DNSOP Working Group
Internet-Draft
Updates: RFC 7766 (if approved)
Intended status: Standards Track
Expires: September 14, 2017

R. Bellis
ISC
S. Cheshire
Apple Inc.
J. Dickinson
S. Dickinson
Sinodun
A. Mankin
Salesforce
T. Pusateri
Unaffiliated

March 13, 2017

DNS Session Signaling
draft-ietf-dnsop-session-signal-02

Abstract

The EDNS(0) Extension Mechanism for DNS is explicitly defined to only have "per-message" semantics. This document defines a new Session Signaling Opcode used to communicate persistent "per-session" operations, expressed using type-length-value (TLV) syntax, and defines an initial set of TLVs used to manage session timeouts and termination.

Status of This Memo

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

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

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

This Internet-Draft will expire on September 14, 2017.

Copyright Notice

Table of Contents

1. Introduction

The use of transports for DNS other than UDP is being increasingly specified, for example, DNS over TCP [RFC1035][RFC7766] and DNS over TLS [RFC7858]. Such transports can offer persistent, long-lived sessions and therefore when using them for transporting DNS messages it is of benefit to have a mechanism that can establish parameters associated with those sessions, such as timeouts. In such situations it is also advantageous to support server initiated messages.

The existing EDNS(0) Extension Mechanism for DNS [RFC6891] is explicitly defined to only have "per-message" semantics. Whilst EDNS(0) has been used to signal at least one session related

parameter (the EDNS(0) TCP KeepAlive option [RFC7828]) the result is less than optimal due to the restrictions imposed by the EDNS(0) semantics and the lack of server-initiated signalling. This document defines a new Session Signaling Opcode used to carry persistent "per-session" operations, expressed using type-length-value (TLV) syntax, and defines an initial set of TLVs used to manage session timeouts and termination.

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 RR. The specifications for each individual option need to define how each different option is to be acknowledged, if necessary.

With Session Signaling, in contrast, there is no compelling motivation to pack multiple operations into a single message for efficiency reasons. Each Session Signaling operation is communicated in its own separate DNS message, and the transport protocol can take care of packing separate DNS messages into a single IP packet if appropriate. For example, TCP can pack multiple small DNS messages into a single TCP segment. The RCODE in each response message indicates the success or failure of the operation in question.

It should be noted that the message format for Session Signaling operations (see Section 3.3) differs from the traditional DNS packet format used for standard queries and responses. The standard twelve-octet header is used, but the four count fields (QDCOUNT, ANCOUNT, NSCOUNT, ARCOUNT) are set to zero and the corresponding sections are empty. The actual data pertaining to Session Signaling operations is appended to the end of the DNS message, following the four (empty) data sections. When displayed using today’s packet analyser tools that have not been updated to recognize the DNS Session Signaling format, this will result in the Session Signaling data being displayed as unknown additional data after the end of the DNS message. It is likely that future updates to these tools will add the ability to recognize, decode, and display the Session Signaling data.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in "Key words for use in RFCs to Indicate Requirement Levels" [RFC2119].

The term "connection" means a bidirectional stream of reliable, in-order messages, such as provided by using DNS over TCP [RFC1035][RFC7766] or DNS over TLS [RFC7858].

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

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

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

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

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

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

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

Session Signaling operations are expressed using type-length-value (TLV) syntax.

A "Session Signaling Operation TLV" specifies the operation to be performed. A Session Signaling request message MUST contain exactly one Operation TLV. Depending on the operation, the corresponding Session Signaling response message MAY contain no Operation TLV, or it may contain a single corresponding response Operation TLV, with the same SSOP-TYPE as in the request message.

A "Session Signaling Modifier TLV" specifies additional parameters relating to the operation. Immediately following the Operation TLV, if present, a Session Signaling message MAY contain one or more Modifier TLVs.

The first TLV in a Session Signaling request message (and its counterpart in the corresponding Session Signaling response message, if present) is the Operation TLV. Any subsequent TLVs after this initial Operation TLV (if present) are Modifier TLVs.

If a Session Signaling request is received containing an unrecognized Operation TLV then an error response with RCODE SSOPNOTIMP (tentatively 11) is returned.

If a Session Signaling message (request or response) is received containing one or more unrecognized Modifier TLVs, the unrecognized Modifier TLVs are silently ignored.

