Stream: Internet Engineering Task Force (IETF)
RFC: 8765
Category: Standards Track
Published: June 2020
ISSN: 2070-1721
Authors: T. Pusateri *Unaffiliated*
S. Cheshire *Apple Inc.*

RFC 8765
DNS Push Notifications

Abstract

The Domain Name System (DNS) was designed to return matching records efficiently for queries for data that are relatively static. When those records change frequently, DNS is still efficient at returning the updated results when polled, as long as the polling rate is not too high. But, there exists no mechanism for a client to be asynchronously notified when these changes occur. This document defines a mechanism for a client to be notified of such changes to DNS records, called DNS Push Notifications.

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/rfc8765.

Copyright Notice

Table of Contents

1. Introduction

Domain Name System (DNS) records may be updated using DNS Update [RFC2136]. Other mechanisms such as a Discovery Proxy [RFC8766] can also generate changes to a DNS zone. This document specifies a protocol for DNS clients to subscribe to receive asynchronous notifications of changes to RRsets of interest. It is immediately relevant in the case of DNS-based Service Discovery [RFC6763] but is not limited to that use case; it provides a general DNS mechanism for DNS record change notifications. Familiarity with the DNS protocol and DNS packet formats is assumed [RFC1034] [RFC1035] [RFC6895].

1.1. 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.

1.2. Fatal Errors

Certain invalid situations are described in this specification, such as a server sending a Push Notification subscription request to a client, or a client sending a Push Notification response to a server. These should never occur with a correctly implemented client and server, and if they do occur, then they indicate a serious implementation error. In these extreme cases, there is no reasonable expectation of a graceful recovery, and the recipient detecting the error should respond by unilaterally aborting the session without regard for data loss. Such cases are addressed by having an engineer investigate the cause of the failure and fixing the problem in the software.

Where this specification says "forcibly abort", it means sending a TCP RST to terminate the TCP connection and the TLS session running over that TCP connection. In the BSD Sockets API, this is achieved by setting the SO_LINGER option to zero before closing the socket.

2. Motivation

As the domain name system continues to adapt to new uses and changes in deployment, polling has the potential to burden DNS servers at many levels throughout the network. Other network protocols have successfully deployed a publish/subscribe model following the Observer design pattern [OBS]. Extensible Messaging and Presence Protocol (XMPP) Publish-Subscribe [XEP0060] and Atom [RFC4287] are examples. While DNS servers are generally highly tuned and capable of a high rate of query/response traffic, adding a publish/subscribe model for tracking changes to DNS records can deliver more timely notifications of changes with reduced CPU usage and lower network traffic.

The guiding design principle of DNS Push Notifications is that clients that choose to use DNS Push Notifications, instead of repeated polling with DNS queries, will receive the same results as they could via sufficiently rapid polling, except more efficiently. This means that the rules for which records match a given DNS Push Notification subscription are the same as the already established rules used to determine which records match a given DNS query [RFC1034]. For example, name comparisons are done in a case-insensitive manner, and a record of type CNAME in a zone matches any DNS TYPE in a query or subscription.

Multicast DNS [RFC6762] implementations always listen on a well-known link-local IP multicast group address, and changes are sent to that multicast group address for all group members to receive. Therefore, Multicast DNS already has asynchronous change notification capability. When DNS-based Service Discovery [RFC6763] is used across a wide area network using Unicast DNS (possibly facilitated via a Discovery Proxy [RFC8766]), it would be beneficial to have an equivalent capability for Unicast DNS in order to allow clients to learn about DNS record changes in a timely manner without polling.

The DNS Long-Lived Queries (LLQ) mechanism [RFC8764] is an existing deployed solution to provide asynchronous change notifications; it was used by Apple's Back to My Mac [RFC6281] service introduced in Mac OS X 10.5 Leopard in 2007. Back to My Mac was designed in an era when the data center operations staff asserted that it was impossible for a server to handle large numbers of TCP connections, even if those connections carried very little traffic and spent most of their time idle. Consequently, LLQ was defined as a UDP-based protocol, effectively replicating much of TCP's connection state management logic in user space and creating its own imitation of existing TCP features like flow control, reliability, and the three-way handshake.

This document builds on experience gained with the LLQ protocol, with an improved design. Instead of using UDP, this specification uses DNS Stateful Operations (DSO) [RFC8490] running over TLS over TCP, and therefore doesn't need to reinvent existing TCP functionality. Using TCP also gives long-lived low-traffic connections better longevity through NAT gateways without depending on the gateway to support NAT Port Mapping Protocol (NAT-PMP) [RFC6886] or Port Control Protocol (PCP) [RFC6887], or resorting to excessive keepalive traffic.

3. Overview

A DNS Push Notification client subscribes for Push Notifications for a particular RRset by connecting to the appropriate Push Notification server for that RRset and sending DSO message (s) indicating the RRset(s) of interest. When the client loses interest in receiving further updates to these records, it unsubscribes.

The DNS Push Notification server for a DNS zone is any server capable of generating the correct change notifications for a name. It may be a primary, secondary, or stealth name server [RFC8499].

The `_dns-push-tls._tcp.<zone>` SRV record for a zone **MAY** reference the same target host and port as that zone's `_dns-update-tls._tcp.<zone>` SRV record. When the same target host and port is offered for both DNS Updates and DNS Push Notifications, a client **MAY** use a single DSO session to that server for both DNS Updates and DNS Push Notification subscriptions. DNS Updates and DNS Push Notifications may be handled on different ports on the same target host, in which case they are not considered to be the "same server" for the purposes of this specification, and communications with these two ports are handled independently. Supporting DNS Updates and DNS Push Notifications on the same server is **OPTIONAL**. A DNS Push Notification server is not required to support DNS Update.

Standard DNS Queries **MAY** be sent over a DNS Push Notification (i.e., DSO) session. For any zone for which the server is authoritative, it **MUST** respond authoritatively for queries for names falling within that zone (e.g., the `_dns-push-tls._tcp.<zone>` SRV record) both for normal DNS queries and for DNS Push Notification subscriptions. For names for which the server is acting as a recursive resolver (e.g., when the server is the local recursive resolver) for any query for which it supports DNS Push Notification subscriptions, it **MUST** also support standard queries.

DNS Push Notifications impose less load on the responding server than rapid polling would, but Push Notifications do still have a cost. Therefore, DNS Push Notification clients **MUST NOT** recklessly create an excessive number of Push Notification subscriptions. Specifically:

(a) A subscription should only be active when there is a valid reason to need live data (for example, an on-screen display is currently showing the results to the user), and the subscription **SHOULD** be canceled as soon as the need for that data ends (for example, when the user dismisses that display). In the case of a device like a smartphone that, after some period of inactivity, goes to sleep or otherwise darkens its screen, it should cancel its subscriptions when darkening the screen (since the user cannot see any changes on the display anyway) and reinstate its subscriptions when reawakening from display sleep.

(b) A DNS Push Notification client **SHOULD NOT** routinely keep a DNS Push Notification subscription active 24 hours a day, 7 days a week, just to keep a list in memory up to date so that if the user does choose to bring up an on-screen display of that data, it can be displayed really fast. DNS Push Notifications are designed to be fast enough that there is no need to pre-load a "warm" list in memory just in case it might be needed later.

Generally, as described in the DNS Stateful Operations specification [RFC8490], a client must not keep a DSO session to a server open indefinitely if it has no subscriptions (or other operations) active on that session. A client should begin closing a DSO session immediately after it becomes idle, and then, if needed in the future, open a new session when required. Alternatively, a client may speculatively keep an idle DSO session open for some time, subject to the constraint that it must not keep a session open that has been idle for more than the session's idle timeout (15 seconds by default) [RFC8490].

Note that a DSO session that has an active DNS Push Notification subscription is not considered idle, even if there is no traffic flowing for an extended period of time. In this case, the DSO inactivity timeout does not apply, because the session is not inactive, but the keepalive interval does still apply, to ensure 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 is described in Section 6.2 of the DSO specification [RFC8490].

4. State Considerations

Each DNS Push Notification server is capable of handling some finite number of Push Notification subscriptions. This number will vary from server to server and is based on physical machine characteristics, network capacity, and operating system resource allocation. After a client establishes a session to a DNS server, each subscription is individually accepted or rejected. Servers may employ various techniques to limit subscriptions to a manageable level. Correspondingly, the client is free to establish simultaneous sessions to alternate DNS servers that support DNS Push Notifications for the zone and distribute subscriptions at the client's discretion. In this way, both clients and servers can react to resource constraints.

5. Transport

Other DNS operations like DNS Update [RFC2136] **MAY** use either DNS over User Datagram Protocol (UDP) [RFC0768] or DNS over Transmission Control Protocol (TCP) [RFC0793] as the transport protocol, provided they follow the historical precedent that DNS queries must first be sent using DNS over UDP and only switch to DNS over TCP if needed [RFC1123]. This requirement to prefer UDP has subsequently been relaxed [RFC7766].

In keeping with the more recent precedent, DNS Push Notification is defined only for TCP. DNS Push Notification clients **MUST** use DNS Stateful Operations [RFC8490] running over TLS over TCP [RFC7858].

Connection setup over TCP ensures return reachability and alleviates concerns of state overload at the server, a potential problem with connectionless protocols, which can be more vulnerable to being exploited by attackers using spoofed source addresses. All subscribers are guaranteed to be reachable by the server by virtue of the TCP three-way handshake. Flooding attacks are possible with any protocol, and a benefit of TCP is that there are already established industry best practices to guard against SYN flooding and similar attacks [SYN] [RFC4953].

