Internet Engineering Task Force (IETF)
Request for Comments: 8490
Updates: 1035, 7766
Category: Standards Track
ISSN: 2070-1721

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

March 2019

DNS Stateful Operations

Abstract

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

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc8490.

Table of Contents

1. Introduction

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

The existing Extension Mechanism for DNS (EDNS(0)) [RFC6891] is explicitly defined to only have "per-message" semantics. While EDNS(0) has been used to signal at least one session-related parameter (edns-tcp-keepalive EDNS(0) Option [RFC7828]), the result is less than optimal due to the restrictions imposed by the EDNS(0) semantics and the lack of server-initiated signaling. 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 for DNS Stateful Operations (DSO) with a value of 6. DSO messages are used to communicate operations within persistent stateful sessions, expressed using Type Length Value (TLV) syntax. This document defines an initial set of three TLVs used to manage session timeouts, termination, and encryption padding.

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

DSO messages may or may not be acknowledged. Whether a DSO message is to be acknowledged (a DSO request message) or is not to be acknowledged (a DSO unidirectional message) is specified in the definition of that particular DSO message type. The MESSAGE ID is nonzero for DSO request messages, and zero for DSO unidirectional messages. Messages are pipelined and responses may appear out of order when multiple requests are being processed concurrently.

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

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

When displayed using packet analyzer tools that have not been updated to recognize the DSO format, this will result in the DSO data being displayed as unknown extra data after the end of the DNS message.

This new format has distinct advantages over an RR-based format because it is more explicit and more compact. Each TLV definition is specific to its use case and, as a result, contains no redundant or overloaded fields. Importantly, it completely avoids conflating DNS Stateful Operations in any way with normal DNS operations or with existing EDNS(0)-based functionality. A goal of this approach is to avoid the operational issues that have befallen EDNS(0), particularly relating to middlebox behavior (see sections discussing EDNS(0), and problems caused by firewalls and load balancers, in the recent work describing causes of DNS failures [Fail]).

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

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

2. Requirements Language

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

3. Terminology

DSO: DNS Stateful Operations.

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

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

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

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

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

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

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

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

responder: the software that receives a DSO request message or a DSO unidirectional message during a DSO Session. Either a client or a server can be a responder.

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

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

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

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

early data: a TLS 1.3 handshake containing data on the first flight that begins a DSO Session (Section 2.3 of the TLS 1.3 specification [RFC8446]). TCP Fast Open [RFC7413] is only permitted when using TLS.

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

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

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

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

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

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

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

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

Additional TLV: any TLVs that follow the Primary TLV in a DSO request message or DSO unidirectional message.

Response Primary TLV: in a DSO response, any TLVs with the same DSO-TYPE as the Primary TLV from the corresponding DSO request message. If present, any Response Primary TLV(s) MUST appear first in the DSO response message, before any Response Additional TLVs.

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

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

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

session timeouts: the inactivity timer and the keepalive timer.

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

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

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

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

4. Applicability

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

4.1. Use Cases

Several use cases for DNS Stateful Operations are described below.

4.1.1. Session Management

In one use case, establishing session parameters such as server-defined timeouts is of great use in the general management of persistent connections. For example, using DSO Sessions for stub-to-recursive DNS-over-TLS [RFC7858] is more flexible for both the client and the server than attempting to manage sessions using just the edns-tcp-keepalive EDNS(0) Option [RFC7828]. The simple set of TLVs defined in this document is sufficient to greatly enhance connection management for this use case.

4.1.2. Long-Lived Subscriptions

In another use case, DNS-based Service Discovery (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 [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 sustaining sufficient traffic to maintain NAT and firewall state. To mitigate this issue, this document introduces a new concept for the DNS -- DSO "keepalive traffic". This traffic carries no DNS data and is not considered 'activity' in the classic DNS sense, but it serves to maintain state in middleboxes and to assure the client and server that they still have connectivity to each other.

4.2. Applicable Transports

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

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

DNS Stateful Operations are not applicable for transports that cannot support clean session semantics or that do not guarantee in-order delivery. While in principle such a transport could be constructed over UDP, the current specification of DNS-over-UDP [RFC1035] does not provide in-order delivery or session semantics and hence cannot be used. Similarly, DNS-over-HTTP [RFC8484] cannot be used because HTTP has its own mechanism for managing sessions, which is incompatible with the mechanism specified here.

Only DNS-over-TCP and DNS-over-TLS are currently defined for use with DNS Stateful Operations. Other transports may be added in the future if they meet the requirements set out in the first paragraph of this section.

5. Protocol Details

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

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

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

DSO Session Establishment in Progress: A client has sent a DSO request within the last 30 seconds, but has not yet received a DSO response for that request. In this phase, the client may send more DSO requests and more DNS requests, but MUST NOT send DSO unidirectional messages (Section 5.1).

DSO Session Establishment Timeout: A client has sent a DSO request, and after 30 seconds has still received no DSO response for that request. This means that the server is now in an indeterminate state. The client forcibly aborts the connection. The client MAY reconnect without using DSO, if appropriate.

DSO Session Establishment Failed: A client has sent a DSO request, and received a corresponding DSO response with a nonzero RCODE. This means that the attempt to establish the DSO Session did not succeed. At this point, the client is permitted to continue operating without a DSO Session (Connected but Sessionless) but does not send further DSO messages (Section 5.1).

DSO Session Established: A client has sent a DSO request, and received a corresponding DSO response with RCODE set to NOERROR (0). A DSO Session has now been successfully established. Both client and server may send DSO messages and DNS messages; both may send replies in response to messages they receive (Section 5.2). The inactivity timer (Section 6.4) is active; the keepalive timer (Section 6.5) is active. Standard DNS-over-TCP inactivity timeout handling is no longer in effect [RFC7766] (see Section 7.1.2 of this document).

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

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

Reconnect: The client disconnected as a result of a server shutdown. The client either waits for the server-specified Retry Delay to expire (Section 6.6.3) or else contacts a different server instance. If the client no longer needs service, it does not reconnect.

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

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

5.1. DSO Session Establishment

In order for a session to be established between a client and a server, the client must first establish a connection to the server using an applicable transport (see Section 4.2).

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

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

described below. This is referred to as explicit DSO Session establishment.

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

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

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

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

5.1.1. DSO Session Establishment Failure

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

If the RCODE in the response is set to DSOTYPENI ("DSO-TYPE Not Implemented"; RCODE 11), this indicates that the server does support DSO but does not implement the DSO-TYPE of the Primary TLV in this DSO request message. A server implementing DSO MUST NOT return DSOTYPENI for a DSO Keepalive request message because the Keepalive TLV is mandatory to implement. But in the future, if a client attempts to establish a DSO Session using a response-requiring DSO request message using some newly-defined DSO-TYPE that the server does not understand, that would result in a DSOTYPENI response. If the server returns DSOTYPENI, then a DSO Session is not considered established. The client is, however, 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.

When a DSO message is received by an existing DNS server that doesn't recognize the DSO OPCODE, two other possible outcomes exist: the server might send no response to the DSO message, or the server might drop the connection.

If the server sends no response to the DSO message, the client SHOULD wait 30 seconds, after which time the server will be assumed not to support DSO. If the server doesn't respond within 30 seconds, it can be assumed that it is not going to respond; this leaves it in an unspecified state: there is no specification requiring that a response be sent to an unknown message, but there is also no specification stating what state the server is in if no response is sent. Therefore the client MUST forcibly abort the connection to the server. The client MAY reconnect, but not use DSO, if appropriate (Section 6.6.3.1). By disconnecting and reconnecting, the client ensures that the server is in a known state before sending any subsequent requests.