3. Protocol Details

Session Signaling messages MUST only be carried in protocols and in environments where a session may be established according to the definition above. Standard DNS over TCP [RFC1035][RFC7766], and DNS over TLS [RFC7858] are suitable protocols. DNS over plain UDP is not appropriate since it fails on the requirement for in-order message delivery, and, in the presence of NAT gateways and firewalls with short UDP timeouts, it fails to provide a persistent bi-directional communication channel unless an excessive amount of keepalive traffic is used.

Session Signaling messages relate only to the specific session in which they are being carried. Where an application-layer middle box (e.g., a DNS proxy, forwarder, or session multiplexer) is in the path the middle box MUST NOT blindly forward the message in either direction. This does not preclude the use of these messages in the presence of an IP-layer middle box such as a NAT that rewrites IP-layer and/or transport-layer headers, but otherwise preserves the effect of a single session.

A client MAY attempt to initiate Session Signaling messages at any time on a connection; receiving a NOTIMP response in reply indicates that the server does not implement Session Signaling, and the client SHOULD NOT issue further Session Signaling messages on that connection.

A server SHOULD NOT initiate Session Signaling messages until a client-initiated Session Signaling message is received first, unless in an environment where it is known in advance by other means that

the client supports Session Signaling. This requirement is to ensure that the clients that do not support Session Signaling do not receive unsolicited inbound Session Signaling messages that they would not know how to handle.

3.1. Session Lifecycle and Timers

A session begins when a client makes a new connection to a server.

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

Two timer values are associated with a session: the idle timeout, and the keepalive interval. On a new session, before any explicit Session Signaling Keepalive message exchange, the default value for both timers is 15 seconds.

At both servers and clients, the generation or reception of any complete DNS message, including DNS requests, responses, updates, or Session Signaling messages, resets both timers for that session [RFC7766], with the exception that a Session Signaling Keepalive message resets only the keepalive interval timer, not the idle timeout timer.

In addition, for as long as the client has an outstanding operation in progress, the idle timeout timer remains fixed at zero, and an idle 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 idle timeout timer is set to zero upon transmission of a request and remains at zero until reception of the corresponding response. At the server, the idle timeout timer is set to zero upon reception of a request and remains at zero until transmission of the corresponding response.

For long-lived DNS operations like Push Notification subscriptions [I-D.ietf-dnssd-push], an operation is considered in progress for as long as the subscription is active, until it is cancelled. This means that a session can exist, with a Push Notification subscription active, with no messages flowing in either direction, for far longer than the idle timeout, and this is not an error. This is why there are two separate timers: the idle timeout, and the keepalive

interval. Just because a session has no traffic for an extended period of time does not automatically make that session "idle", if it has an active Push Notification subscription that is awaiting notification events.

The first timer value, the idle timeout, is the maximum time for which a client may speculatively keep a session open in the expectation that it may have future requests to send to that server.

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

The second timer value, the keepalive interval, is the maximum permitted interval between client messages to the server if the client wishes to keep the session alive.

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 attempt re-connection if appropriate.

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

For the idle timeout value, lower values result in more frequent session teardown and re-establishment. Higher values result in lower traffic and CPU load on the server, but a larger memory burden to maintain state for idle sessions.

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

The two timer values are independent. The idle timeout may be lower, the same, or higher than the keepalive interval, though in most cases

the idle timeout is expected to be shorter than the keepalive interval.

A shorter idle timeout with a longer keepalive interval signals to the client that it should not speculatively keep idle sessions open for very long for no reason, but when it does have an active reason to keep a session open, it doesn't need to be sending an agressive level of keepalive traffic. Only when the client has a very long-lived low-traffic operation outstanding like a Push Notification subscription, does the keepalive interval timer come into play, to ensure that a sufficient residual amount of traffic is generated to maintain NAT and firewall state.

A longer idle timeout with a shorter keepalive interval signals to the client that it may speculatively keep idle sessions open for a long time, but it should be sending a lot of keepalive traffic on those idle sessions. This configuration is expected to be less common.

If, at any time during the life of the session, the idle timeout value (i.e., 15 seconds by default) elapses without there being any operation active on the session, the client MUST gracefully close the connection with a TCP FIN (or equivalent for other protocols).