Use of TCP also allows DNS Push Notifications to take advantage of current and future developments in TCP such as Multipath TCP (MPTCP) [RFC8684], TCP Fast Open (TFO) [RFC7413], the TCP RACK fast loss detection algorithm [TCPRACK], and so on.

Transport Layer Security (TLS) [RFC8446] is well understood and is used by many application-layer protocols running over TCP. TLS is designed to prevent eavesdropping, tampering, and message forgery. TLS is **REQUIRED** for every connection between a client subscriber and server in this protocol specification. Additional security measures such as client authentication during TLS negotiation may also be employed to increase the trust relationship between client and server.

6. Protocol Operation

The DNS Push Notification protocol is a session-oriented protocol and makes use of DNS Stateful Operations (DSO) [RFC8490].

For details of the DSO message format, refer to the DNS Stateful Operations specification [RFC8490]. Those details are not repeated here.

DNS Push Notification clients and servers **MUST** support DSO. A single server can support DNS Queries, DNS Updates, and DNS Push Notifications (using DSO) on the same TCP port.

A DNS Push Notification exchange begins with the client discovering the appropriate server, using the procedure described in Section 6.1, and then making a TLS/TCP connection to it.

After making the TLS/TCP connection to the server, a typical DNS Push Notification client will then immediately issue a DSO Keepalive operation to establish the DSO session and request a session timeout and/or keepalive interval longer than the 15-second default values, but this is not required. A DNS Push Notification client **MAY** issue other requests on the session first, and only issue a DSO Keepalive operation later if it determines that to be necessary. Sending either a DSO Keepalive operation or a Push Notification subscription request over the TLS/TCP connection to the server signals the client's support of DSO and serves to establish a DSO session.

In accordance with the current set of active subscriptions, the server sends relevant asynchronous Push Notifications to the client. Note that a client **MUST** be prepared to receive (and silently ignore) Push Notifications for subscriptions it has previously removed, since there is no way to prevent the situation where a Push Notification is in flight from server to client while the client's UNSUBSCRIBE message canceling that subscription is simultaneously in flight from client to server.

6.1. Discovery

The first step in establishing a DNS Push Notification subscription is to discover an appropriate DNS server that supports DNS Push Notifications for the desired zone.

The client begins by opening a DSO session to its normal configured DNS recursive resolver and requesting a Push Notification subscription. This connection is made to TCP port 853, the default port for DNS over TLS [RFC7858]. If the request for a Push Notification subscription is successful,

and the recursive resolver doesn't already have an active subscription for that name, type, and class, then the recursive resolver will make a corresponding Push Notification subscription on the client's behalf. Results received are relayed to the client. This is closely analogous to how a client sends a normal DNS query to its configured DNS recursive resolver, which, if it doesn't already have appropriate answer(s) in its cache, issues an upstream query to satisfy the request.

In many contexts, the recursive resolver will be able to handle Push Notifications for all names that the client may need to follow. Use of VPN tunnels and Private DNS [RFC8499] can create some additional complexity in the client software here; the techniques to handle VPN tunnels and Private DNS for DNS Push Notifications are the same as those already used to handle this for normal DNS queries.

If the recursive resolver does not support DNS over TLS, or supports DNS over TLS but is not listening on TCP port 853, or supports DNS over TLS on TCP port 853 but does not support DSO on that port, then the DSO session establishment will fail [RFC8490].

If the recursive resolver does support DSO on TCP port 853 but does not support Push Notification subscriptions, then when the client attempts to create a subscription, the server will return the DSO error code DSOTYPENI (11).

In some cases, the recursive resolver may support DSO and Push Notification subscriptions but may not be able to subscribe for Push Notifications for a particular name. In this case, the recursive resolver should return SERVFAIL to the client. This includes being unable to establish a connection to the zone's DNS Push Notification server or establishing a connection but receiving a non-success response code. In some cases, where the client has a pre-established trust relationship with the owner of the zone (that is not handled via the usual mechanisms for VPN software), the client may handle these failures by contacting the zone's DNS Push Notification server directly.

In any of the cases described above where the client fails to establish a DNS Push Notification subscription via its configured recursive resolver, the client should proceed to discover the appropriate server for direct communication. The client **MUST** also determine on which TCP port the server is listening for connections, which need not be, and often is not, TCP port 53 (traditionally used for conventional DNS) or TCP port 853 (traditionally used for DNS over TLS).

The discovery algorithm described here is an iterative algorithm, which starts with the full name of the record to which the client wishes to subscribe. Successive SOA queries are then issued, trimming one label each time, until the closest enclosing authoritative server is discovered. There is also an optimization to enable the client to take a "short cut" directly to the SOA record of the closest enclosing authoritative server in many cases.

1. The client begins the discovery by sending a DNS query to its local resolver, with record type SOA [RFC1035] for the record name to which it wishes to subscribe. As an example, suppose the client wishes to subscribe to PTR records with the name `_ipp._tcp.headoffice.example.com` (to discover Internet Printing Protocol (IPP) printers [RFC8010] [RFC8011] being advertised in the head office of Example Company). The client begins by sending an SOA query for `_ipp._tcp.headoffice.example.com` to the local

recursive resolver. The goal is to determine the server that is authoritative for the name `_ipp._tcp.headoffice.example.com`. The closest enclosing DNS zone containing the name `_ipp._tcp.headoffice.example.com` could be `example.com`, or `headoffice.example.com`, or `_tcp.headoffice.example.com`, or even `_ipp._tcp.headoffice.example.com`. The client does not know in advance where the closest enclosing zone cut occurs, which is why it uses the iterative procedure described here to discover this information.

2. If the requested SOA record exists, it will be returned in the Answer Section with a NOERROR response code, and the client has succeeded in discovering the information it needs.

   (This language is not placing any new requirements on DNS recursive resolvers. This text merely describes the existing operation of the DNS protocol [RFC1034] [RFC1035].)

3. If the requested SOA record does not exist, the client will get back a NOERROR/NODATA response or an NXDOMAIN/Name Error response. In either case, the local resolver would normally include the SOA record for the closest enclosing zone of the requested name in the Authority Section. If the SOA record is received in the Authority Section, then the client has succeeded in discovering the information it needs.

   (This language is not placing any new requirements on DNS recursive resolvers. This text merely describes the existing operation of the DNS protocol regarding negative responses [RFC2308].)

4. If the client receives a response containing no SOA record, then it proceeds with the iterative approach. The client strips the leading label from the current query name, and if the resulting name has at least two labels in it, then the client sends an SOA query for that new name and processing continues at step 2 above, repeating the iterative search until either an SOA is received or the query name consists of a single label, i.e., a Top-Level Domain (TLD). In the case of a single-label name (TLD), this is a network configuration error, which should not happen, and the client gives up. The client may retry the operation at a later time of the client's choosing, such as after a change in network attachment.
5. Once the SOA is known (by virtue of being seen either in the Answer Section or in the Authority Section), the client sends a DNS query with type SRV [RFC2782] for the record name `_dns-push-tls._tcp.<zone>`, where <zone> is the owner name of the discovered SOA record.
6. If the zone in question is set up to offer DNS Push Notifications, then this SRV record **MUST** exist. (If this SRV record does not exist, then the zone is not correctly configured for DNS Push Notifications as specified in this document.) The SRV `target` contains the name of the server providing DNS Push Notifications for the zone. The port number on which to contact the server is in the SRV record `port` field. The address(es) of the target host **MAY** be included in the Additional Section, however, the address records **SHOULD** be authenticated before use as described in Section 7.2 and in the specification for using DNS-Based Authentication of Named Entities (DANE) TLSA Records with SRV Records [RFC7673], if applicable.
7. More than one SRV record may be returned. In this case, the `priority` and `weight` values in the returned SRV records are used to determine the order in which to contact the servers for subscription requests. As described in the SRV specification [RFC2782], the server with the lowest `priority` is first contacted. If more than one server has the same `priority`, the

`weight` indicates the weighted probability that the client should contact that server. Higher weights have higher probabilities of being selected. If a server is not willing to accept a subscription request, or is not reachable within a reasonable time, as determined by the client, then a subsequent server is to be contacted.

Each time a client makes a new DNS Push Notification subscription, it **SHOULD** repeat the discovery process in order to determine the preferred DNS server for that subscription at that time. If a client already has a DSO session with that DNS server, the client **SHOULD** reuse that existing DSO session for the new subscription; otherwise, a new DSO session is established. The client **MUST** respect the DNS TTL values on records it receives while performing the discovery process and store them in its local cache with this lifetime (as it will generally do anyway for all DNS queries it performs). This means that, as long as the DNS TTL values on the authoritative records are set to reasonable values, repeated application of the discovery process can be completed practically instantaneously by the client, using only locally stored cached data.

6.2. DNS Push Notification SUBSCRIBE

After connecting, and requesting a longer idle timeout and/or keepalive interval if necessary, a DNS Push Notification client then indicates its desire to receive DNS Push Notifications for a given domain name by sending a SUBSCRIBE request to the server. A SUBSCRIBE request is encoded in a DSO message [RFC8490]. This specification defines a DSO Primary TLV for DNS Push Notification SUBSCRIBE Requests (DSO Type Code 0x0040).

DSO messages with the SUBSCRIBE TLV as the Primary TLV are permitted in TLS early data, provided that the precautions described in Section 7.3 are followed.

