Keepalive 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(0) TCP Keepalive Option this is a fatal error and the recipient of the defective message MUST forcibly abort the connection immediately.

4.3. Message Handling

The initiator MUST set the value of the QR bit in the DNS header to zero (0), and the responder MUST set it to one (1).

As described above in [Section 4.2.1](#) whether an outgoing message with QR=0 is unacknowledged or acknowledged 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.

A DSO unacknowledged message has both the QR bit and the MESSAGE ID field set to zero, and MUST NOT elicit a response.

Every DSO request message (QR=0) with a nonzero MESSAGE ID field is an acknowledged DSO request, and MUST elicit a corresponding response (QR=1), which MUST have the same MESSAGE ID in the DNS message header as in the corresponding request.

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.

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 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 4.2.1](#), an initiator MUST NOT reuse a MESSAGE ID that it already has in use for an outstanding request (unless specified otherwise by the relevant specification for the DSO-TYPE in question). At the very least, this means that a MESSAGE ID MUST NOT be reused in a particular direction on a particular DSO

Session while the initiator is waiting for a response to a previous request 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 MUST NOT 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.

4.3.1. Error Responses

When a DSO unacknowledged message is unsuccessful for some reason, the responder immediately aborts the connection.

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 5.6.1](#).

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 5.6.1](#)) then the client SHOULD continue to use that DSO Session for subsequent operations.

4.4. DSO Response Generation

With most TCP implementations, for DSO requests that generate a response, the TCP data acknowledgement (generated because data has been received by TCP), the TCP window update (generated because TCP has delivered that data to the receiving software), and the DSO response (generated by the receiving application-layer software itself) are all combined into a single IP packet. Combining these three elements into a single IP packet can give a significant improvement in network efficiency.

For DSO requests that do not generate a response, the TCP implementation generally doesn't have any way to know that no response will be forthcoming, so it waits fruitlessly for the application-layer software to generate a response, until the Delayed ACK timer fires [[RFC1122](#)] (typically 200 milliseconds) and only then does it send the TCP ACK and window update. In conjunction with Nagle's Algorithm at the sender, this can delay the sender's transmission of its next (non-full-sized) TCP segment, while the sender is waiting for its previous (non-full-sized) TCP segment to be acknowledged, which won't happen until the Delayed ACK timer fires. Nagle's Algorithm exists to combine multiple small application writes into more-efficient large TCP segments, to guard against wasteful use of the network by applications that would otherwise transmit a stream of small TCP segments, but in this case Nagle's Algorithm (created to improve network efficiency) can interact badly with TCP's Delayed ACK feature (also created to improve network efficiency) [[NagleDA](#)] with the result of delaying some messages by up to 200 milliseconds.

Possible mitigations for this problem include:

- o Disable Nagle's Algorithm at the sender. This is not great, because it results in less efficient use of the network.
- o Disable Delayed ACK at the receiver. This is not great, because it results in less efficient use of the network.
- o Use a networking API that lets the receiver signal to the TCP implementation that the receiver has received and processed a client request for which it will not be generating any immediate response. This allows the TCP implementation to operate efficiently in both cases; for requests that generate a response, the TCP ACK, window update, and DSO response are transmitted together in a single TCP segment, and for requests that do not generate a response, the application-layer software informs the TCP implementation that it should go ahead and send the TCP ACK and window update immediately, without waiting for the Delayed ACK timer. Unfortunately it is not known at this time which (if any)

of the widely-available networking APIs currently include this capability.

4.5. 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.sctl-dnssd-mdns-relay](#)].

Generally speaking, a long-lived operation is initiated by the initiator, and, if successful, remains 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 previously valid 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.

5. DSO Session Lifecycle and Timers

5.1. DSO Session Initiation

A DSO Session begins as described in [Section 4.1](#).

The client may perform as many DNS operations as it wishes using the newly created DSO Session. Operations SHOULD be pipelined (i.e., the client doesn't need wait for a response before sending the next message). The server MUST act on messages in the order they are transmitted, but responses to those messages SHOULD be sent out of order when appropriate.

5.2. 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 DSO Keepalive TLV ([Section 6.1](#)).

The first timeout value, the inactivity timeout, is the maximum time for which a client may speculatively keep a 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 keepalive traffic to maintain that session.

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 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 very 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.

5.3. Inactive DSO Sessions

At both servers and clients, the generation or reception of any complete DNS message, including DNS requests, responses, updates, or DSO messages, resets both timers for that DSO Session, with the 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.sctl-dnssd-mdns-relay](#)]), an operation is considered in progress for as long as the operation is active, 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.

5.4. The Inactivity Timeout

The purpose of the inactivity timeout is for the server to balance its 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.