If, at any time during the life of the session, twice the idle timeout value (i.e., 30 seconds by default) elapses without there being any operation active on the session, the server SHOULD consider the client delinquent, and forcibly abort the session. For sessions over TCP (or over TLS over TCP), to avoid the burden of having a connection in TIME-WAIT state, instead of closing the connection gracefully with a TCP FIN the server SHOULD abort the connection with a TCP RST (or equivalent for other protocols). (In the BSD Sockets API this is achieved by setting the SO_LINGER option to zero before closing the socket.)

In this context, an operation being active on a session includes a query waiting for a response, an update waiting for a response, or an outstanding Push Notification subscription [I-D.ietf-dnssd-push], but not a Session Signaling Keepalive message exchange itself. A Session Signaling Keepalive message exchange resets only the keepalive interval timer, not the idle timeout timer.

If, at any time during the life of the session, the keepalive interval value (i.e., 15 seconds by default) elapses without any DNS messages being sent or received on a session, the client MUST take action to keep the session alive. To keep the session alive the client MUST send a Session Signaling Keepalive message (see

Section 4). A Session Signaling Keepalive message exchange resets only the keepalive interval timer, not the idle timeout timer.

If a client disconnects from the network abruptly, without cleanly closing its session, leaving long-lived outstanding operations like Push Notification subscriptions uncanceled, the server learns of this after failing to receive the required keepalive traffic from that client. If, at any time during the life of the session, twice the keepalive interval value (i.e., 30 seconds by default) elapses without any DNS messages being sent or received on a session, the server SHOULD consider the client delinquent, and forcibly abort the connection with a TCP RST (or equivalent for other protocols).

If the client wishes to keep an idle session open for longer than the default duration without having to send traffic every 15 seconds, then it uses the Session Signaling Keepalive message to request longer timeout values, as described in Section 4.

3.1.1. Client-Initiated Termination

A client is NOT required to wait until the idle-timeout timer expires before closing a session. A client MAY close a session at any time, at the client's discretion. If a client determines that it has no current or reasonably anticipated future need for an idle session, then the client SHOULD close that connection.

Upon receiving an error response from the server, a client SHOULD NOT automatically close the session. An error relating to one particular operation on a session does not necessarily imply that all other operations on that 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 close the session, based on the server's determination of whether the error condition pertains to this particular operation, or would also apply to any subsequent operations. If the server does not close the session then the client SHOULD continue to use that session for subsequent operations.

3.1.2. Server-Initiated Termination

After sending an error response to a client, the server MAY close the session, or may allow the 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 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

close the session by sending a Retry Delay request message, as described in Section 5.

There may be rare cases where a server is overloaded and wishes to shed load. If the server handles this by simply closing connections, the likely behaviour of clients is to detect this as a network failure, and reconnect.

To avoid this reconnection implosion, in this situation the server also sends a Retry Delay request message, with an RCODE of SERVFAIL, to inform the client of the overload situation.

After sending a Retry Delay request message, the server MUST NOT send any further messages on that session.

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

A Retry Delay request message MUST NOT be initiated by a client. If a server receives a Retry Delay request message this is an error and the server MUST immediately terminate the connection with a TCP RST (or equivalent for other protocols).

Upon receipt of a Retry Delay request from the server, the client MUST make note of the reconnect delay for this server, and then immediately close the connection. This is to place the burden of TCP’s TIME-WAIT state on the client.

After sending the Retry Delay request 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 abort the connection with a TCP RST (or equivalent for other protocols).

In the case where some, but not all, of the existing operations on a 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 sessions en masse with a REFUSED (5) RCODE, 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.

In the case where a server is terminating a large number of 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).

Apart from the cases described above, a server MUST NOT close a session with a client, except in extraordinary error conditions. Closing the session is the client's responsibility, to be done at the client's discretion, when it so chooses. A server only closes a session under exceptional circumstances, such as when the server application software or underlying operating system is restarting, the server application terminated unexpectedly (perhaps due to a bug that makes it crash), or the server is undergoing maintenance procedures. When possible, a server SHOULD send a Retry Delay message informing the client of the reason for the session being closed, and allow the client five seconds to receive it before the server resorts to forcibly aborting the connection.