The entity that initiates a SUBSCRIBE request is by definition the client. A server **MUST NOT** send a SUBSCRIBE request over an existing session from a client. If a server does send a SUBSCRIBE request over a DSO session initiated by a client, this is a fatal error and the client **MUST** forcibly abort the connection immediately.

Each SUBSCRIBE request generates exactly one SUBSCRIBE response from the server. The entity that initiates a SUBSCRIBE response is by definition the server. A client **MUST NOT** send a SUBSCRIBE response. If a client does send a SUBSCRIBE response, this is a fatal error and the server **MUST** forcibly abort the connection immediately.

6.2.1. SUBSCRIBE Request

A SUBSCRIBE request begins with the standard DSO 12-byte header [RFC8490], followed by the SUBSCRIBE Primary TLV. A SUBSCRIBE request is illustrated in Figure 1.

The MESSAGE ID field **MUST** be set to a unique value that the client is not using for any other active operation on this DSO session. For the purposes here, a MESSAGE ID is in use on this session if either the client has used it in a request for which it has not yet received a response, or if the client has used it for a subscription that it has not yet canceled using UNSUBSCRIBE. In the SUBSCRIBE response, the server **MUST** echo back the MESSAGE ID value unchanged.

The other header fields **MUST** be set as described in the DSO specification [RFC8490]. The DNS OPCODE field contains the OPCODE value for DNS Stateful Operations (6). The four count fields must be zero, and the corresponding four sections must be empty (i.e., absent).

The DSO-TYPE is SUBSCRIBE (0x0040).

The DSO-LENGTH is the length of the DSO-DATA that follows, which specifies the name, type, and class of the record(s) being sought.

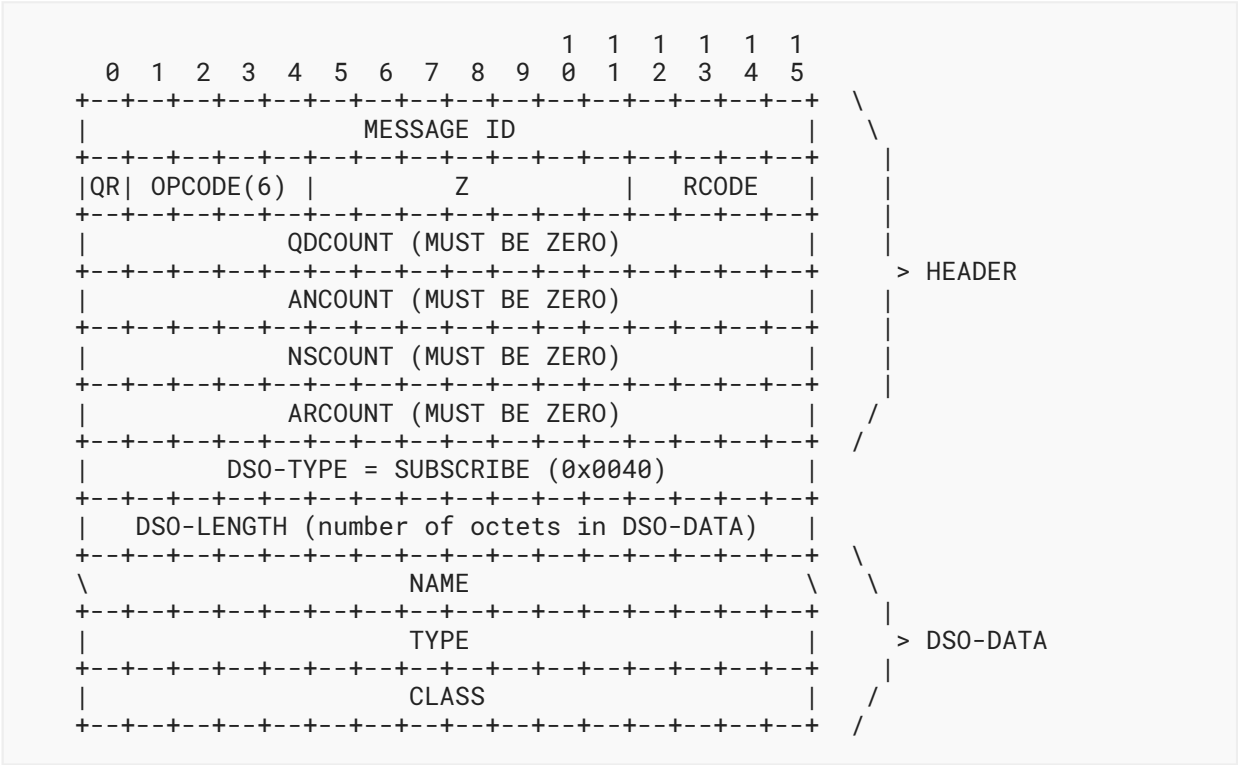

Figure 1: SUBSCRIBE Request

The DSO-DATA for a SUBSCRIBE request **MUST** contain exactly one NAME, TYPE, and CLASS. Since SUBSCRIBE requests are sent over TCP, multiple SUBSCRIBE DSO request messages can be concatenated in a single TCP stream and packed efficiently into TCP segments.

If accepted, the subscription will stay in effect until the client cancels the subscription using UNSUBSCRIBE or until the DSO session between the client and the server is closed.

SUBSCRIBE requests on a given session **MUST** be unique. A client **MUST NOT** send a SUBSCRIBE message that duplicates the name, type and class of an existing active subscription on that DSO session. For the purpose of this matching, the established DNS case insensitivity for US-ASCII letters [RFC0020] applies (e.g., "example.com" and "Example.com" are the same). If a server receives such a duplicate SUBSCRIBE message, this is a fatal error and the server **MUST** forcibly abort the connection immediately.

DNS wildcarding is not supported. That is, an asterisk character ("*") in a SUBSCRIBE message matches only a literal asterisk character ("*") in a name and nothing else. Similarly, a CNAME in a SUBSCRIBE message matches only a CNAME record with that name in the zone and no other records with that name.

A client may SUBSCRIBE to records that are unknown to the server at the time of the request (providing that the name falls within one of the zone(s) the server is responsible for), and this is not an error. The server **MUST NOT** return NXDOMAIN in this case. The server **MUST** accept these requests and send Push Notifications if and when matching records are found in the future.

If neither TYPE nor CLASS are ANY (255), then this is a specific subscription to changes for the given name, type, and class. If one or both of TYPE or CLASS are ANY (255), then this subscription matches all types and/or all classes as appropriate.

NOTE: A little-known quirk of DNS is that in DNS QUERY requests, QTYPE and QCLASS 255 mean "ANY", not "ALL". They indicate that the server should respond with ANY matching records of its choosing, not necessarily ALL matching records. This can lead to some surprising and unexpected results, where a query returns some valid answers, but not all of them, and makes QTYPE = 255 (ANY) queries less useful than people sometimes imagine.

When used in conjunction with SUBSCRIBE, TYPE 255 and CLASS 255 should be interpreted to mean "ALL", not "ANY". After accepting a subscription where one or both of TYPE or CLASS are 255, the server **MUST** send Push Notification Updates for ALL record changes that match the subscription, not just some of them.

6.2.2. SUBSCRIBE Response

A SUBSCRIBE response begins with the standard DSO 12-byte header [RFC8490]. The QR bit in the header is set indicating it is a response. The header **MAY** be followed by one or more optional Additional TLVs such as a Retry Delay Additional TLV. A SUBSCRIBE response is illustrated in Figure 2.

The MESSAGE ID field **MUST** echo the value given in the MESSAGE ID field of the SUBSCRIBE request. This is how the client knows which request is being responded to.

The other header fields **MUST** be set as described in the DSO specification [RFC8490]. The DNS OPCODE field contains the OPCODE value for DNS Stateful Operations (6). The four count fields must be zero, and the corresponding four sections must be empty (i.e., absent).

A SUBSCRIBE response message **MUST NOT** include a SUBSCRIBE TLV. If a client receives a SUBSCRIBE response message containing a SUBSCRIBE TLV, then the response message is processed but the SUBSCRIBE TLV **MUST** be silently ignored.

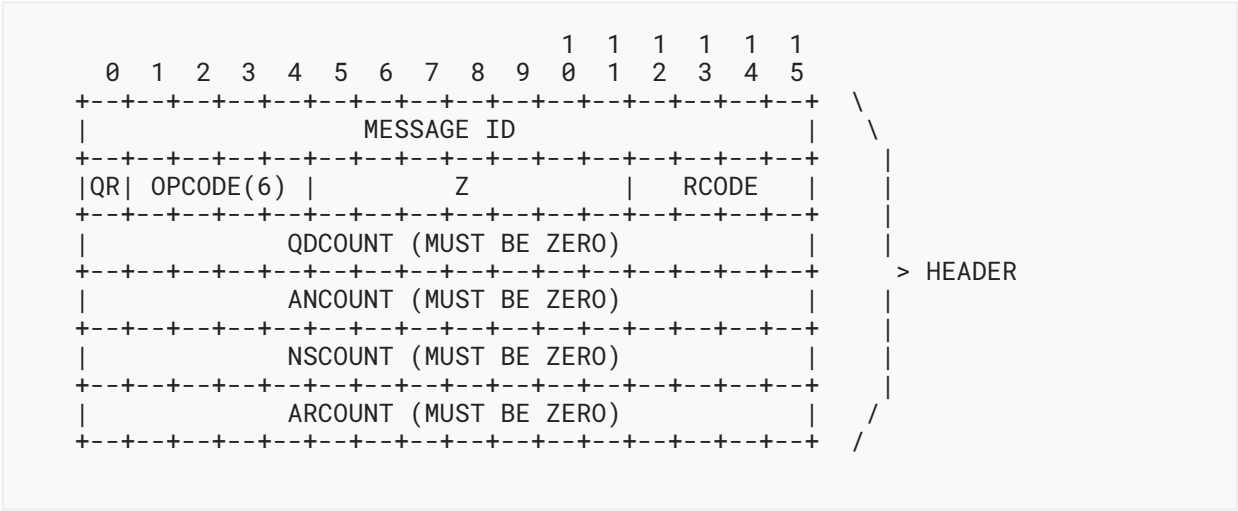

Figure 2: SUBSCRIBE Response

In the SUBSCRIBE response, the RCODE indicates whether or not the subscription was accepted. Supported RCODEs are as follows:

Mnemonic	Value	Description
NOERROR	0	SUBSCRIBE successful.
FORMERR	1	Server failed to process request due to a malformed request.
SERVFAIL	2	Server failed to process request due to a problem with the server.
NOTIMP	4	Server does not implement DSO.
REFUSED	5	Server refuses to process request for policy or security reasons.
NOTAUTH	9	Server is not authoritative for the requested name.
DSOTYPENI	11	SUBSCRIBE operation not supported.