5.4.1. 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 **SHOULD** consider the client delinquent, and **SHOULD** 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 6.1](#).

5.4.2. 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 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 Keepalive message dictating new Session Timeout values to the client.

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

5.5. 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.

5.5.1. 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 6.1](#)). 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 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.

5.5.2. Values for the Keepalive Interval

For the keepalive interval value, lower values result in a higher volume of 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 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 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 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](#)].

Note that the lower the keepalive interval value, the higher the load on client and server. For example, a hypothetical keepalive interval value of 100ms would result in a continuous stream of at least ten messages per second, in both directions, to keep the DSO Session alive. And, in this extreme example, a single packet loss and retransmission over a long path could introduce a momentary pause in the stream of messages, long enough to cause the server to overzealously abort the connection.

Because of this concern, the server MUST NOT send a 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 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 keepalive traffic. Note that after signaling that the client should generate no keepalive traffic in this way, at any point the server may revise that keepalive traffic requirement by sending a new Keepalive message dictating new Session Timeout values to the client.

The largest *finite* keepalive interval supported by the current DSO Keepalive TLV is 0xFFFFFFFEE ($2^{32}-2$ milliseconds, approximately 49.7 days).

5.6. Server-Initiated Session Termination

In addition to cancelling individual long-lived operations selectively ([Section 4.5](#)) 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.

Normally a server **MUST NOT** close a DSO Session with a client. A server only causes a DSO Session to be ended in the exceptional circumstances outlined below. 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.

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).
- 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 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 5.2](#)).
- 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.

5.6.1. 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. For example, an RCODE value of SERVFAIL indicates that the server is overloaded due to resource exhaustion, or is restarting. An RCODE value of NOTAUTH indicates that the server has been reconfigured and is no longer able to perform one or more of the functions currently being performed on this DSO Session because it no longer has authority over the names in question. An RCODE value of REFUSED indicates a policy change regarding this session. An RCODE value of FORMERR indicates that the client requests are too badly malformed for the session to continue. The format of the Retry Delay TLV is described in [Section 6.2](#). After sending a Retry Delay message, the server MUST NOT send any further messages on that DSO Session.

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.

5.6.1.1. 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).

5.6.1.2. 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 server instance, or to another suitable server instance, if more than one is available. If reconnecting to the same server instance, the client MUST respect the indicated delay, if available, before attempting to reconnect.

If the server 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 server instance does not want a client to reconnect ever (perhaps the server 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 server 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 server instance the client should use.

6. Base TLVs for DNS Stateful Operations

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

6.1. Keepalive TLV

The Keepalive TLV (DSO-TYPE=1) performs two functions: to reset the keepalive timer for the DSO Session, and to establish the values for the Session Timeouts.

The TYPE-DEPENDENT DATA for the the Keepalive TLV is as follows:

```

          1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 3 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                                INACTIVITY TIMEOUT (32 bits)                                |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                                KEEPALIVE INTERVAL (32 bits)                               |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+

```

INACTIVITY TIMEOUT: The inactivity timeout for the current DSO Session, specified as a 32-bit unsigned integer, in network (big endian) byte order, in units of milliseconds. This is the timeout at which the client **MUST** begin closing an inactive DSO Session. The inactivity timeout can be any value of the server's choosing. If the client does not gracefully close an inactive DSO Session, then after twice this interval, or five seconds, whichever is greater, the server will forcibly abort the connection.

KEEPALIVE INTERVAL: The keepalive interval for the current DSO Session, specified as a 32-bit unsigned integer, in network (big endian) byte order, in units of milliseconds. This is the interval at which a client **MUST** generate keepalive traffic to maintain connection state. The keepalive interval **MUST NOT** be less than ten seconds. If the client does not generate the mandated 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 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 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 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 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 Keepalive message does not reset the inactivity timer.

When sent by a client, the Keepalive request message **MUST** be sent as an acknowledged request, with a nonzero MESSAGE ID. If a server receives a Keepalive DSO message with a zero MESSAGE ID then this is a fatal error and the server **MUST** forcibly abort the connection immediately. The Keepalive request message resets a DSO Session's keepalive timer, and at the same time communicates to the server the the client's requested Session Timeout values. In a server response to a client-initiated 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 Keepalive DSO requests 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 4.1](#)) a Keepalive message **MAY** be initiated by a server. When sent by a server, the Keepalive message **MUST** be sent as an unacknowledged 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 Keepalive request message with a nonzero MESSAGE ID then this is a fatal error and the client **MUST** forcibly abort the connection immediately. The Keepalive unacknowledged 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 message to this unilateral declaration is required or allowed.

The Keepalive TLV is not used as an Additional TLV.