After a session is closed by the server, the client SHOULD try to reconnect, to that server, or to another suitable server, if more than one is available. If reconnecting to the same server, the client MUST respect the indicated delay before attempting to reconnect.

If a server is low on resources it MAY simply terminate a client connection with a TCP RST (or equivalent for other protocols). However, the likely behaviour of the client may be simply to reconnect immediately, putting more burden on the server. Therefore, a server SHOULD instead choose to shed client load by sending a Retry Delay message, as described above. Upon reception of the Termination TLV the client is expected to close the session, and if it does not then the server will abort the session five seconds later.

3.2. Connection Sharing

A client that supports Session Signaling SHOULD NOT make multiple connections to the same DNS server.

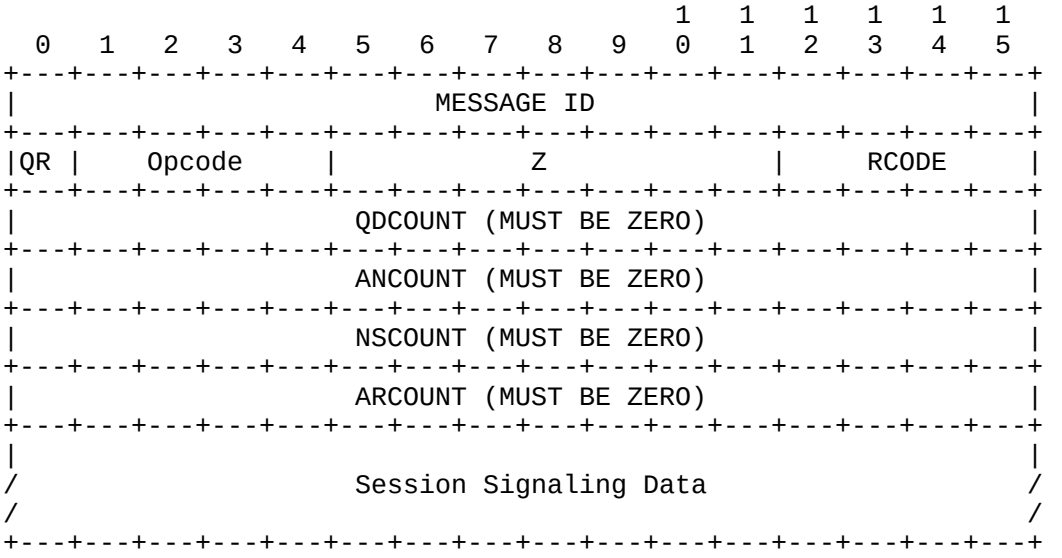
A single server may support multiple services, including DNS Updates [RFC2136], DNS Push Notifications [I-D.ietf-dnssd-push], and other

services, for one or more DNS zones. When a client discovers that the target server for several different operations is the same target hostname and port, the client SHOULD use a single shared session for all those operations. A client SHOULD NOT open multiple connections to the same target host and port just because the names being operated on are different or happen to fall within different zones. This is to reduce unnecessary connection load on the DNS server.

However, server implementers and operators should be aware that connection sharing may not be possible in all cases. A single client 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.

3.3. Message Format

A Session Signaling message begins with the standard twelve-octet DNS message header [RFC1035] with the Opcode field set to the Session Signaling Opcode (tentatively 6). However, unlike standard DNS messages, the question section, answer section, authority records section and additional records sections are all empty. The corresponding count fields (QDCOUNT, ANCOUNT, NSCOUNT, ARCOUNT) MUST be set to zero on transmission. If a Session Signaling message is received where any of the count fields are not zero, then data in the corresponding section MUST be silently skipped by the receiver (unless specified otherwise by a future update to this specification). The skipped data is silently ignored. Any skipped data in a Session Signaling request is discarded, and not copied to the corresponding sections in the Session Signaling response.



In a request the DNS Header QR bit MUST be zero. If the QR bit MUST is not zero the message is not a request.

In a response the DNS Header QR bit MUST be one. If the QR bit is not one the message is not a response.

In a request the MESSAGE ID field MUST be set to a unique value, that the initiator is not using for any other active operation on this connection. For the purposes here, a MESSAGE ID is in use on this connection if the initiator has used it in a request for which it has not yet received a response, or if if the client has used it for a subscription which it has not yet cancelled [I-D.ietf-dnssd-push].