Table 1: SUBSCRIBE Response Codes

This document specifies only these RCODE values for SUBSCRIBE Responses. Servers sending SUBSCRIBE Responses **SHOULD** use one of these values. Note that NXDOMAIN is not a valid RCODE in response to a SUBSCRIBE Request. However, future circumstances may create situations where other RCODE values are appropriate in SUBSCRIBE Responses, so clients **MUST** be prepared to accept and handle SUBSCRIBE Responses with any other nonzero RCODE error values.

If the server sends a nonzero RCODE in the SUBSCRIBE response, that means:

a. the client is (at least partially) misconfigured, or
b. the server resources are exhausted, or

c. there is some other unknown failure on the server.

In any case, the client shouldn't retry the subscription to this server right away. If multiple SRV records were returned as described in Section 6.1, Paragraph 9, Item 7, a subsequent server **MAY** be tried immediately.

If the client has other successful subscriptions to this server, these subscriptions remain even though additional subscriptions may be refused. Neither the client nor the server is required to close the connection, although either end may choose to do so.

If the server sends a nonzero RCODE, then it **SHOULD** append a Retry Delay Additional TLV [RFC8490] to the response specifying a delay before the client attempts this operation again. Recommended values for the delay for different RCODE values are given below. These recommended values apply both to the default values a server should place in the Retry Delay Additional TLV and the default values a client should assume if the server provides no Retry Delay Additional TLV.

For RCODE = 1 (FORMERR), the delay may be any value selected by the implementer. A value of five minutes is **RECOMMENDED** to reduce the risk of high load from defective clients.

For RCODE = 2 (SERVFAIL), the delay should be chosen according to the level of server overload and the anticipated duration of that overload. By default, a value of one minute is **RECOMMENDED**. If a more serious server failure occurs, the delay may be longer in accordance with the specific problem encountered.

For RCODE = 4 (NOTIMP), which occurs on a server that doesn't implement DNS Stateful Operations [RFC8490], it is unlikely that the server will begin supporting DSO in the next few minutes, so the retry delay **SHOULD** be one hour. Note that in such a case, a server that doesn't implement DSO is unlikely to place a Retry Delay Additional TLV in its response, so this recommended value in particular applies to what a client should assume by default.

For RCODE = 5 (REFUSED), which occurs on a server that implements DNS Push Notifications but is currently configured to disallow DNS Push Notifications, the retry delay may be any value selected by the implementer and/or configured by the operator.

If the server being queried is listed in a `_dns-push-tls._tcp.<zone>` SRV record for the zone, then this is a misconfiguration, since this server is being advertised as supporting DNS Push Notifications for this zone, but the server itself is not currently configured to perform that task. Since it is possible that the misconfiguration may be repaired at any time, the retry delay should not be set too high. By default, a value of 5 minutes is **RECOMMENDED**.

For RCODE = 9 (NOTAUTH), which occurs on a server that implements DNS Push Notifications but is not configured to be authoritative for the requested name, the retry delay may be any value selected by the implementer and/or configured by the operator.

If the server being queried is listed in a `_dns-push-tls._tcp.<zone>` SRV record for the zone, then this is a misconfiguration, since this server is being advertised as supporting DNS Push Notifications for this zone, but the server itself is not currently configured to perform that task. Since it is possible that the misconfiguration may be repaired at any time, the retry delay should not be set too high. By default, a value of 5 minutes is **RECOMMENDED**.

For RCODE = 11 (DSOTYPENI), which occurs on a server that implements DSO but doesn't implement DNS Push Notifications, it is unlikely that the server will begin supporting DNS Push Notifications in the next few minutes, so the retry delay **SHOULD** be one hour.

For other RCODE values, the retry delay should be set by the server as appropriate for that error condition. By default, a value of 5 minutes is **RECOMMENDED**.

For RCODE = 9 (NOTAUTH), the time delay applies to requests for other names falling within the same zone. Requests for names falling within other zones are not subject to the delay. For all other RCODEs, the time delay applies to all subsequent requests to this server.

After sending an error response, the server **MAY** allow the session to remain open, or **MAY** follow it with a DSO Retry Delay operation (using the Retry Delay Primary TLV) instructing the client to close the session as described in the DSO specification [RFC8490]. Clients **MUST** correctly handle both cases. Note that the DSO Retry Delay operation (using the Retry Delay Primary TLV) is different to the Retry Delay Additional TLV mentioned above.

6.3. DNS Push Notification Updates

Once a subscription has been successfully established, the server generates PUSH messages to send to the client as appropriate. In the case that the answer set was already non-empty at the moment the subscription was established, an initial PUSH message will be sent immediately following the SUBSCRIBE Response. Subsequent changes to the answer set are then communicated to the client in subsequent PUSH messages.

A client **MUST NOT** send a PUSH message. If a client does send a PUSH message, or a PUSH message is sent with the QR bit set indicating that it is a response, this is a fatal error and the receiver **MUST** forcibly abort the connection immediately.

6.3.1. PUSH Message

A PUSH unidirectional message begins with the standard DSO 12-byte header [RFC8490], followed by the PUSH Primary TLV. A PUSH message is illustrated in Figure 3.

In accordance with the definition of DSO unidirectional messages, the MESSAGE ID field **MUST** be zero. There is no client response to a PUSH message.

The other header fields **MUST** be set as described in the DSO specification [RFC8490]. The DNS OPCODE field contains the OPCODE value for DNS Stateful Operations (6). The four count fields must be zero, and the corresponding four sections must be empty (i.e., absent).

The DSO-TYPE is PUSH (0x0041).

The DSO-LENGTH is the length of the DSO-DATA that follows, which specifies the changes being communicated.

The DSO-DATA contains one or more change notifications. A PUSH Message **MUST** contain at least one change notification. If a PUSH Message is received that contains no change notifications, this is a fatal error and the client **MUST** forcibly abort the connection immediately.

The change notification records are formatted similarly to how DNS Resource Records are conventionally expressed in DNS messages, as illustrated in Figure 3, and are interpreted as described below.

The TTL field holds an unsigned 32-bit integer [RFC2181]. If the TTL is in the range 0 to 2,147,483,647 seconds (0 to 2^{31} - 1, or 0x7FFFFFFF), then a new DNS Resource Record with the given name, type, class, and RDATA is added. Type and class **MUST NOT** be 255 (ANY). If either type or class are 255 (ANY), this is a fatal error and the client **MUST** forcibly abort the connection immediately. A TTL of 0 means that this record should be retained for as long as the subscription is active and should be discarded immediately the moment the subscription is canceled.

If the TTL has the value 0xFFFFFFFF, then the DNS Resource Record with the given name, type, class, and RDATA is removed. Type and class **MUST NOT** be 255 (ANY). If either type or class are 255 (ANY), this is a fatal error and the client **MUST** forcibly abort the connection immediately.

If the TTL has the value 0xFFFFFFFE, then this is a 'collective' remove notification. For collective remove notifications, RDLEN **MUST** be zero, and consequently, the RDATA **MUST** be empty. If a change notification is received where TTL = 0xFFFFFFFE and RDLEN is not zero, this is a fatal error and the client **MUST** forcibly abort the connection immediately.

There are three types of collective remove notification. For collective remove notifications:

- If CLASS is not 255 (ANY) and TYPE is not 255 (ANY), then for the given name, this removes all records of the specified type in the specified class.
- If CLASS is not 255 (ANY) and TYPE is 255 (ANY), then for the given name, this removes all records of all types in the specified class.
- If CLASS is 255 (ANY), then for the given name, this removes all records of all types in all classes. In this case, TYPE **MUST** be set to zero on transmission and **MUST** be silently ignored on reception.

Summary of change notification types:

- Remove all RRsets from a name in all classes:
  TTL = 0xFFFFFFFE, RDLEN = 0, CLASS = 255 (ANY).
- Remove all RRsets from a name in given class:
  TTL = 0xFFFFFFFE, RDLEN = 0, CLASS gives class, TYPE = 255 (ANY).
- Remove specified RRset from a name in given class:
  TTL = 0xFFFFFFFE, RDLEN = 0,
  CLASS and TYPE specify the RRset being removed.
- Remove an individual RR from a name:
  TTL = 0xFFFFFFFF,
  CLASS, TYPE, RDLEN, and RDATA specify the RR being removed.