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

5.1.2. DSO Session Establishment Success

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

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

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

5.2. Operations after DSO Session Establishment

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

Because a server that supports DNS Stateful Operations MUST return an RCODE of "NOERROR" when it receives a Keepalive TLV DSO request message, the Keepalive TLV is an ideal candidate for use in establishing a DSO Session. Any other option that can only succeed

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

5.3. DSO Session Termination

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

- Where this specification says "close gracefully", it means sending a TLS close_notify (if TLS is in use) followed by a TCP FIN, or the equivalent 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 in order to place the burden of TCP's TIME-WAIT state on the client rather than the server.

- Where this specification says "forcibly abort", it means sending a TCP RST or the equivalent for other protocols. In the BSD Sockets API, this is achieved by setting the SO_LINGER option to zero before closing the socket.

5.3.1. Handling Protocol Errors

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

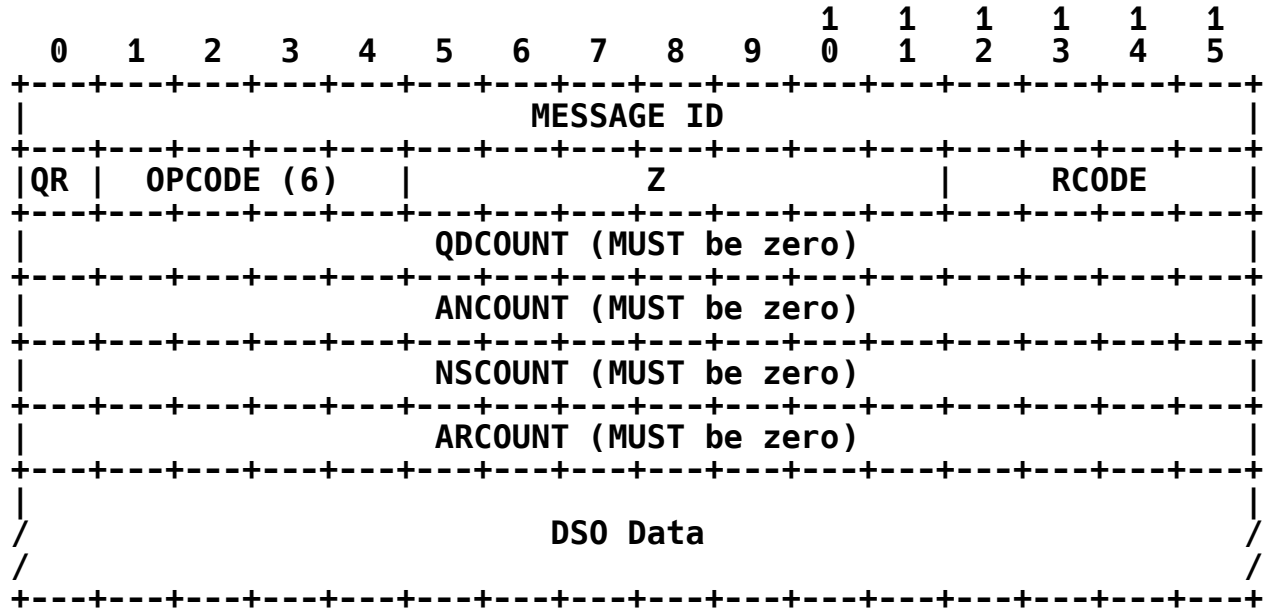
The second is situations that should never happen when communicating with a compliant DSO implementation. If they do happen, they indicate a serious flaw in the protocol implementation beyond what is reasonable to expect software to recover from. This document

describes this latter form of error condition as a "fatal error" and specifies that an implementation encountering a fatal error condition "MUST forcibly abort the connection immediately".

5.4. Message Format

A DSO message begins with the standard twelve-byte DNS message header [RFC1035] with the OPCODE field set to the DSO OPCODE (6). However, unlike standard DNS messages, the question section, answer section, authority records section, and additional records sections are not present. The corresponding count fields (QDCOUNT, ANCOUNT, NSCOUNT, ARCOUNT) MUST be set to zero on transmission.

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



5.4.1. DNS Header Fields in DSO Messages

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

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

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

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

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

The DNS Header OPCODE field holds the DSO OPCODE value (6).

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

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

specified otherwise, the RCODE is set to zero on transmission, and silently ignored on reception.

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

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

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

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

The RCODE field is followed by the four zero-valued count fields, followed by the DSO Data.

5.4.2. DSO Data

The standard twelve-byte DNS message header with its zero-valued count fields is followed by the DSO Data, expressed using TLV syntax, as described in Section 5.4.4.

A DSO request message or DSO unidirectional message MUST contain at least one TLV. The first TLV in a DSO request message or DSO unidirectional message is referred to as the "Primary TLV" and determines the nature of the operation being performed, including whether it is a DSO request or a DSO unidirectional operation. In some cases, it may be appropriate to include other TLVs in a DSO request message or DSO unidirectional message, such as the DSO

Encryption Padding TLV (Section 7.3). Additional TLVs follow the Primary TLV. Additional TLVs are not limited to what is defined in this document. New Additional TLVs may be defined in the future. Their definitions will describe when their use is appropriate.

An unrecognized Primary TLV results in a DSOTYPENI error response. Unrecognized Additional TLVs are silently ignored, as described in Sections 5.4.5 and 8.2.

A DSO response message may contain no TLVs, or may contain one or more TLVs, appropriate to the information being communicated.

Any TLVs with the same DSO-TYPE as the Primary TLV from the corresponding DSO request message are Response Primary TLV(s) and MUST appear first in a DSO response message. A DSO response message may contain multiple Response Primary TLVs, or a single Response Primary TLV, or in some cases, no Response Primary TLV. A Response Primary TLV is not required; for most DSO operations the MESSAGE ID field in the DNS message header is sufficient to identify the DSO request message to which a particular response message relates.

Any other TLVs in a DSO response message are Response Additional TLVs, such as the DSO Encryption Padding TLV (Section 7.3). Response Additional TLVs follow the Response Primary TLV(s), if present. Response Additional TLVs are not limited to what is defined in this document. New Response Additional TLVs may be defined in the future. Their definitions will describe when their use is appropriate. Unrecognized Response Additional TLVs are silently ignored, as described in Sections 5.4.5 and 8.2.

The specification for each DSO TLV determines what TLVs are required in a response to a DSO request message using that TLV. If a DSO response is received for an operation where the specification requires that the response carry a particular TLV or TLVs, and the required TLV(s) are not present, then this is a fatal error and the recipient of the defective response message MUST forcibly abort the connection immediately. Similarly, if more than the specified number of instances of a given TLV are present, this is a fatal error and the recipient of the defective response message MUST forcibly abort the connection immediately.

5.4.3. DSO Unidirectional Messages

It is anticipated that most DSO operations will be specified to use DSO request messages, which generate corresponding DSO responses. In some specialized high-traffic use cases, it may be appropriate to specify DSO unidirectional messages. DSO unidirectional messages can be more efficient on the network because they don't generate a stream of corresponding reply messages. Using DSO unidirectional messages can also simplify software in some cases by removing the 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 used as a Primary TLV (i.e., first) in an outgoing DSO request message (i.e., QR=0) states that a message is to be unidirectional, the MESSAGE ID field MUST be set to zero and the receiver MUST NOT generate any response message corresponding to that DSO unidirectional message.