In a response the MESSAGE ID field contain a copy of the value of the MESSAGE ID field in the request being responded to.

The DNS Header Opcode field holds the Session Signaling Opcode value (tentatively 6).

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

In a request message (QR=0) the RCODE is generally set to zero on transmission, and silently ignored on reception, except where specified otherwise (for example, the Retry Delay operation, where the RCODE indicates the reason for termination).

The standard twelve-octet DNS message header and the four (usually) empty sections are followed by at most one Session Signaling Operation TLV. The (optional) Operation TLV may be followed by one or more Modifier TLVs, such as the Retry Delay TLV (0), which, in error responses, indicates the time interval during which the client SHOULD NOT re-attempt a failed operation.

Future specifications may define additional Modifier TLVs.

A Session Signaling message MUST contain at most one Operation TLV. In all cases a Session Signaling request message MUST contain exactly one Operation TLV, indicating the operation to be performed. In some cases a Session Signaling response message MAY contain no Operation TLV, because it is simply a response to a previous request message, and the message ID in the header is sufficient to identify the request in question. The specification for each Session Signaling operation type determines whether a response for that operation type is required to carry the Operation TLV.

If a Session Signaling request is received containing an unrecognized Operation TLV, the receiver MUST send a response with matching MESSAGE ID, and RCODE SSOPNOTIMP (tentatively 11). The response MUST NOT contain an Operation TLV.

If a Session Signaling response is received for an operation which requires that the response carry an Operation TLV, and the required Operation TLV is not the first Session Signaling TLV in the response message, then this is a fatal error and the recipient of the defective response message MUST immediately terminate the connection with a TCP RST (or equivalent for other protocols).

If a Session Signaling message (request or response) is received containing one or more unrecognized Modifier TLVs, the unrecognized Modifier TLVs MUST be silently ignored, and the remainder of the message is interpreted and handled as if the unrecognized parts were not present.

Since the ARCOUNT field MUST be zero, a Session Signaling message MUST NOT contain an EDNS(0) option in the additional records section. If functionality provided by current or future EDNS(0) options is desired for Session Signaling messages, a Session Signaling Operation TLV or Modifier TLV needs to be defined to carry the necessary information.

For example, the EDNS(0) Padding Option used for security purposes [RFC7830] is not permitted in a Session Signaling message, so if message padding is desired for Session Signaling messages, a Session Signaling Modifier TLV needs to be defined to perform this function.

Similarly, a Session Signaling message MUST NOT contain a TSIG record. A TSIG record in a conventional DNS message is added as the last record in the additional records section, and carries a signature computed over the preceding message content. Since Session Signaling data appears after the additional records section, it would not be included in the signature calculation. If use of signatures with Session Signaling messages becomes necessary in the future, an explicit Session Signaling Modifier TLV needs to be defined to perform this function.

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

This specification explicitly prohibits use of the EDNS(0) TCP Keepalive Option [RFC7828] in _any_ messages sent on a Session Signaling session (because it duplicates the functionality provided by the Session Signaling Keepalive operation), but messages may contain other EDNS(0) options as appropriate.

3.4. Message Handling

On a session between a client and server that support Session Signaling, once the client has sent at least one Session Signaling message (or it is known in advance by other means that the client supports Session Signaling) either end may unilaterally send Session Signaling messages at any time, and therefore either client or server

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

Every Session Signaling request message (QR=0) MUST elicit a response (QR=1), which MUST have the same MESSAGE ID in the DNS message header as in the corresponding request. Session Signaling request messages sent by the client elicit a response from the server, and Session Signaling request messages sent by the server elicit a response from the client. With most TCP implementations, the TCP data acknowledgement (generated because data has been received by TCP), the TCP window update (generated because TCP has delivered that data to the receiving software) and the DNS Session Signaling response (generated by the receiving software itself) are all combined into a single packet, so in practice the requirement that every Session Signaling request message MUST elicit a Session Signaling response incurs minimal extra cost on the network. Requiring that every request elicit a corresponding response also avoids performance problems caused by interaction between Nagle's Algorithm and Delayed Ack [NagleDA].