- Add individual RR to a name:
  TTL >= 0 and TTL <= 0x7FFFFFFF,
  CLASS, TYPE, RDLEN, RDATA, and TTL specify the RR being added.

Note that it is valid for the RDATA of an added or removed DNS Resource Record to be empty (zero length). For example, an Address Prefix List Resource Record [RFC3123] may have empty RDATA. Therefore, a change notification with RDLEN = 0 does not automatically indicate a remove notification. If RDLEN = 0 and TTL is in the range 0 to 0x7FFFFFFF, this change notification signals the addition of a record with the given name, type, class, and empty RDATA. If RDLEN = 0 and TTL = 0xFFFFFFFF, this change notification signals the removal specifically of that single record with the given name, type, class, and empty RDATA.

If the TTL is any value other than 0xFFFFFFFF, 0xFFFFFFFE, or a value in the range 0 to 0x7FFFFFFF, then the receiver **SHOULD** silently ignore this particular change notification record. The connection is not terminated and other valid change notification records within this PUSH message are processed as usual.

In the case where a single change affects more than one active subscription, only one PUSH message is sent. For example, a PUSH message adding a given record may match both a SUBSCRIBE request with the same TYPE and a different SUBSCRIBE request with TYPE = 255 (ANY). It is not the case that two PUSH messages are sent because the new record matches two active subscriptions.

The server **SHOULD** encode change notifications in the most efficient manner possible. For example, when three AAAA records are removed from a given name, and no other AAAA records exist for that name, the server **SHOULD** send a "Remove specified RRset from a name in given class" PUSH message, not three separate "Remove an individual RR from a name" PUSH messages. Similarly, when both an SRV and a TXT record are removed from a given name, and no other records of any kind exist for that name in that class, the server **SHOULD** send a "Remove all RRsets from a name in given class" PUSH message, not two separate "Remove specified RRset from a name in given class" PUSH messages.

For efficiency, when generating a PUSH message, rather than sending each change notification as a separate DSO message, a server **SHOULD** include as many change notifications as it has immediately available to send to that client, even if those change notifications apply to different subscriptions from that client. Conceptually, a PUSH message is a session-level mechanism, not a subscription-level mechanism. Once it has exhausted the list of change notifications immediately available to send to that client, a server **SHOULD** then send the PUSH message immediately rather than waiting speculatively to see if additional change notifications become available.

For efficiency, when generating a PUSH message a server **SHOULD** use standard DNS name compression, with offsets relative to the beginning of the DNS message [RFC1035]. When multiple change notifications in a single PUSH message have the same owner name, this name compression can yield significant savings. Name compression should be performed as specified in Section 18.14 of the Multicast DNS specification [RFC6762]; namely, owner names should always be compressed, and names appearing within RDATA should be compressed for only the RR types listed below:

NS, CNAME, PTR, DNAME, SOA, MX, AFSDB, RT, KX, RP, PX, SRV, NSEC

Servers may generate PUSH messages up to a maximum DNS message length of 16,382 bytes, counting from the start of the DSO 12-byte header. Including the two-byte length prefix that is used to frame DNS over a byte stream like TLS, this makes a total of 16,384 bytes. Servers **MUST NOT** generate PUSH messages larger than this. Where the immediately available change notifications are sufficient to exceed a DNS message length of 16,382 bytes, the change notifications **MUST** be communicated in separate PUSH messages of up to 16,382 bytes each. DNS name compression becomes less effective for messages larger than 16,384 bytes, so little efficiency benefit is gained by sending messages larger than this.

If a client receives a PUSH message with a DNS message length larger than 16,382 bytes, this is a fatal error and the client **MUST** forcibly abort the connection immediately.

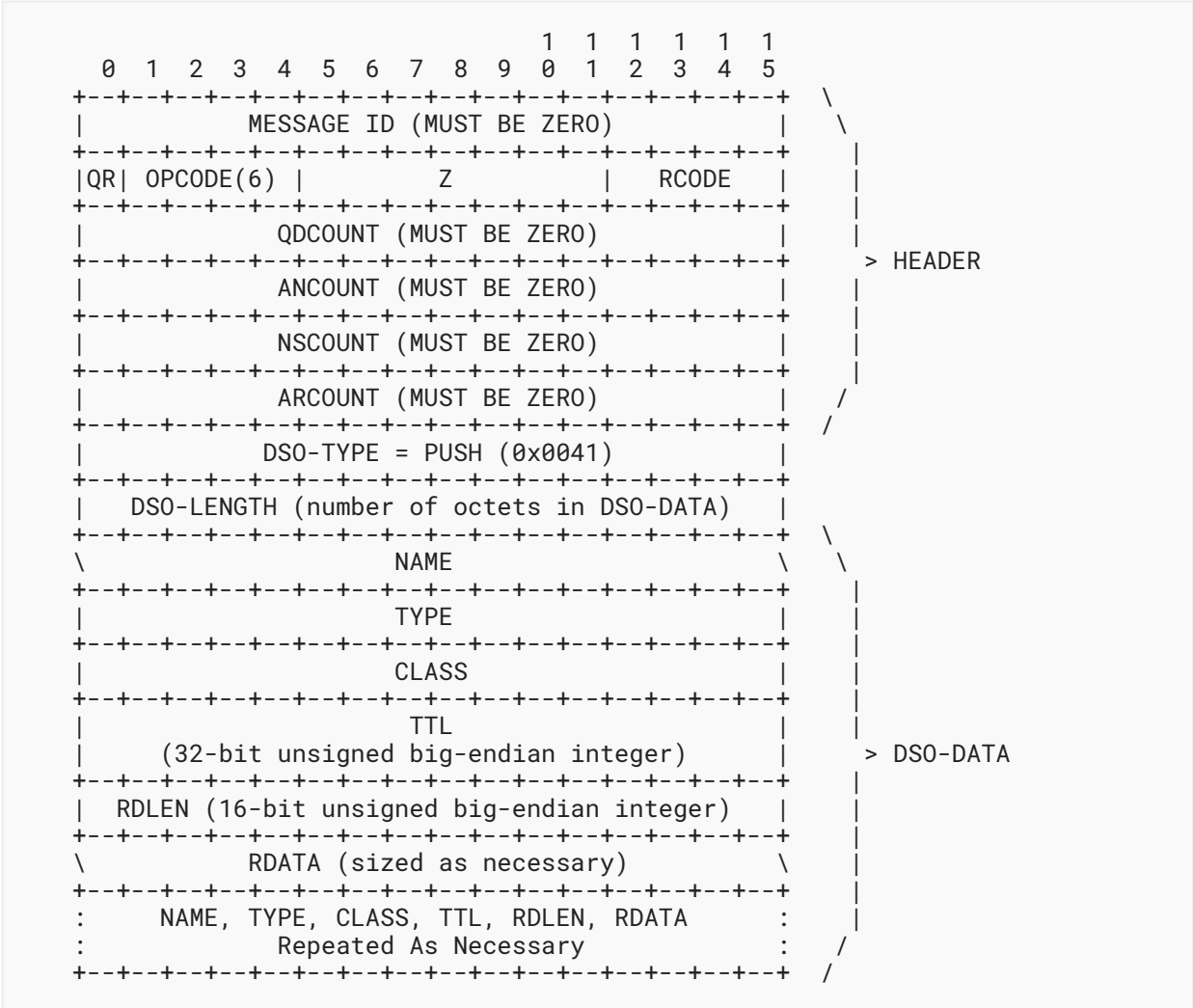

Figure 3: PUSH Message

When processing the records received in a PUSH Message, the receiving client **MUST** validate that the records being added or removed correspond with at least one currently active subscription on that session. Specifically, the record name **MUST** match the name given in the SUBSCRIBE request, subject to the usual established DNS case-insensitivity for US-ASCII letters. For individual additions and removals, if the TYPE in the SUBSCRIBE request was not ANY (255), then the TYPE of the record must either be CNAME or match the TYPE given in the SUBSCRIBE request, and if the CLASS in the SUBSCRIBE request was not ANY (255), then the CLASS of the record must match the CLASS given in the SUBSCRIBE request. For collective removals, at least one of the records being removed must match an active subscription. If a matching active subscription on that session is not found, then that particular addition/removal record is silently ignored. The processing of other additions and removal records in this message is not affected. The DSO session is not closed. This is to allow for the unavoidable race condition where a client sends an outbound UNSUBSCRIBE while inbound PUSH messages for that subscription from the server are still in flight.

The TTL of an added record is stored by the client. While the subscription is active the TTL is not decremented, because a change to the TTL would produce a new update. For as long as a relevant subscription remains active, the client **SHOULD** assume that when a record goes away, the server will notify it of that fact. Consequently, a client does not have to poll to verify that the record is still there. Once a subscription is canceled (individually, or as a result of the DSO session being closed), record aging for records covered by the subscription resumes and records are removed from the local cache when their TTL reaches zero.

6.4. DNS Push Notification UNSUBSCRIBE

To cancel an individual subscription without closing the entire DSO session, the client sends an UNSUBSCRIBE message over the established DSO session to the server.

The entity that initiates an UNSUBSCRIBE message is by definition the client. A server **MUST NOT** send an UNSUBSCRIBE message over an existing session from a client. If a server does send an UNSUBSCRIBE message over a DSO session initiated by a client, or an UNSUBSCRIBE message is sent with the QR bit set indicating that it is a response, this is a fatal error and the receiver **MUST** forcibly abort the connection immediately.