In response messages the Keepalive TLV is used only as a Response Primary TLV, replying to a Keepalive request message from the client. A Keepalive TLV MUST NOT be added as to other responses a Response Additional TLV. If the server wishes to update a client's Session Timeout values other than in response to a Keepalive request message from the client, then it does so by sending an unacknowledged Keepalive 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 Keepalive request message is the only way for a client to initiate a DSO Session.

6.1.1. 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 DSO 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 DSO 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 DSO 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.

6.1.2. Relation to EDNS(0) TCP Keepalive Option

The inactivity timeout value in the Keepalive TLV (DSO-TYPE=1) has similar intent to the EDNS(0) TCP Keepalive Option [[RFC7828](#)]. A client/server pair that supports DSO MUST NOT use the EDNS(0) TCP KeepAlive option within any message after a DSO Session has been established. Once a DSO Session has been established, if either client or server receives a DNS message over the DSO Session that contains an EDNS(0) TCP Keepalive option, this is a fatal error and the receiver of the EDNS(0) TCP Keepalive option MUST forcibly abort the connection immediately.

6.2. Retry Delay TLV

The Retry Delay TLV (DSO-TYPE=2) can be used as a Primary TLV (unacknowledged) in a server-to-client message, or as a Response Additional TLV in either direction.

The TYPE-DEPENDENT DATA for the the Retry Delay TLV is as follows:

```

          1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 3 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                                RETRY DELAY (32 bits)                                |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+

```

RETRY DELAY: A time value, specified as a 32-bit unsigned integer, in network (big endian) byte order, in units of milliseconds, within which the initiator **MUST NOT** retry this operation, or retry connecting to this server. Recommendations for the RETRY DELAY value are given in [Section 5.6.1](#).

6.2.1. 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 an unacknowledged 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 5.6.1](#). The RCODE in the message header **MUST** indicate the reason for the termination:

- o NOERROR indicates a routine shutdown.
- o FORMERR indicates that the client requests are too badly malformed for the session to continue.
- o SERVFAIL indicates that the server is overloaded due to resource exhaustion, or is restarting.
- o REFUSED indicates that the server has been reconfigured and due to a policy change is no longer able to perform one or more of the functions currently being performed on this DSO Session
- o NOTAUTH indicates that the server has been reconfigured and is no longer able to perform one or more of the functions currently being performed on this DSO Session because it no longer has 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 Retry Delay request. Servers sending Retry Delay requests SHOULD use one of these values. However, future circumstances may create situations where other RCODE values are appropriate in Retry Delay requests, so clients MUST be prepared to accept Retry Delay requests with any RCODE value.

The RCODE value received in the message header of a Retry Delay unacknowledged message is generally usually useful 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 server 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).

A Retry Delay message from server to client is an unacknowledged 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 request message or DSO unacknowledged message to a server. If a server receives a DNS request message (i.e., QR=0) where the Primary TLV is the Retry Delay TLV, this is a fatal error and the server MUST forcibly abort the connection immediately.

6.2.2. Retry Delay TLV used as a Response Additional TLV

In the case of a request that returns 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.

6.3. 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 TYPE-DEPENDENT DATA for the the Padding TLV is optional and is a variable length field containing non-specified values. A DATA LENGTH of 0 essentially provides for 4 bytes of padding (the minimum amount).

| | | | | | | | | | | | | | | | | |
|--|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | 1 | 1 | 1 | 1 | 1 | 1 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 | 1 | 2 | 3 | 4 | 5 | |
| +---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+--- | | | | | | | | | | | | | | | | |
| / | | | | | | | | | | | | | | | | |
| / | | | | | | | | | | | | | | | | |
| / | | | | | | | | | | | | | | | | |
| +---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+--- | | | | | | | | | | | | | | | | |

VARIABLE NUMBER OF BYTES

As specified for the EDNS(0) Padding Option [[RFC7830](#)] the PADDING bytes SHOULD be set to 0x00. Other values MAY be used, for example, in cases where there is a concern that the padded message could be subject to compression before encryption. PADDING bytes of any value MUST be accepted in the messages received.

The Encryption Padding TLV may be included in either a DSO request, response, or both. As specified for the EDNS(0) Padding Option [[RFC7830](#)] if a request is received with an Encryption Padding TLV, then the 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](#)].

7. Summary Highlights

This section summarizes some noteworthy highlights about various components of the DSO protocol.