The namespaces of 16-bit MESSAGE IDs are disjoint in each direction. For example, it is _not_ an error for both client and server to send a request message with the same ID. In effect, the 16-bit MESSAGE ID combined with the identity of the initiator (client or server) serves as a 17-bit unique identifier for a particular operation on a session.

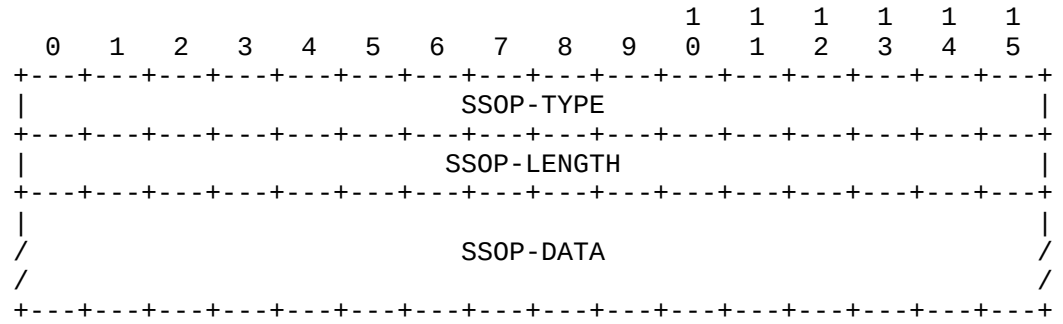
An initiator MUST NOT reuse a MESSAGE ID that is already in use for an outstanding request, unless specified otherwise by the relevant specification for the Session Signaling operation in question. At the very least, this means that a MESSAGE ID MUST NOT be reused in a particular direction on a particular session while the initiator is waiting for a response to a previous request on that session, unless specified otherwise by the relevant specification for the Session Signaling operation in question. (For a long-lived operation, such as a DNS Push Notification subscription [I-D.ietf-dnssd-push] the MESSAGE ID for the operation MUST NOT be reused for a new subscription as long as the existing subscription using that MESSAGE ID remains active.)

If a client or server receives a response (QR=1) where the MESSAGE ID does not match any of its outstanding operations, this is a fatal error and it MUST immediately terminate the connection with a TCP RST (or equivalent for other protocols).

The RCODE value in a response may be one of the following values:

Code	Mnemonic	Description
0	NOERROR	Operation processed successfully
1	FORMERR	Format error
4	NOTIMP	Session Signaling not supported
5	REFUSED	Operation declined for policy reasons
11	SSOPNOTIMP	Session Signaling operation Type Code not supported

3.5. TLV Format



SSOP-TYPE: A 16 bit field in network order giving the type of the current Session Signaling TLV per the IANA DNS Session Signaling Type Codes Registry.

SSOP-LENGTH: A 16 bit field in network order giving the size in octets of SSOP-DATA.

SSOP-DATA: Type-code specific.

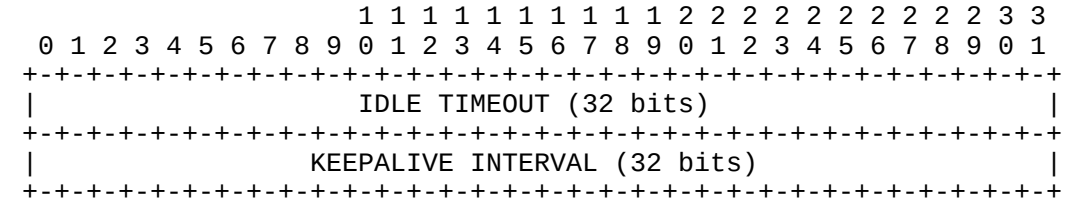
4. Keepalive TLV

The Keepalive TLV (1) performs three functions. When sent by a client, it resets a session's keepalive timer, and at the same time requests what the idle timeout and keepalive interval should be from this point forward in the session.

Once the client has sent at least one Session Signaling message (or it is known in advance by other means that the client supports Session Signaling) the Keepalive TLV also MAY be initiated by a server. When sent by a server, it resets a session's keepalive timer, and unilaterally informs the client of the new idle timeout and keepalive interval to use from this point forward in this session.