6.4.1. UNSUBSCRIBE Message

An UNSUBSCRIBE unidirectional message begins with the standard DSO 12-byte header [RFC8490], followed by the UNSUBSCRIBE Primary TLV. An UNSUBSCRIBE message is illustrated in Figure 4.

In accordance with the definition of DSO unidirectional messages, the MESSAGE ID field **MUST** be zero. There is no server response to an UNSUBSCRIBE message.

The other header fields **MUST** be set as described in the DSO specification [RFC8490]. The DNS OPCODE field contains the OPCODE value for DNS Stateful Operations (6). The four count fields must be zero, and the corresponding four sections must be empty (i.e., absent).

The DSO-TYPE is UNSUBSCRIBE (0x0042).

The DSO-LENGTH field contains the value 2, the length of the 2-octet MESSAGE ID contained in the DSO-DATA.

The DSO-DATA contains the value previously given in the MESSAGE ID field of an active SUBSCRIBE request. This is how the server knows which SUBSCRIBE request is being canceled. After receipt of the UNSUBSCRIBE message, the SUBSCRIBE request is no longer active.

It is allowable for the client to issue an UNSUBSCRIBE message for a previous SUBSCRIBE request for which the client has not yet received a SUBSCRIBE response. This is to allow for the case where a client starts and stops a subscription in less than the round-trip time to the server. The client is NOT required to wait for the SUBSCRIBE response before issuing the UNSUBSCRIBE message.

Consequently, it is possible for a server to receive an UNSUBSCRIBE message that does not match any currently active subscription. This can occur when a client sends a SUBSCRIBE request, which subsequently fails and returns an error code, but the client sent an UNSUBSCRIBE message before it became aware that the SUBSCRIBE request had failed. Because of this, servers **MUST** silently ignore UNSUBSCRIBE messages that do not match any currently active subscription.

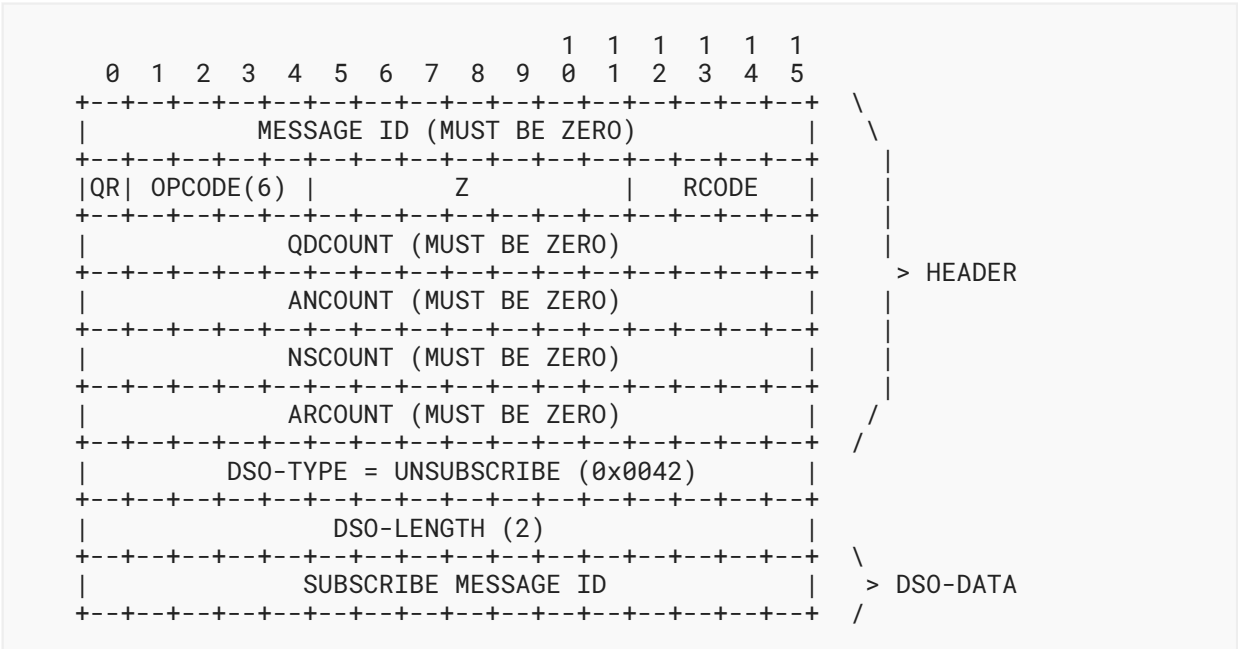

Figure 4: UNSUBSCRIBE Message

6.5. DNS Push Notification RECONFIRM

Sometimes, particularly when used with a Discovery Proxy [RFC8766], a DNS Zone may contain stale data. When a client encounters data that it believes may be stale (e.g., an SRV record referencing a target host+port that is not responding to connection requests), the client can send a RECONFIRM message to ask the server to re-verify that the data is still valid. For a Discovery

Proxy, this causes it to issue new Multicast DNS queries to ascertain whether the target device is still present. How the Discovery Proxy causes these new Multicast DNS queries to be issued depends on the details of the underlying Multicast DNS implementation being used. For example, a Discovery Proxy built on Apple's dns_sd.h API [SD-API] responds to a DNS Push Notification RECONFIRM message by calling the underlying API's DNSServiceReconfirmRecord() routine.

For other types of DNS server, the RECONFIRM operation is currently undefined and **SHOULD** result in a NOERROR response, but it need not cause any other action to occur.

Frequent use of RECONFIRM operations may be a sign of network unreliability, or some kind of misconfiguration, so RECONFIRM operations **MAY** be logged or otherwise communicated to a human administrator to assist in detecting and remedying such network problems.

If, after receiving a valid RECONFIRM message, the server determines that the disputed records are in fact no longer valid, then subsequent DNS PUSH Messages will be generated to inform interested clients. Thus, one client discovering that a previously advertised device (like a network printer) is no longer present has the side effect of informing all other interested clients that the device in question is now gone.

The entity that initiates a RECONFIRM message is by definition the client. A server **MUST NOT** send a RECONFIRM message over an existing session from a client. If a server does send a RECONFIRM message over a DSO session initiated by a client, or a RECONFIRM message is sent with the QR bit set indicating that it is a response, this is a fatal error and the receiver **MUST** forcibly abort the connection immediately.

6.5.1. RECONFIRM Message

A RECONFIRM unidirectional message begins with the standard DSO 12-byte header [RFC8490], followed by the RECONFIRM Primary TLV. A RECONFIRM message is illustrated in Figure 5.

In accordance with the definition of DSO unidirectional messages, the MESSAGE ID field **MUST** be zero. There is no server response to a RECONFIRM message.

The other header fields **MUST** be set as described in the DSO specification [RFC8490]. The DNS OPCODE field contains the OPCODE value for DNS Stateful Operations (6). The four count fields must be zero, and the corresponding four sections must be empty (i.e., absent).

The DSO-TYPE is RECONFIRM (0x0043).

The DSO-LENGTH is the length of the data that follows, which specifies the name, type, class, and content of the record being disputed.

A DNS Push Notifications RECONFIRM message contains exactly one RECONFIRM Primary TLV. The DSO-DATA in a RECONFIRM Primary TLV **MUST** contain exactly one record. The DSO-DATA in a RECONFIRM Primary TLV has no count field to specify more than one record. Since RECONFIRM messages are sent over TCP, multiple RECONFIRM messages can be concatenated in a single TCP stream and packed efficiently into TCP segments. Note that this means that DNS name compression cannot be used between different RECONFIRM messages. However, when a

client is sending multiple RECONFIRM messages this indicates a situation with serious network problems, and this is not expected to occur frequently enough that optimizing efficiency in this case is important.

TYPE **MUST NOT** be the value ANY (255) and CLASS **MUST NOT** be the value ANY (255).

DNS wildcarding is not supported. That is, an asterisk character ("*") in a RECONFIRM message matches only a literal asterisk character ("*") in a name and nothing else. Similarly, a CNAME in a RECONFIRM message matches only a CNAME record with that name in the zone and no other records with that name.

Note that there is no RDLEN field, since the length of the RDATA can be inferred from DSO-LENGTH, so an additional RDLEN field would be redundant.

Following the same rules as for PUSH messages, DNS name compression SHOULD be used within the RDATA of the RECONFIRM message, with offsets relative to the beginning of the DNS message [RFC1035].