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

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

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

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

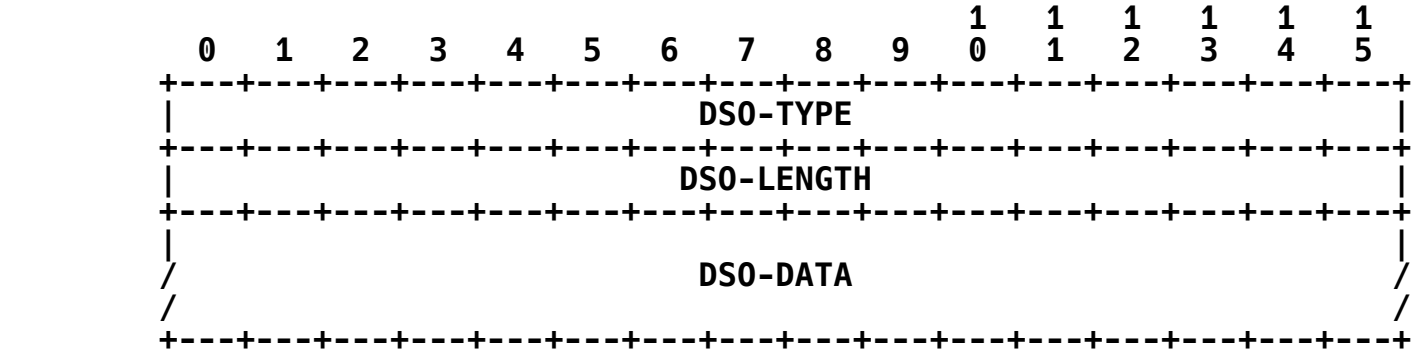
Similarly, after a Discovery Relay client has subscribed to receive inbound multicast DNS (mDNS) [RFC6762] traffic from a Discovery Relay, the subsequent stream of received packets is then sent using DSO unidirectional messages. This is appropriate because the client initiated the message stream by virtue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

5.4.4. TLV Syntax

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

A specification that defines a new TLV must specify whether the DSO-TYPE can be used as a Primary TLV, and whether the DSO-TYPE can be used as an Additional TLV. Some DSO-TYPEs are dual-purpose and can be used as Primary TLVs in some messages, and in other messages as Additional TLVs. The specification for a DSO-TYPE must also state whether, when used as the Primary (i.e., first) TLV in a DSO message (i.e., QR=0), that DSO message is unidirectional, or is a DSO request message that requires a response.

If a DSO request message requires a response, the specification must also state which TLVs, if any, are to be included in the response and how many instances of each of the TLVs are allowed. The Primary TLV may or may not be contained in the response depending on what is specified for that TLV. If multiple instances of the Primary TLV are allowed the specification should clearly describe how they should be processed.



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

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

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

5.4.5. Unrecognized TLVs

If a 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 a matching MESSAGE ID, and RCODE DSOTYPENI. The error response MUST NOT contain a copy of the unrecognized Primary TLV.

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

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

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

5.4.6. EDNS(0) and TSIG

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

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

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

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

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

5.5. Message Handling

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

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

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

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

5.5.1. Delayed Acknowledgement Management

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

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

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

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

5.5.2. MESSAGE ID Namespaces

The namespaces of 16-bit MESSAGE IDs are independent in each direction. This means it is *not* an error for both client and server to send DSO request messages at the same time as each other, using the same MESSAGE ID, in different directions. This simplification is necessary in order for the protocol to be implementable. It would be infeasible to require the client and server to coordinate with each other regarding allocation of new unique MESSAGE IDs. It is also not necessary to require the client and server to coordinate with each other regarding allocation of new unique MESSAGE IDs. The value of the 16-bit MESSAGE ID combined with the identity of the initiator (client or server) is sufficient to unambiguously identify the operation in question. This can be thought of as a 17-bit message identifier space using message identifiers 0x00001-0x0FFFF for client-to-server DSO request messages, and 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 in Section 5.4.1, an initiator MUST NOT reuse a MESSAGE ID that it already has in use for an outstanding DSO request message (unless specified otherwise by the relevant specification for the DSO-TYPE in question). At the very least, this means that a MESSAGE ID can't be reused in a particular direction on a particular DSO Session while the initiator is waiting for a response to a previous DSO request message using that MESSAGE ID on that DSO Session (unless specified otherwise by the relevant specification for the DSO-TYPE in question), and for a long-lived operation, the MESSAGE ID for the operation can't be reused while that operation remains active.

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

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

5.5.3. Error Responses

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

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

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

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

When a DSO unidirectional message type is received (MESSAGE ID field is zero), the receiver should already be expecting this DSO message type. Section 5.4.5 describes the handling of unknown DSO message types. When a DSO unidirectional message of an unexpected type is received, the receiver SHOULD forcibly abort the connection. Whether the connection should be forcibly aborted for other internal errors processing the DSO unidirectional message is implementation dependent according to the severity of the error.

5.6. Responder-Initiated Operation Cancellation

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

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

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

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

The responder performs this selective cancellation by sending a new DSO 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 DSO response message giving the reason for cancellation.

After a DSO 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 DSO 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 canceled operation's MESSAGE ID is now free for reuse.

6. DSO Session Lifecycle and Timers

6.1. DSO Session Initiation

A DSO Session begins as described in Section 5.1.

Once a DSO Session has been created, client or server may initiate as many DNS operations as they wish using the DSO Session.

When an initiator has multiple messages to send, it SHOULD NOT wait for each response before sending the next message.

A responder MUST act on messages in the order they are received, and SHOULD return responses to request messages as they become available. A responder SHOULD NOT delay sending responses for the purpose of delivering responses in the same order that the corresponding requests were received.

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

6.2. DSO Session Timeouts

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

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

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

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

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

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

In the usual case where the inactivity timeout is shorter than the keepalive interval, it is only when a client has a long-lived, low-traffic operation that the keepalive interval comes into play in order to ensure that a sufficient residual amount of traffic is generated to maintain NAT and firewall state, and to assure the 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 a higher CPU load on the server.

6.3. Inactive DSO Sessions

At both servers and clients, the generation or reception of any complete DNS message (including DNS requests, responses, updates, DSO messages, etc.) resets both timers for that DSO Session, with the one exception being 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 [Push] or a Discovery Relay interface subscription [Relay]), an operation is considered "in progress" for as long as the operation is active, i.e., until it is canceled. 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. 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, it does not automatically make that DSO Session "inactive", if it has an active operation that is awaiting events.

6.4. The Inactivity Timeout

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

6.4.1. Closing Inactive DSO Sessions

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

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

If, at any time during the life of the DSO Session, too much time elapses without there being any operation active on the DSO Session, then the server MUST consider the client delinquent and MUST forcibly abort the DSO Session. What is considered "too much time" in this context is five seconds or twice the current inactivity timeout value, whichever is greater. If the inactivity timeout has its default value of 15 seconds, this means that a client will be considered delinquent and disconnected if it has not closed its connection after 30 seconds of inactivity.

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

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

6.4.2. Values for the Inactivity Timeout

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

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

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

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

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

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

6.5. The Keepalive Interval

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

6.5.1. Keepalive Interval Expiry

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

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

6.5.2. Values for the Keepalive Interval

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

It may be appropriate for clients and servers to select different keepalive intervals depending on the type of 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 longer keepalive interval because frequent DSO keepalive traffic is not required.

A public DNS server that is serving primarily residential consumer clients, where it is likely there will be a NAT gateway on the path, may impose a shorter keepalive interval to generate more frequent DSO keepalive traffic.

A smart client may be adaptive to its environment. A client using a private IPv4 address [RFC1918] to communicate with a DNS server at an address outside that IPv4 private address block may conclude that there is likely to be a NAT gateway on the path, and accordingly request a shorter 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. Also, it 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 and 4 minutes [RFC5382] [RFC7857].