It is not required that the Keepalive TLV be used in every session. While many Session Signaling operations (such as DNS Push Notifications [I-D.ietf-dnssd-push]) will be used in conjunction with a long-lived session, not all Session Signaling operations require a long-lived session, and in some cases the default 15-second value for both idle timeout and keepalive interval may be perfectly appropriate.

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



IDLE TIMEOUT: the idle timeout for the current session, specified as a 32 bit word in network (big endian) order in units of milliseconds. This is the timeout at which the client MUST close an idle session. If the client does not gracefully close an idle session then after twice this interval the server will forcibly terminate the connection with a TCP RST (or equivalent for other protocols).

KEEPALIVE INTERVAL: the idle timeout for the current session, specified as a 32-bit word, in network (big endian) order, in units of milliseconds. This is the interval at which a client MUST generate keepalive traffic to maintain connection state. If the client does not generate the necessary keepalive traffic then after twice this interval the server will forcibly terminate the connection with a TCP RST (or equivalent for other protocols).

In a client-initiated Session Signaling Keepalive message, the idle timeout and keepalive interval contain the client's requested values. In a server response to a client-initiated message, the idle timeout and keepalive interval contain the server's chosen values, 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.

In a server-initiated Session Signaling Keepalive message, the idle timeout and keepalive interval unilaterally inform the client of the new values from this point forward in this session. The client MUST generate a response to the server-initiated Session Signaling Keepalive message. The Message ID in the response message MUST match the ID from the server-initiated Session Signaling Keepalive message, and the response message MUST NOT contain any Operation TLV.

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

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

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

A smart client may be adaptive to its environment. A client using a private IPv4 address [RFC1918] to communicate with a DNS server at an address that is not in the same IPv4 private address block, may conclude that there is likely to be a NAT gateway on the path, and accordingly request a lower keepalive interval.

For environments where there is a NAT gateway or firewalls on the path, it is RECOMMENDED that clients request, and servers grant, a keepalive interval of 15 minutes. In other environments it is RECOMMENDED that clients request, and servers grant, a keepalive interval of 60 minutes.

Note that the lower the keepalive interval value, the higher the load on client and server. For example, an keepalive interval value of 100ms would result in a continuous stream of at least ten messages per second, in both directions, to keep the session alive. And, in this extreme example, a single packet loss and retransmission over a

long path could introduce a momentary pause in the stream of messages, long enough to cause the server to overzealously abort the connection.

Because of this concern, the server MUST NOT send a Keepalive message (either a response to a client-initiated request, or a server-initiated message) with an keepalive interval value less than ten seconds. If a client receives an Keepalive message specifying an keepalive interval value less than ten seconds this is an error and the client MUST immediately terminate the connection with a TCP RST (or equivalent for other protocols).

Similarly, the server MUST NOT send a Keepalive message (either a response to a client-initiated request, or a server-initiated message) with an idle timeout value less than ten seconds. If a client receives an Keepalive message specifying an idle timeout value less than ten seconds this is an error and the client MUST immediately terminate the connection with a TCP RST (or equivalent for other protocols).

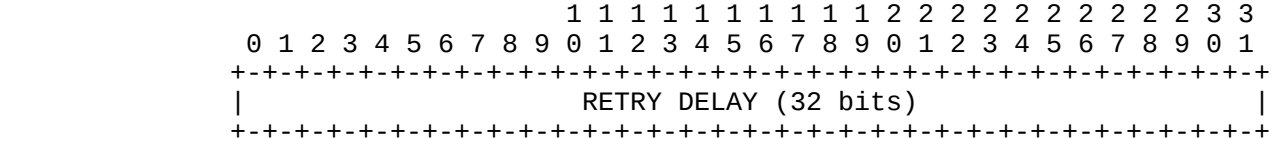
When a client is sending its second and subsequent Keepalive Session Signaling request to the server, the client SHOULD continue to request its preferred values each time. This allows flexibility, so that if conditions change during the lifefime of a session, the server can adapt its responses to better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

The Keepalive TLV (1) has similar intent to the EDNS(0) TCP Keepalive Option [RFC7828]. A client/server pair that supports Session Signaling MUST NOT use the EDNS(0) TCP KeepAlive option within any message on a session once bi-directional Session Signaling support has been confirmed. Once bi-directional Session Signaling support has been confirmed, if either client or server receives a DNS message over the session that contains an EDNS(0) TCP KeepAlive option, this is an error and the receiver of the EDNS(0) TCP KeepAlive option MUST immediately terminate the connection with a TCP RST (or equivalent for other protocols).