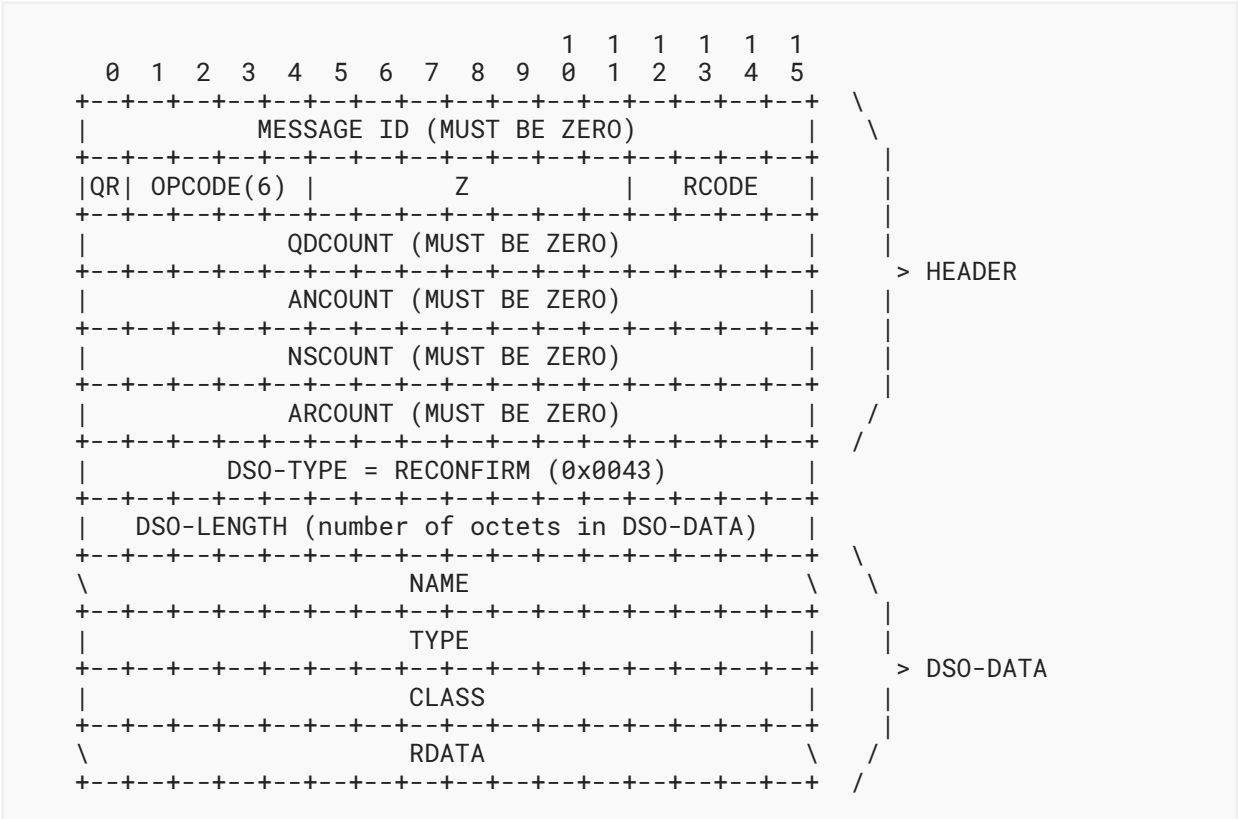

Figure 5: RECONFIRM Message

6.6. DNS Stateful Operations TLV Context Summary

This document defines four new DSO TLVs. As recommended in Section 8.2 of the DNS Stateful Operations specification [RFC8490], the valid contexts of these new TLV types are summarized below.

The client TLV contexts are:

C-P: Client request message, Primary TLV
C-U: Client Unidirectional message, primary TLV
C-A: Client request or unidirectional message, Additional TLV
CRP: Response back to client, Primary TLV
CRA: Response back to client, Additional TLV

TLV Type	C-P	C-U	C-A	CRP	CRA
SUBSCRIBE	X				
PUSH					
UNSUBSCRIBE		X			
RECONFIRM		X			

Table 2: DSO TLV Client Context Summary

The server TLV contexts are:

S-P: Server request message, Primary TLV
S-U: Server Unidirectional message, primary TLV
S-A: Server request or unidirectional message, Additional TLV
SRP: Response back to server, Primary TLV
SRA: Response back to server, Additional TLV

TLV Type	S-P	S-U	S-A	SRP	SRA
SUBSCRIBE					
PUSH		X			
UNSUBSCRIBE					
RECONFIRM					

Table 3: DSO TLV Server Context Summary

6.7. Client-Initiated Termination

An individual subscription is terminated by sending an UNSUBSCRIBE TLV for that specific subscription, or all subscriptions can be canceled at once by the client closing the DSO session. When a client terminates an individual subscription (via UNSUBSCRIBE) or all subscriptions on that DSO session (by ending the session), it is signaling to the server that it is no longer interested in receiving those particular updates. It is informing the server that the server may release any state information it has been keeping with regards to these particular subscriptions.

After terminating its last subscription on a session via UNSUBSCRIBE, a client **MAY** close the session immediately or it may keep it open if it anticipates performing further operations on that session in the future. If a client wishes to keep an idle session open, it **MUST** respect the maximum idle time required by the server [RFC8490].

If a client plans to terminate one or more subscriptions on a session and doesn't intend to keep that session open, then as an efficiency optimization, it **MAY** instead choose to simply close the session, which implicitly terminates all subscriptions on that session. This may occur because the client computer is being shut down, is going to sleep, the application requiring the subscriptions has terminated, or simply because the last active subscription on that session has been canceled.

When closing a session, a client should perform an orderly close of the TLS session. Typical APIs will provide a session close method that will send a TLS close_notify alert as described in Section 6.1 of the TLS 1.3 specification [RFC8446]. This instructs the recipient that the sender will not send any more data over the session. After sending the TLS close_notify alert, the client **MUST** gracefully close the underlying connection using a TCP FIN so that the TLS close_notify is reliably delivered. The mechanisms for gracefully closing a TCP connection with a TCP FIN vary depending on the networking API. For example, in the BSD Sockets API, sending a TCP FIN is achieved by calling "shutdown(s,SHUT_WR)" and keeping the socket open until all remaining data has been read from it.

If the session is forcibly closed at the TCP level by sending a RST from either end of the connection, data may be lost.

6.8. Client Fallback to Polling

There are cases where a client may exhaust all avenues for establishing a DNS Push Notification subscription without success. This can happen if the client's configured recursive resolver does not support DNS over TLS, or supports DNS over TLS but is not listening on TCP port 853, or supports DNS over TLS on TCP port 853 but does not support DSO on that port, or for some other reason is unable to provide a DNS Push Notification subscription. In this case, the client will attempt to communicate directly with an appropriate server, and it may be that the zone apex discovery fails, or there is no `_dns-push-tls._tcp.<zone>` SRV record, or the server indicated in the SRV record is misconfigured, overloaded, or is unresponsive for some other reason.

Regardless of the reason for the failure, after being unable to establish the desired DNS Push Notification subscription, it is likely that the client will still wish to know the answer it seeks, even if that answer cannot be obtained with the timely change notifications provided by DNS Push Notifications. In such cases, it is likely that the client will obtain the answer it seeks via a conventional DNS query instead, repeated at some interval to detect when the answer RRset changes.

In the case where a client responds to its failure to establish a DNS Push Notification subscription by falling back to polling with conventional DNS queries instead, the polling rate should be controlled to avoid placing excessive burden on the server. The interval between successive DNS queries for the same name, type, and class **SHOULD** be at least the minimum of 900 seconds (15 minutes) or two seconds more than the TTL of the answer RRset.

The reason that for TTLs up to 898 seconds the query should not be reissued until two seconds *after* the answer RRset has expired, is to ensure that the answer RRset has also expired from the cache on the client's configured recursive resolver. Otherwise (particularly if the clocks on the client and the recursive resolver do not run at precisely the same rate), there's a risk of a race condition where the client queries its configured recursive resolver just as the answer RRset has one second remaining in the recursive resolver's cache. The client would receive a reply telling it that the answer RRset has one second remaining; the client would then requery the recursive resolver again one second later. If by this time the answer RRset has actually expired from the recursive resolver's cache, the recursive resolver would then issue a new query to fetch fresh data from the authoritative server. Waiting until the answer RRset has definitely expired from the cache on the client's configured recursive resolver avoids this race condition and any unnecessary additional queries it causes.

Each time a client is about to reissue its query to discover changes to the answer RRset, it should first make a new attempt to establish a DNS Push Notification subscription using previously cached DNS answers as appropriate. After a temporary misconfiguration has been remedied, this allows a client that is polling to return to using DNS Push Notifications for asynchronous notification of changes.

7. Security Considerations

The Strict Privacy profile for DNS over TLS is **REQUIRED** for DNS Push Notifications [RFC8310]. Cleartext connections for DNS Push Notifications are not permissible. Since this is a new protocol, transition mechanisms from the Opportunistic Privacy profile are unnecessary.

Also, see Section 9 of the document Usage Profiles for DNS over (D)TLS [RFC8310] for additional recommendations for various versions of TLS usage.

As a consequence of requiring TLS, client certificate authentication and verification may also be enforced by the server for stronger client-server security or end-to-end security. However, recommendations for security in particular deployment scenarios are outside the scope of this document.

DNSSEC is **RECOMMENDED** for the authentication of DNS Push Notification servers. TLS alone does not provide complete security. TLS certificate verification can provide reasonable assurance that the client is really talking to the server associated with the desired host name, but since the desired host name is learned via a DNS SRV query, if the SRV query is subverted, then the client may have a secure connection to a rogue server. DNSSEC can provide added confidence that the SRV query has not been subverted.

7.1. Security Services

It is the goal of using TLS to provide the following security services:

Confidentiality: All application-layer communication is encrypted with the goal that no party should be able to decrypt it except the intended receiver.

Data integrity protection: Any changes made to the communication in transit are detectable by the receiver.

Authentication: An endpoint of the TLS communication is authenticated as the intended entity to communicate with.

Anti-replay protection: TLS provides for the detection of and prevention against messages sent previously over a TLS connection (such as DNS Push Notifications). If prior messages are re-sent at a later time as a form of a man-in-the-middle attack, then the receiver will detect this and reject the replayed messages.

Deployment recommendations on the appropriate key lengths and cipher suites are beyond the scope of this document. Please refer to the current TLS Recommendations [BCP195] for the best current practices. Keep in mind that best practices only exist for a snapshot in time, and recommendations will continue to change. Updated versions or errata may exist for these recommendations.