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

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

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

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

6.6. Server-Initiated DSO Session Termination

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

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

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

- The server application software or underlying operating system is shutting down or restarting.
- The server application software terminates unexpectedly (perhaps due to a bug that makes it crash, causing the underlying operating system to send a TCP RST).
- 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.
- The client fails to meet its obligation to generate the required DSO keepalive traffic or to close an inactive session by the prescribed time (five seconds or twice the time interval dictated by the server, whichever is greater, as described in Section 6.2).
- The client sends a grossly invalid or malformed request that is indicative of a seriously defective client implementation.
- The server is over capacity and needs to shed some load.

6.6.1. Server-Initiated Retry Delay Message

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

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

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

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

6.6.1.1. Outstanding Operations

At the instant a server chooses to initiate a DSO 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 DSO 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 DSO 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 will now 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 DSO Retry Delay message, the reconnect delay MAY be zero, indicating that the clients SHOULD immediately attempt to re-establish operations.

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

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

6.6.2. Misbehaving Clients

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

6.6.3. Client Reconnection

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

If a particular service instance will only be out of service for a short maintenance period, it should indicate a retry delay value that is 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 promptly after it resumes service.

If a service instance does not want a client to reconnect ever (perhaps the service instance is being decommissioned), it SHOULD set the retry delay to the maximum value 0xFFFFFFFF (2^32-1 milliseconds, approximately 49.7 days). It is not possible to instruct a client to stay away for longer than 49.7 days. If, after 49.7 days, the DNS or other configuration information still indicates that this is the valid service instance for a particular service, then clients MAY attempt to reconnect. In reality, if a client is rebooted or otherwise loses state, it may well attempt to reconnect before 49.7 days elapse, for as long as the DNS or other configuration information continues to indicate that this is the service instance the client should use.

6.6.3.1. Reconnecting after a Forcible Abort

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

6.6.3.2. Reconnecting after an Unexplained Connection Drop

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

6.6.3.3. Probing for Working DSO Support

Once a server has been marked by the client as not supporting DSO, the client SHOULD NOT attempt DSO operations on that server until some time has elapsed. A reasonable minimum would be an hour. Since forcibly aborted connections are the result of a software failure, it's not likely that the problem will be solved in the first hour after it's first encountered. However, by restricting the retry interval to an hour, the client will be able to notice when the problem has been fixed without placing an undue burden on the server.

7. Base TLVs for DNS Stateful Operations

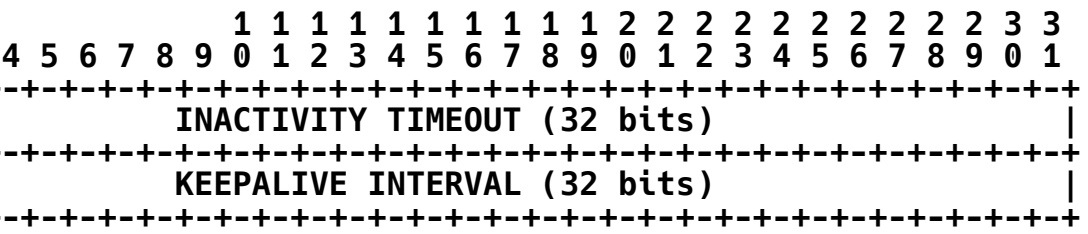
This section describes the three base TLVs for DNS Stateful Operations: Keepalive, Retry Delay, and Encryption Padding.

7.1. Keepalive TLV

The Keepalive TLV (DSO-TYPE=1) performs two functions. Primarily, it establishes the values for the Session Timeouts. Incidentally, it also resets the keepalive timer for the DSO Session, meaning that it can be used as a kind of "no-op" message for the purpose of keeping a session alive. The client will request the desired Session Timeout values and the server will acknowledge with the response values that it requires the client to use.

DSO messages with the Keepalive TLV as the Primary TLV may appear in early data.

The DSO-DATA for the Keepalive TLV is as follows:



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

KEEPALIVE INTERVAL: The keepalive interval for the current DSO Session, specified as a 32-bit unsigned integer, in network (big endian) byte order in units of milliseconds. This is the interval at which a client MUST generate DSO keepalive traffic to maintain connection state. The keepalive interval MUST NOT be less than ten seconds. If the client does not generate the mandated DSO keepalive traffic, then after twice this interval the server will forcibly abort the connection. Since the minimum allowed keepalive interval is ten seconds, the minimum time at which a server will forcibly disconnect a client for failing to generate the mandated DSO keepalive traffic is twenty seconds.

The transmission or reception of DSO Keepalive messages (i.e., messages where the Keepalive TLV is the first TLV) reset only the keepalive timer, not the inactivity timer. The reason for this is that periodic DSO Keepalive messages are sent for the sole purpose of keeping a DSO Session alive when that DSO Session has current or recent non-maintenance activity that warrants keeping that DSO Session alive. Sending DSO keepalive traffic itself is not considered a client activity; it is considered a maintenance activity that is performed in service of other client activities. If DSO keepalive traffic itself were to reset the inactivity timer, then that would create a circular livelock where keepalive traffic would be sent indefinitely to keep a DSO Session alive. In this scenario, the only activity on that DSO Session would be the keepalive traffic keeping the DSO Session alive so that further keepalive traffic can be sent. For a DSO Session to be considered active, it must be carrying something more than just keepalive traffic. This is why merely sending or receiving a DSO Keepalive message does not reset the inactivity timer.

When sent by a client, the DSO Keepalive request message MUST be sent as a DSO request message with a nonzero MESSAGE ID. If a server receives a DSO Keepalive message with a zero MESSAGE ID, then this is a fatal error and the server MUST forcibly abort the connection immediately. The DSO Keepalive request message resets a DSO Session's keepalive timer and, at the same time, communicates to the server the client's requested Session Timeout values. In a server response to a client-initiated DSO Keepalive request message, the Session Timeouts contain the server's chosen values from this point forward in the DSO Session, which the client MUST respect. This is modeled after the DHCP protocol, where the client requests a certain lease lifetime using DHCP option 51 [RFC2132], but the server is the ultimate authority for deciding what lease lifetime is actually granted.

When a client is sending its second and subsequent DSO Keepalive request messages to the server, the client SHOULD continue to request its preferred values each time. This allows flexibility so that if conditions change during the lifetime of a DSO Session, the server can adapt its responses to better fit the client's needs.

Once a DSO Session is in progress (Section 5.1), a DSO Keepalive message MAY be initiated by a server. When sent by a server, the DSO Keepalive message MUST be sent as a DSO unidirectional message with the MESSAGE ID set to zero. The client MUST NOT generate a response to a server-initiated DSO Keepalive message. If a client receives a DSO Keepalive request message with a nonzero MESSAGE ID, then this is a fatal error and the client MUST forcibly abort the connection immediately. The DSO Keepalive unidirectional message from the

server resets a DSO Session's keepalive timer and, at the same time, unilaterally informs the client of the new Session Timeout values to use from this point forward in this DSO Session. No client DSO response to this unilateral declaration is required or allowed.

In DSO Keepalive response messages, exactly one instance of the Keepalive TLV MUST be present and is used only as a Response Primary TLV sent as a reply to a DSO Keepalive request message from the client. A Keepalive TLV MUST NOT be added to other responses as a Response Additional TLV. If the server wishes to update a client's Session Timeout values other than in response to a DSO Keepalive request message from the client, then it does so by sending a DSO Keepalive unidirectional message of its own, as described above.

It is not required that the Keepalive TLV be used in every DSO Session. While many DSO operations will be used in conjunction with a long-lived session state, not all DSO operations require a long-lived session state, and in some cases the default 15-second value for both the inactivity timeout and keepalive interval may be perfectly appropriate. However, note that for clients that implement only the DSO-TYPEs defined in this document, a DSO Keepalive request message is the only way for a client to initiate a DSO Session.