5. Retry Delay TLV

The Retry Delay TLV (0) is used by a server to request that a client close the session, and not to reconnect for the indicated time interval. It is also used as a modifier on error responses, to indicate how long the client should wait before retrying that particular operation.

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



RETRY DELAY: a time value, specified as a 32 bit word in network order in units of milliseconds, within which the client MUST NOT retry this operation, or retry connecting to this server.

The RECOMMENDED value is 10 seconds.

In the case of a client request that returns a nonzero RCODE value, the server MAY append a Retry Delay TLV (0) to the response, indicating the time interval during which the client SHOULD NOT attempt this operation again.

When appended to a Session Signaling response message for some client request, the Retry Delay TLV (0) is considered a Modifier TLV. The indicated time interval during which the client SHOULD NOT retry applies only to the failed operation, not to the session as a whole.

When sent in a Session Signaling request message, from server to client, the Retry Delay TLV (0) is considered an Operation TLV. It applies to the session as a whole, and the client MUST close the session, as described previously. The RCODE MUST indicate the reason for the termination. RCODE NOERROR indicates a routine shutdown. RCODE SERVFAIL indicates that the server is overloaded due to resource exhaustion. RCODE REFUSED indicates that the server has been reconfigured and is no longer able to perform one or more of the functions currently being performed on this session (for example, a DNS Push Notification server could be reconfigured such that is is no longer accepting DNS Push Notification requests for one or more of the currently subscribed names).

This document specifies only these three RCODE values for Retry Delay request. Servers sending Retry Delay requests SHOULD use one of these three values. However, future circumstances may create

situations where other RCODE values are appropriate in Retry Delay requests, so clients MUST be prepared to accept Retry Delay requests with any RCODE value.

6. IANA Considerations

6.1. DNS Session Signaling Opcode Registration

IANA are directed to assign a value (tentatively 6) in the DNS Opcodes Registry for the Session Signaling Opcode.

6.2. DNS Session Signaling RCODE Registration

IANA are directed to assign a value (tentatively 11) in the DNS RCODE Registry for the SSOPNOTIMP error code.

6.3. DNS Session Signaling Type Codes Registry

IANA are directed to create the DNS Session Signaling Type Codes Registry, with initial values as follows:

Type	Name	Status	Reference
0x0000	SSOP-RetryDelay	Standard	RFC-TBD
0x0001	SSOP-KeepAlive	Standard	RFC-TBD
0x0002 - 0x003F	Unassigned, reserved for session management TLVs		
0x0040 - 0xF7FF	Unassigned		
0xF800 - 0xFBFF	Reserved for local / experimental use		
0xFC00 - 65535	Reserved for future expansion		

Registration of additional Session Signaling Type Codes requires publication of an appropriate IETF "Standards Action" or "IESG Approval" document [RFC5226].

7. 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.

8. Acknowledgements

Thanks to Tim Chown, Ralph Droms, Jan Komissar, and Manju Shankar Rao for their helpful contributions to this document.

9. References

9.1. Normative References

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

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

[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, <http://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, <http://www.rfc-editor.org/info/rfc2119>.

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

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

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, DOI 10.17487/RFC5226, May 2008, <http://www.rfc-editor.org/info/rfc5226>.

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

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

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

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

9.2. Informative References

[I-D.ietf-dnssd-push] Pusateri, T. and S. Cheshire, "DNS Push Notifications", draft-ietf-dnssd-push-09 (work in progress), October 2016.

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

[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, <http://www.rfc-editor.org/info/rfc7858>.

Authors' Addresses

Ray Bellis
Internet Systems Consortium, Inc.
950 Charter Street
Redwood City CA 94063
USA

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

Stuart Cheshire
Apple Inc.
1 Infinite Loop
Cupertino CA 95014
USA

Phone: +1 408 974 3207
Email: cheshire@apple.com

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

Email: jad@sinodun.com

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

Email: sara@sinodun.com

Allison Mankin
Salesforce

Email: allison.mankin@gmail.com

Tom Pusateri
Unaffiliated

Phone: +1 843 473 7394
Email: pusateri@bangj.com