7.1. 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 Unacknowledged 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 | Unacknowledged Message | | Request Message | |
| +-----+ | | | | |
| QR=1 | Invalid - Fatal Error | | Response Message | |
| +-----+ | | | | |

7.2. 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 requests or unacknowledged 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 Unacknowledged 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 request message or unacknowledged message from client to server.
- o CRP - Response Primary TLV, included in response message sent to back 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 to back 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: requests or unacknowledged messages from server to client, and the corresponding responses from client back to server.

| | +-----+-----+-----+-----+-----+ | | | | | +-----+-----+-----+-----+-----+ | | | | |
|------------|---------------------------------|-----|-----|-----|-----|---------------------------------|-----|-----|-----|-----|
| | C-P | C-U | C-A | CRP | CRA | S-P | S-U | S-A | SRP | SRA |
| +-----+ | +-----+-----+-----+-----+-----+ | | | | | | | | | |
| KeepAlive | X | | | X | | | X | | | |
| +-----+ | +-----+-----+-----+-----+-----+ | | | | | | | | | |
| RetryDelay | | | | | 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.

8. IANA Considerations

8.1. DSO OPCODE Registration

The IANA is requested to record the value (tentatively) 6 for the DSO OPCODE in the DNS OPCODE Registry. DSO stands for DNS Stateful Operations.

8.2. DSO RCODE Registration

The IANA is requested to record the value (tentatively) 11 for the DSOTYPENI error code in the DNS RCODE Registry. The DSOTYPENI error code ("DSO-TYPE Not Implemented") signifies that the DSO-TYPE of the primary TLV in the DSO request message is not implemented by the receiver.

8.3. 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 | Status | Reference |
|-----------|---|----------|-----------|
| 0000 | Reserved | Standard | RFC-TBD |
| 0001 | KeepAlive | Standard | RFC-TBD |
| 0002 | RetryDelay | Standard | RFC-TBD |
| 0003 | EncryptionPadding | Standard | RFC-TBD |
| 0004-003F | Unassigned, reserved for DSO session-management TLVs | | |
| 0040-F7FF | Unassigned | | |
| F800-FBFF | Reserved for experimental/local use | | |
| FC00-FFFF | Reserved for future expansion | | |

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](#)].

Requests to register additional new DSO Type Codes in the "Unassigned" range 0040-F7FF are to be recorded by IANA after Expert Review [[RFC8126](#)]. At the time of publication of this document, the Designated Expert for the newly created DSO Type Code registry is [*TBD*].

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.

9. 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 Retry Delay Unacknowledged Message into the data stream, specifying an unreasonably large RETRY DELAY, causing a denial-of-service attack against the client.

10. Acknowledgements

Thanks to Stephane Bortzmeyer, Tim Chown, Ralph Droms, Paul Hoffman, Jan Komissar, Edward Lewis, Allison Mankin, Rui Paulo, David Schinazi, Manju Shankar Rao, and Bernie Volz for their helpful contributions to this document.

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

- [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>>.
- [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>>.

11.2. Informative References

- [I-D.ietf-dnssd-push]
Pusateri, T. and S. Cheshire, "DNS Push Notifications", [draft-ietf-dnssd-push-13](#) (work in progress), October 2017.
- [I-D.ietf-dprive-padding-policy]
Mayrhofer, A., "Padding Policy for EDNS(0)", [draft-ietf-dprive-padding-policy-04](#) (work in progress), February 2018.
- [I-D.ietf-tls-tls13]
Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", [draft-ietf-tls-tls13-24](#) (work in progress), February 2018.
- [I-D.sctl-dnssd-mdns-relay]
Cheshire, S. and T. Lemon, "Multicast DNS Discovery Relay", [draft-sctl-dnssd-mdns-relay-02](#) (work in progress), November 2017.
- [NagleDA] Cheshire, S., "TCP Performance problems caused by interaction between Nagle's Algorithm and Delayed ACK", May 2005, <<http://www.stuartcheshire.org/papers/nagledelayedack/>>.
- [RFC0768] Postel, J., "User Datagram Protocol", STD 6, [RFC 768](#), DOI 10.17487/RFC0768, August 1980, <<https://www.rfc-editor.org/info/rfc768>>.

- [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>>.
- [RFC2782] Gulbrandsen, A., Vixie, P., and L. Esibov, "A DNS RR for specifying the location of services (DNS SRV)", [RFC 2782](#), DOI 10.17487/RFC2782, February 2000, <<https://www.rfc-editor.org/info/rfc2782>>.
- [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>>.
- [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>>.
- [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>>.

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.
1 Infinite Loop
Cupertino CA 95014
USA

Phone: +1 408 974 3207
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
Barefoot Consulting
Brattleboro
VT 05301
USA

Email: mellon@fugue.com

Tom Pusateri
Unaffiliated
Raleigh NC 27608
USA

Phone: +1 919 867 1330
Email: pusateri@bangj.com