7.1.1. Client Handling of Received Session Timeout Values

When a client receives a response to its client-initiated DSO Keepalive request message, or receives a server-initiated DSO Keepalive unidirectional message, the client has then received Session Timeout values dictated by the server. The two timeout values contained in the Keepalive TLV from the server may each be higher, lower, or the same as the respective Session Timeout values the client previously had for this DSO Session.

In the case of the keepalive timer, the handling of the received value is straightforward. The act of receiving the message containing the DSO Keepalive TLV itself resets the keepalive timer and updates the keepalive interval for the DSO Session. The new keepalive interval indicates the maximum time that may elapse before another message must be sent or received on this DSO Session, if the DSO Session is to remain alive.

In the case of the inactivity timeout, the handling of the received value is a little more subtle, though the meaning of the inactivity timeout remains as specified; it still indicates the maximum permissible time allowed without useful activity on a DSO Session. The act of receiving the message containing the Keepalive TLV does not itself reset the inactivity timer. The time elapsed since the last useful activity on this DSO Session is unaffected by exchange of

DSO Keepalive messages. The new inactivity timeout value in the Keepalive TLV in the received message does update the timeout associated with the running inactivity timer; that becomes the new maximum permissible time without activity on a DSO Session.

- 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.

- 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.

- 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.

- 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 either five seconds or one quarter of the new inactivity timeout, whichever is greater.

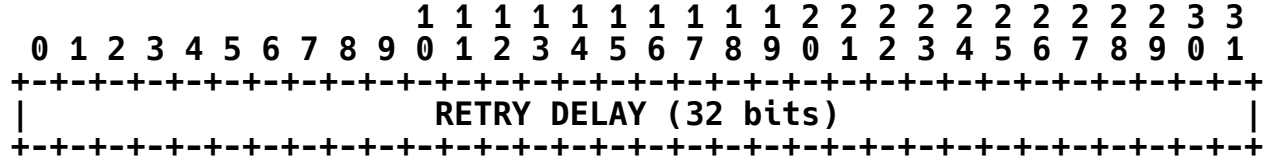
7.1.2. Relationship to edns-tcp-keepalive EDNS(0) Option

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

7.2. Retry Delay TLV

The Retry Delay TLV (DSO-TYPE=2) can be used as a Primary TLV (unidirectional) in a server-to-client message, or as a Response Additional TLV in either direction. DSO messages with a Relay Delay TLV as their Primary TLV are not permitted in early data.

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



RETRY DELAY: A time value, specified as a 32-bit unsigned integer in network (big endian) byte order, in units of milliseconds, within which the initiator MUST NOT retry this operation or retry connecting to this server. Recommendations for the RETRY DELAY value are given in Section 6.6.1.

7.2.1. Retry Delay TLV Used as a Primary TLV

When used as the Primary TLV in a DSO unidirectional message, the Retry Delay TLV is sent from server to client. It is used by a server to instruct a client to close the DSO Session and underlying connection, and not to reconnect for the indicated time interval.

In this case, it applies to the DSO Session as a whole, and the client MUST begin closing the DSO Session as described in Section 6.6.1. The RCODE in the message header SHOULD indicate the principal reason for the termination:

- NOERROR indicates a routine shutdown or restart.
- FORMERR indicates that a client DSO request was too badly malformed for the session to continue.
- SERVFAIL indicates that the server is overloaded due to resource exhaustion and needs to shed load.
- REFUSED indicates that the server has been reconfigured, and at this time it is now unable to perform one or more of the long-lived client operations that were previously being performed on this DSO Session.

- NOTAUTH indicates that the server has been reconfigured and at this time it is now unable to perform one or more of the long-lived client operations that were previously being performed on this DSO Session because it does not have authority over the names in question (for example, a DNS Push Notification server could be reconfigured such that it is no longer accepting DNS Push Notification requests for one or more of the currently subscribed names).

This document specifies only these RCODE values for the DSO Retry Delay message. Servers sending DSO Retry Delay messages SHOULD use one of these values. However, future circumstances may create situations where other RCODE values are appropriate in DSO Retry Delay messages, so clients MUST be prepared to accept DSO Retry Delay messages with any RCODE value.

In some cases, when a server sends a DSO Retry Delay unidirectional message to a client, there may be more than one reason for the server wanting to end the session. Possibly, the configuration could have been changed such that some long-lived client operations can no longer be continued due to policy (REFUSED), and other long-lived client operations can no longer be performed due to the server no longer being authoritative for those names (NOTAUTH). In such cases, the server MAY use any of the applicable RCODE values, or RCODE=NOERROR (routine shutdown or restart).

Note that the selection of RCODE value in a DSO Retry Delay message is not critical since the RCODE value is generally used only for information purposes such as writing to a log file for future human analysis regarding the nature of the disconnection. Generally, clients do not modify their behavior depending on the RCODE value. The RETRY DELAY in the message tells the client how long it should wait before attempting a new connection to this service instance.

For clients that do in some way modify their behavior depending on the RCODE value, they should treat unknown RCODE values the same as RCODE=NOERROR (routine shutdown or restart).

A DSO Retry Delay message (DSO message where the Primary TLV is Retry Delay) from server to client is a DSO unidirectional message; the MESSAGE ID MUST be set to zero in the outgoing message and the client MUST NOT send a response.

A client MUST NOT send a DSO Retry Delay message to a server. If a server receives a DSO message where the Primary TLV is the Retry Delay TLV, this is a fatal error and the server MUST forcibly abort the connection immediately.

7.2.2. Retry Delay TLV Used as a Response Additional TLV

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

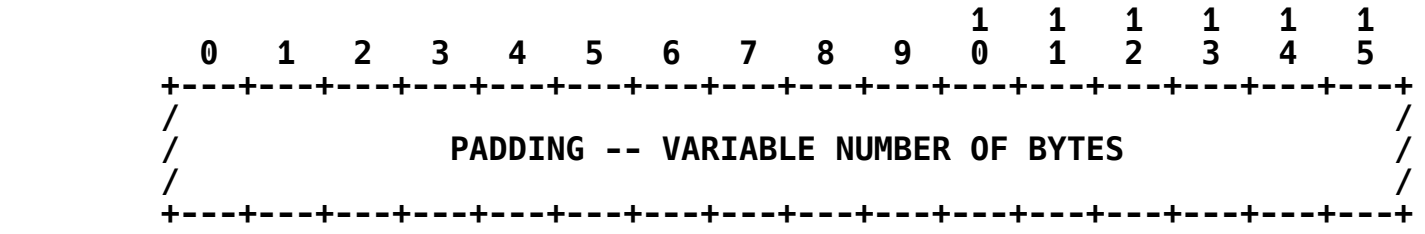
The indicated time interval during which the initiator SHOULD NOT retry applies only to the failed operation, not to the DSO Session as a whole.

Either a client or a server, whichever is acting in the role of the responder for a particular DSO request message, MAY append a Retry Delay TLV to an error response that it sends.

7.3. Encryption Padding TLV

The Encryption Padding TLV (DSO-TYPE=3) can only be used as an Additional or Response Additional TLV. It is only applicable when the DSO Transport layer uses encryption such as TLS.

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



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

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

The length of padding is intentionally not specified in this document and is a function of current best practices with respect to the type and length of data in the preceding TLVs [RFC8467].

8. Summary Highlights

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

8.1. QR Bit and MESSAGE ID

In DSO request messages, the QR bit is 0 and the MESSAGE ID is nonzero.

In DSO response messages, the QR bit is 1 and the MESSAGE ID is nonzero.

In DSO unidirectional messages, the QR bit is 0 and the MESSAGE ID is zero.

The table below illustrates which combinations are legal and how they are interpreted:

	MESSAGE ID zero	MESSAGE ID nonzero
QR=0	DSO unidirectional message	DSO request message
QR=1	Invalid - Fatal Error	DSO response message

8.2. TLV Usage

The table below indicates, for each of the three TLVs defined in this document, whether they are valid in each of ten different contexts.

The first five contexts are DSO requests or DSO unidirectional messages from client to server, and the corresponding responses from server back to client:

- 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.
- C-U - Primary TLV, sent in DSO unidirectional message, from client to server, with zero MESSAGE ID indicating that this request MUST NOT generate response message.
- C-A - Additional TLV, optionally added to a DSO request message or DSO unidirectional message from client to server.
- CRP - Response Primary TLV, included in response message sent back to the client (in response to a client "C-P" request with nonzero MESSAGE ID indicating that a response is required) where the DSO-TYPE of the Response TLV matches the DSO-TYPE of the Primary TLV in the request.
- CRA - Response Additional TLV, included in response message sent back to the client (in response to a client "C-P" request with nonzero MESSAGE ID indicating that a response is required) where the DSO-TYPE of the Response TLV does not match the DSO-TYPE of the Primary TLV in the request.

The second five contexts are their counterparts in the opposite direction: DSO requests or DSO unidirectional messages from server to client, and the corresponding responses from client back to server.

- S-P - Primary TLV, sent in DSO request message, from server to client, with nonzero MESSAGE ID indicating that this request MUST generate response message.
- S-U - Primary TLV, sent in DSO unidirectional message, from server to client, with zero MESSAGE ID indicating that this request MUST NOT generate response message.
- S-A - Additional TLV, optionally added to a DSO request message or DSO unidirectional message from server to client.

- SRP - Response Primary TLV, included in response message sent back to the server (in response to a server "S-P" request with nonzero MESSAGE ID indicating that a response is required) where the DSO-TYPE of the Response TLV matches the DSO-TYPE of the Primary TLV in the request.

- SRA - Response Additional TLV, included in response message sent back to the server (in response to a server "S-P" request with nonzero MESSAGE ID indicating that a response is required) where the DSO-TYPE of the Response TLV does not match the DSO-TYPE of the Primary TLV in the request.

	C-P	C-U	C-A	CRP	CRA	S-P	S-U	S-A	SRP	SRA
KeepAlive	X			X			X			
RetryDelay					X		X			X
Padding			X		X			X		X

Note that some of the columns in this table are currently empty. The table provides a template for future TLV definitions to follow. It is recommended that definitions of future TLVs include a similar table summarizing the contexts where the new TLV is valid.

9. Additional Considerations

9.1. Service Instances

We use the term "service instance" to refer to software running on a host that can receive connections on some set of { IP address, port } tuples. What makes the software an instance is that regardless of which of these tuples the client uses to connect to it, the client is connected to the same software, running on the same logical node (see Section 9.2), and will receive the same answers and the same keying information.

Service instances are identified from the perspective of the client. If the client is configured with { IP address, port } tuples, it has no way to tell if the service offered at one tuple is the same server that is listening on a different tuple. So in this case, the client treats each different tuple as if it references a different service instance.

In some cases, a client is configured with a hostname and a port number. The port number may be given explicitly, along with the hostname. The port number may be omitted, and assumed to have some default value. The hostname and a port number may be learned from the network, as in the case of DNS SRV records. In these cases, the { hostname, port } tuple uniquely identifies the service instance, subject to the usual case-insensitive DNS comparison of names [RFC1034].

It is possible that two hostnames might point to some common IP addresses; this is a configuration anomaly that the client is not obliged to detect. The effect of this could be that after being told to disconnect, the client might reconnect to the same server because it is represented as a different service instance.

Implementations SHOULD NOT resolve hostnames and then perform the process of matching IP address(es) in order to evaluate whether two entities should be determined to be the "same service instance".

9.2. Anycast Considerations

When an anycast service is configured on a particular IP address and port, it must be the case that although there is more than one physical server responding on that IP address, each such server can be treated as equivalent. What we mean by "equivalent" here is that both servers can provide the same service and, where appropriate, the same authentication information, such as PKI certificates, when establishing connections.

If a change in network topology causes packets in a particular TCP connection to be sent to an anycast server instance that does not know about the connection, the new server will automatically terminate the connection with a TCP reset, since it will have no record of the connection, and then the client can reconnect or stop using the connection as appropriate.

If, after the connection is re-established, the client's assumption that it is connected to the same instance is violated in some way, that would be considered an incorrect behavior in this context. It is, however, out of the possible scope for this specification to make specific recommendations in this regard; that would be up to follow-on documents that describe specific uses of DNS Stateful Operations.

9.3. Connection Sharing

As previously specified for DNS-over-TCP [RFC7766]:

> To mitigate the risk of unintentional server overload, DNS clients MUST take care to minimize the number of concurrent TCP connections made to any individual server. It is RECOMMENDED that for any given client/server interaction there SHOULD be no more than one connection for regular queries, one for zone transfers, and one for each protocol that is being used on top of TCP (for example, if the resolver was using TLS). However, it is noted that certain primary/secondary configurations with many busy zones might need to use more than one TCP connection for zone transfers for operational reasons (for example, to support concurrent transfers of multiple zones).

A single server may support multiple services, including DNS Updates [RFC2136], DNS Push Notifications [Push], and other services, for one or more DNS zones. When a client discovers that the target server for several different operations is the same service instance (see Section 9.1), the client SHOULD use a single shared DSO Session for all those operations.

This requirement has two benefits. First, it reduces unnecessary connection load on the DNS server. Second, it avoids the connection startup time that would be spent establishing each new additional connection to the same 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.

9.4. Operational Considerations for Middleboxes

Where an application-layer middlebox (e.g., a DNS proxy, forwarder, or session multiplexer) is in the path, care must be taken to avoid a configuration in which DSO traffic is mishandled. The simplest way to avoid such problems is to avoid using middleboxes. When this is not possible, middleboxes should be evaluated to make sure that they behave correctly.

Correct behavior for middleboxes consists of one of the following:

- The middlebox does not forward DSO messages and responds to DSO messages with a response code other than NOERROR or DSOTYPENI.
- The middlebox acts as a DSO server and follows this specification in establishing connections.
- There is a 1:1 correspondence between incoming and outgoing connections such that when a connection is established to the middlebox, it is guaranteed that exactly one corresponding connection will be established from the middlebox to some DNS resolver, and all incoming messages will be forwarded without modification or reordering. An example of this would be a NAT forwarder or TCP connection optimizer (e.g., for a high-latency connection such as a geosynchronous satellite link).

Middleboxes that do not meet one of the above criteria are very likely to fail in unexpected and difficult-to-diagnose ways. For example, a DNS load balancer might unbundle DNS messages from the incoming TCP stream and forward each message from the stream to a different DNS server. If such a load balancer is in use, and the DNS servers it points to implement DSO and are configured to enable DSO, DSO Session establishment will succeed, but no coherent session will exist between the client and the server. If such a load balancer is pointed at a DNS server that does not implement DSO or is configured not to allow DSO, no such problem will exist, but such a configuration risks unexpected failure if new server software is installed that does implement DSO.

It is of course possible to implement a middlebox that properly supports DSO. It is even possible to implement one that implements DSO with long-lived operations. This can be done either by maintaining a 1:1 correspondence between incoming and outgoing connections, as mentioned above, or by terminating incoming sessions at the middlebox but maintaining state in the middlebox about any long-lived operations that are requested. Specifying this in detail is beyond the scope of this document.

9.5. TCP Delayed Acknowledgement Considerations

Most modern implementations of the Transmission Control Protocol (TCP) include a feature called "Delayed Acknowledgement" [RFC1122].

Without this feature, TCP can be very wasteful on the network. For illustration, consider a simple example like remote login using a very simple TCP implementation that lacks delayed acks. When the user types a keystroke, a data packet is sent. When the data packet arrives at the server, the simple TCP implementation sends an immediate acknowledgement. Mere milliseconds later, the server process reads the one byte of keystroke data, and consequently the simple TCP implementation sends an immediate window update. Mere milliseconds later, the server process generates the character echo and sends this data back in reply. The simple TCP implementation then sends this data packet immediately too. In this case, this simple TCP implementation sends a burst of three packets almost instantaneously (ack, window update, data).

Clearly it would be more efficient if the TCP implementation were to combine the three separate packets into one, and this is what the delayed ack feature enables.

With delayed ack, the TCP implementation waits after receiving a data packet, typically for 200 ms, and then sends its ack if (a) more data packet(s) arrive, (b) the receiving process generates some reply data, or (c) 200 ms elapse without either of the above occurring.

With delayed ack, remote login becomes much more efficient, generating just one packet instead of three for each character echo.

The logic of delayed ack is that the 200 ms delay cannot do any significant harm. If something at the other end were waiting for something, then the receiving process should generate the reply that the thing at the other end is waiting for, and TCP will then immediately send that reply (combined with the ack and window update). And if the receiving process does not in fact generate any reply for this particular message, then by definition the thing at the other end cannot be waiting for anything. Therefore, the 200 ms delay is harmless.

This assumption may be true unless the sender is using Nagle's algorithm, a similar efficiency feature, created to protect the network from poorly written client software that performs many rapid small writes in succession. Nagle's algorithm allows these small writes to be coalesced into larger, less wasteful packets.

Unfortunately, Nagle's algorithm and delayed ack, two valuable efficiency features, can interact badly with each other when used together [NagleDA].

DSO request messages elicit responses; DSO unidirectional messages and DSO response messages do not.

For DSO request messages, which do elicit responses, Nagle's algorithm and delayed ack work as intended.

For DSO messages that do not elicit responses, the delayed ack mechanism causes the ack to be delayed by 200 ms. The 200 ms delay on the ack can in turn cause Nagle's algorithm to prevent the sender from sending any more data for 200 ms until the awaited ack arrives. On an enterprise Gigabit Ethernet (GigE) backbone with sub-millisecond round-trip times, a 200 ms delay is enormous in comparison.

When this issues is raised, there are two solutions that are often offered, neither of them ideal:

1. Disable delayed ack. For DSO messages that elicit no response, removing delayed ack avoids the needless 200 ms delay and sends back an immediate ack that tells Nagle's algorithm that it should immediately grant the sender permission to send its next packet. Unfortunately, for DSO messages that *do* elicit a response, removing delayed ack removes the efficiency gains of combining acks with data, and the responder will now send two or three packets instead of one.

2. Disable Nagle's algorithm. When acks are delayed by the delayed ack algorithm, removing Nagle's algorithm prevents the sender from being blocked from sending its next small packet immediately. Unfortunately, on a network with a higher round-trip time, removing Nagle's algorithm removes the efficiency gains of combining multiple small packets into fewer larger ones, with the goal of limiting the number of small packets in flight at any one time.

The problem here is that with DSO messages that elicit no response, the TCP implementation is stuck waiting, unsure if a response is about to be generated or whether the TCP implementation should go ahead and send an ack and window update.

The solution is networking APIs that allow the receiver to inform the TCP implementation that a received message has been read, processed, and no response for this message will be generated. TCP can then

stop waiting for a response that will never come, and immediately go ahead and send an ack and window update.

For implementations of DSO, disabling delayed ack is NOT RECOMMENDED because of the harm this can do to the network.

For implementations of DSO, disabling Nagle's algorithm is NOT RECOMMENDED because of the harm this can do to the network.

At the time that this document is being prepared for publication, it is known that at least one TCP implementation provides the ability for the recipient of a TCP message to signal that it is not going to send a response, and hence the delayed ack mechanism can stop waiting. Implementations on operating systems where this feature is available SHOULD make use of it.

10. IANA Considerations

10.1. DSO OPCODE Registration

The IANA has assigned the value 6 for DNS Stateful Operations (DSO) in the "DNS OpCodes" registry.

10.2. DSO RCODE Registration

IANA has assigned the value 11 for the DSOTYPENI error code in the "DNS RCODEs" registry. The DSOTYPENI error code ("DSO-TYPE Not Implemented") indicates that the receiver does implement DNS Stateful Operations, but does not implement the specific DSO-TYPE of the Primary TLV in the DSO request message.

10.3. DSO Type Code Registry

The IANA has created the 16-bit "DSO Type Codes" registry, with initial (hexadecimal) values as shown below:

Type	Name	Early Data	Status	Reference
0000	Reserved	NO	Standards Track	RFC 8490
0001	KeepAlive	OK	Standards Track	RFC 8490
0002	RetryDelay	NO	Standards Track	RFC 8490
0003	EncryptionPadding	NA	Standards Track	RFC 8490
0004-003F	Unassigned, reserved for DSO session-management TLVs	NO		
0040-F7FF	Unassigned	NO		
F800-FBFF	Experimental/local use	NO		
FC00-FFFF	Reserved for future expansion	NO		

The meanings of the fields are as follows:

Type: The 16-bit DSO type code.

Name: The human-readable name of the TLV.

Early Data: If OK, this TLV may be sent as early data in a TLS zero round-trip (Section 2.3 of the TLS 1.3 specification [RFC8446]) initial handshake. If NA, the TLV may appear as an Additional TLV in a DSO message that is sent as early data.

Status: RFC status (e.g., "Standards Track") or "External" if not documented in an RFC.

Reference: A stable reference to the document in which this TLV is defined.

Note: 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]. The expert review should validate that the requested type code is specified in a way that conforms to this specification, and that the intended use for the code would not be addressed with an experimental/local assignment.

DSO Type Codes in the "experimental/local" range F800-FBFF may be used as Experimental Use or Private Use values [RFC8126] and may be used freely for development purposes or for other purposes within a single site. No attempt is made to prevent multiple sites from using the same value in different (and incompatible) ways. There is no need for IANA to review such assignments (since IANA does not record them) and assignments are not generally useful for broad interoperability. It is the responsibility of the sites making use of "experimental/local" values to ensure that no conflicts occur within the intended scope of use.

Any document defining a new TLV that lists a value of "OK" in the Early Data column must include a threat analysis for the use of the TLV in the case of TLS zero round-trip. See Section 11.1 for details.

11. Security Considerations

If this mechanism is to be used with DNS-over-TLS, then these messages are subject to the same constraints as any other DNS-over-TLS messages and MUST NOT be sent in the clear before the TLS session is established.

The data field of the "Encryption Padding" TLV could be used as a covert channel.

When designing new DSO TLVs, the potential for data in the TLV to be used as a tracking identifier should be taken into consideration and should be avoided when not required.

When used without TLS or similar cryptographic protection, a malicious entity may be able to inject a malicious unidirectional DSO Retry Delay message into the data stream, specifying an unreasonably large RETRY DELAY, causing a denial-of-service attack against the client.

The establishment of DSO Sessions has an impact on the number of open TCP connections on a DNS server. Additional resources may be used on the server as a result. However, because the server can limit the number of DSO Sessions established and can also close existing DSO Sessions as needed, denial of service or resource exhaustion should not be a concern.

11.1. TLS Zero Round-Trip Considerations

DSO permits zero round-trip operation using TCP Fast Open with TLS 1.3 [RFC8446] early data to reduce or eliminate round trips in session establishment. TCP Fast Open is only permitted in combination with TLS 1.3 early data. In the rest of this section, we refer to TLS 1.3 early data in a TLS zero round-trip initial handshake message, regardless of whether or not it is included in a TCP SYN packet with early data using the TCP Fast Open option, as "early data."

A DSO message may or may not be permitted to be sent as early data. The definition for each TLV that can be used as a Primary TLV is required to state whether or not that TLV is permitted as early data. Only response-requiring messages are ever permitted as early data, and only clients are permitted to send a DSO message as early data unless there is an implicit DSO session (see Section 5.1).

For DSO messages that are permitted as early data, a client MAY include one or more such messages as early data without having to wait for a DSO response to the first DSO request message to confirm successful establishment of a DSO Session.

However, unless there is an implicit DSO session, a client MUST NOT send DSO unidirectional messages until after a DSO Session has been mutually established.

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

Caution must be taken to ensure that DSO messages sent as early data are idempotent or are otherwise immune to any problems that could result from the inadvertent replay that can occur with zero round-trip operation.

It would be possible to add a TLV that requires the server to do some significant work and send that to the server as initial data in a TCP SYN packet. A flood of such packets could be used as a DoS attack on the server. None of the TLVs defined here have this property.

If a new TLV is specified that does have this property, that TLV must be specified as not permitted in zero round-trip messages. This prevents work from being done until a round-trip has occurred from the server to the client to verify that the source address of the packet is reachable.

Documents that define new TLVs must state whether each new TLV may be sent as early data. Such documents must include a threat analysis in the security considerations section for each TLV defined in the document that may be sent as early data. This threat analysis should be done based on the advice given in Sections 2.3, 8, and Appendix E.5 of the TLS 1.3 specification [RFC8446].

12. References

12.1. Normative References

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

[RFC1035] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, DOI 10.17487/RFC1035, November 1987, <https://www.rfc-editor.org/info/rfc1035>.

[RFC1918] Rekhter, Y., Moskowitz, B., Karrenberg, D., de Groot, G., and E. Lear, "Address Allocation for Private Internets", BCP 5, RFC 1918, DOI 10.17487/RFC1918, February 1996, <https://www.rfc-editor.org/info/rfc1918>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC2136] Vixie, P., Ed., Thomson, S., Rekhter, Y., and J. Bound, "Dynamic Updates in the Domain Name System (DNS UPDATE)", RFC 2136, DOI 10.17487/RFC2136, April 1997, <https://www.rfc-editor.org/info/rfc2136>.

[RFC6891] Damas, J., Graff, M., and P. Vixie, "Extension Mechanisms for DNS (EDNS(0))", STD 75, RFC 6891, DOI 10.17487/RFC6891, April 2013, <https://www.rfc-editor.org/info/rfc6891>.

[RFC7766] Dickinson, J., Dickinson, S., Bellis, R., Mankin, A., and D. Wessels, "DNS Transport over TCP - Implementation Requirements", RFC 7766, DOI 10.17487/RFC7766, March 2016, <https://www.rfc-editor.org/info/rfc7766>.

[RFC7830] Mayrhofer, A., "The EDNS(0) Padding Option", RFC 7830, DOI 10.17487/RFC7830, May 2016, <https://www.rfc-editor.org/info/rfc7830>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/info/rfc8126>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

12.2. Informative References

[Fail] Andrews, M. and R. Bellis, "A Common Operational Problem in DNS Servers - Failure To Communicate", Work in Progress, draft-ietf-dnsop-no-response-issue-13, February 2019.

[NagleDA] Cheshire, S., "TCP Performance problems caused by interaction between Nagle's Algorithm and Delayed ACK", May 2005, <http://www.stuartcheshire.org/papers/nagledelayedack/>.

[Push] Pusateri, T. and S. Cheshire, "DNS Push Notifications", Work in Progress, draft-ietf-dnssd-push-18, March 2019.

[Relay] Lemon, T. and S. Cheshire, "Multicast DNS Discovery Relay", Work in Progress, draft-ietf-dnssd-mdns-relay-02, March 2019.

[RFC1122] Braden, R., Ed., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, DOI 10.17487/RFC1122, October 1989, <https://www.rfc-editor.org/info/rfc1122>.

[RFC2132] Alexander, S. and R. Droms, "DHCP Options and BOOTP Vendor Extensions", RFC 2132, DOI 10.17487/RFC2132, March 1997, <https://www.rfc-editor.org/info/rfc2132>.

[RFC5382] Guha, S., Ed., Biswas, K., Ford, B., Sivakumar, S., and P. Srisuresh, "NAT Behavioral Requirements for TCP", BCP 142, RFC 5382, DOI 10.17487/RFC5382, October 2008, <https://www.rfc-editor.org/info/rfc5382>.

[RFC6762] Cheshire, S. and M. Krochmal, "Multicast DNS", RFC 6762, DOI 10.17487/RFC6762, February 2013, <https://www.rfc-editor.org/info/rfc6762>.

[RFC6763] Cheshire, S. and M. Krochmal, "DNS-Based Service Discovery", RFC 6763, DOI 10.17487/RFC6763, February 2013, <https://www.rfc-editor.org/info/rfc6763>.

[RFC7413] Cheng, Y., Chu, J., Radhakrishnan, S., and A. Jain, "TCP Fast Open", RFC 7413, DOI 10.17487/RFC7413, December 2014, <https://www.rfc-editor.org/info/rfc7413>.

[RFC7828] Wouters, P., Abley, J., Dickinson, S., and R. Bellis, "The edns-tcp-keepalive EDNS0 Option", RFC 7828, DOI 10.17487/RFC7828, April 2016, <https://www.rfc-editor.org/info/rfc7828>.

[RFC7857] Penno, R., Perreault, S., Boucadair, M., Ed., Sivakumar, S., and K. Naito, "Updates to Network Address Translation (NAT) Behavioral Requirements", BCP 127, RFC 7857, DOI 10.17487/RFC7857, April 2016, <https://www.rfc-editor.org/info/rfc7857>.

[RFC7858] Hu, Z., Zhu, L., Heidemann, J., Mankin, A., Wessels, D., and P. Hoffman, "Specification for DNS over Transport Layer Security (TLS)", RFC 7858, DOI 10.17487/RFC7858, May 2016, <https://www.rfc-editor.org/info/rfc7858>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

[RFC8467] Mayrhofer, A., "Padding Policies for Extension Mechanisms for DNS (EDNS(0))", RFC 8467, DOI 10.17487/RFC8467, October 2018, <https://www.rfc-editor.org/info/rfc8467>.

[RFC8484] Hoffman, P. and P. McManus, "DNS Queries over HTTPS (DoH)", RFC 8484, DOI 10.17487/RFC8484, October 2018, <https://www.rfc-editor.org/info/rfc8484>.

Acknowledgements

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

Authors' Addresses

Ray Bellis
Internet Systems Consortium, Inc.
950 Charter Street
Redwood City, CA 94063
United States of America

Phone: +1 (650) 423-1200
Email: ray@isc.org

Stuart Cheshire
Apple Inc.
One Apple Park Way
Cupertino, CA 95014
United States of America

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

John Dickinson
Sinodun Internet Technologies
Magadalen Centre
Oxford Science Park
Oxford OX4 4GA
United Kingdom

Email: jad@sinodun.com

Sara Dickinson
Sinodun Internet Technologies
Magadalen Centre
Oxford Science Park
Oxford OX4 4GA
United Kingdom

Email: sara@sinodun.com

Ted Lemon
Nibbhaya Consulting
P.O. Box 958
Brattleboro, VT 05302-0958
United States of America

Email: mellon@fugue.com

Tom Pusateri
Unaffiliated
Raleigh, NC 27608
United States of America

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