7.2. TLS Name Authentication

As described in Section 6.1, the client discovers the DNS Push Notification server using an SRV lookup for the record name `_dns-push-tls._tcp.<zone>`. The server connection endpoint **SHOULD** then be authenticated using DANE TLSA records for the associated SRV record. This associates the target's name and port number with a trusted TLS certificate [RFC7673]. This procedure uses the TLS Server Name Indication (SNI) extension [RFC6066] to inform the server of the name the client has authenticated through the use of TLSA records. Therefore, if the SRV record passes DNSSEC validation and a TLSA record matching the target name is usable, an SNI extension must be used for the target name to ensure the client is connecting to the server it has authenticated. If the target name does not have a usable TLSA record, then the use of the SNI extension is optional. See Usage Profiles for DNS over TLS and DNS over DTLS [RFC8310] for more information on authenticating domain names.

7.3. TLS Early Data

DSO messages with the SUBSCRIBE TLV as the Primary TLV are permitted in TLS early data. Using TLS early data can save one network round trip and can result in the client obtaining results faster.

However, there are some factors to consider before using TLS early data.

TLS early data is not forward secret. In cases where forward secrecy of DNS Push Notification subscriptions is required, the client should not use TLS early data.

With TLS early data, there are no guarantees of non-replay between connections. If packets are duplicated and delayed in the network, the later arrivals could be mistaken for new subscription requests. Generally, this is not a major concern since the amount of state generated on the server for these spurious subscriptions is small and short lived since the TCP connection will not complete the three-way handshake. Servers **MAY** choose to implement rate-limiting measures that are activated when the server detects an excessive number of spurious subscription requests.

For further guidance on use of TLS early data, please see discussion of zero round-trip data in Sections 2.3 and 8, and Appendix E.5, of the TLS 1.3 specification [RFC8446].

7.4. TLS Session Resumption

TLS session resumption [RFC8446] is permissible on DNS Push Notification servers. However, closing the TLS connection terminates the DSO session. When the TLS session is resumed, the DNS Push Notification server will not have any subscription state and will proceed as with any other new DSO session. Use of TLS session resumption may allow a TLS connection to be set up more quickly, but the client will still have to recreate any desired subscriptions.

8. IANA Considerations

This document defines a new service name, only applicable for the TCP protocol, which has been recorded in the IANA "Service Name and Transport Protocol Port Number Registry" [RFC6335] [SRVTYPE].

Name	Port	Value	Section
DNS Push Notification Service Type	None	`_dns-push-tls._tcp`	6.1

Table 4: IANA Service Type Assignments

This document defines four new DNS Stateful Operation TLV types, which have been recorded in the IANA "DSO Type Codes" registry [RFC8490] [DSOTYPE].

Name	Value	Early Data	Status	Section
SUBSCRIBE	0x0040	OK	Standards Track	6.2
PUSH	0x0041	NO	Standards Track	6.3
UNSUBSCRIBE	0x0042	NO	Standards Track	6.4
RECONFIRM	0x0043	NO	Standards Track	6.5

Table 5: IANA DSO TLV Type Code Assignments

This document defines no new DNS OPCODEs or RCODEs.

9. References

9.1. Normative References

[DSOTYPE] IANA, "Domain Name System (DNS) Parameters", <https://www.iana.org/assignments/dns-parameters/>.

[RFC0020] Cerf, V., "ASCII format for network interchange", STD 80, RFC 20, DOI 10.17487/RFC0020, October 1969, <https://www.rfc-editor.org/info/rfc20>.

[RFC0768] Postel, J., "User Datagram Protocol", STD 6, RFC 768, DOI 10.17487/RFC0768, August 1980, <https://www.rfc-editor.org/info/rfc768>.

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981, <https://www.rfc-editor.org/info/rfc793>.

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

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

[RFC1123] Braden, R., Ed., "Requirements for Internet Hosts - Application and Support", STD 3, RFC 1123, DOI 10.17487/RFC1123, October 1989, <https://www.rfc-editor.org/info/rfc1123>.

[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>.

[RFC2181] Elz, R. and R. Bush, "Clarifications to the DNS Specification", RFC 2181, DOI 10.17487/RFC2181, July 1997, <https://www.rfc-editor.org/info/rfc2181>.

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

[RFC6066] Eastlake 3rd, D., "Transport Layer Security (TLS) Extensions: Extension Definitions", RFC 6066, DOI 10.17487/RFC6066, January 2011, <https://www.rfc-editor.org/info/rfc6066>.

[RFC6335] Cotton, M., Eggert, L., Touch, J., Westerlund, M., and S. Cheshire, "Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry", BCP 165, RFC 6335, DOI 10.17487/RFC6335, August 2011, <https://www.rfc-editor.org/info/rfc6335>.

[RFC6895] Eastlake 3rd, D., "Domain Name System (DNS) IANA Considerations", BCP 42, RFC 6895, DOI 10.17487/RFC6895, April 2013, <https://www.rfc-editor.org/info/rfc6895>.

[RFC7673] Finch, T., Miller, M., and P. Saint-Andre, "Using DNS-Based Authentication of Named Entities (DANE) TLSA Records with SRV Records", RFC 7673, DOI 10.17487/RFC7673, October 2015, <https://www.rfc-editor.org/info/rfc7673>.

[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>.

[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>.

[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>.

[RFC8310] Dickinson, S., Gillmor, D., and T. Reddy, "Usage Profiles for DNS over TLS and DNS over DTLS", RFC 8310, DOI 10.17487/RFC8310, March 2018, <https://www.rfc-editor.org/info/rfc8310>.

[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>.

[RFC8490] Bellis, R., Cheshire, S., Dickinson, J., Dickinson, S., Lemon, T., and T. Pusateri, "DNS Stateful Operations", RFC 8490, DOI 10.17487/RFC8490, March 2019, <https://www.rfc-editor.org/info/rfc8490>.

[SRVTYPE] IANA, "Service Name and Transport Protocol Port Number Registry", <https://www.iana.org/assignments/service-names-port-numbers/>.

9.2. Informative References

[BCP195] Sheffer, Y., Holz, R., and P. Saint-Andre, "Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", BCP 195, RFC 7525, May 2015, <https://www.rfc-editor.org/info/bcp195>.

[OBS] Wikipedia, "Observer pattern", February 2020, <https://en.wikipedia.org/w/index.php?title=Observer_pattern&oldid=939702131>.

[RFC2308] Andrews, M., "Negative Caching of DNS Queries (DNS NCACHE)", RFC 2308, DOI 10.17487/RFC2308, March 1998, <https://www.rfc-editor.org/info/rfc2308>.

[RFC3123] Koch, P., "A DNS RR Type for Lists of Address Prefixes (APL RR)", RFC 3123, DOI 10.17487/RFC3123, June 2001, <https://www.rfc-editor.org/info/rfc3123>.

[RFC4287] Nottingham, M., Ed. and R. Sayre, Ed., "The Atom Syndication Format", RFC 4287, DOI 10.17487/RFC4287, December 2005, <https://www.rfc-editor.org/info/rfc4287>.

[RFC4953] Touch, J., "Defending TCP Against Spoofing Attacks", RFC 4953, DOI 10.17487/RFC4953, July 2007, <https://www.rfc-editor.org/info/rfc4953>.

[RFC6281] Cheshire, S., Zhu, Z., Wakikawa, R., and L. Zhang, "Understanding Apple's Back to My Mac (BTMM) Service", RFC 6281, DOI 10.17487/RFC6281, June 2011, <https://www.rfc-editor.org/info/rfc6281>.

[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>.

[RFC6886] Cheshire, S. and M. Krochmal, "NAT Port Mapping Protocol (NAT-PMP)", RFC 6886, DOI 10.17487/RFC6886, April 2013, <https://www.rfc-editor.org/info/rfc6886>.

[RFC6887] Wing, D., Ed., Cheshire, S., Boucadair, M., Penno, R., and P. Selkirk, "Port Control Protocol (PCP)", RFC 6887, DOI 10.17487/RFC6887, April 2013, <https://www.rfc-editor.org/info/rfc6887>.

[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>.

[RFC8010] Sweet, M. and I. McDonald, "Internet Printing Protocol/1.1: Encoding and Transport", STD 92, RFC 8010, DOI 10.17487/RFC8010, January 2017, <https://www.rfc-editor.org/info/rfc8010>.

[RFC8011] Sweet, M. and I. McDonald, "Internet Printing Protocol/1.1: Model and Semantics", STD 92, RFC 8011, DOI 10.17487/RFC8011, January 2017, <https://www.rfc-editor.org/info/rfc8011>.

[RFC8499] Hoffman, P., Sullivan, A., and K. Fujiwara, "DNS Terminology", BCP 219, RFC 8499, DOI 10.17487/RFC8499, January 2019, <https://www.rfc-editor.org/info/rfc8499>.

[RFC8684] Ford, A., Raiciu, C., Handley, M., Bonaventure, O., and C. Paasch, "TCP Extensions for Multipath Operation with Multiple Addresses", RFC 8684, DOI 10.17487/RFC8684, March 2020, <https://www.rfc-editor.org/info/rfc8684>.

[RFC8764] Cheshire, S. and M. Krochmal, "Apple's DNS Long-Lived Queries Protocol", RFC 8764, DOI 10.17487/RFC8764, June 2020, <https://www.rfc-editor.org/info/rfc8764>.

[RFC8766] Cheshire, S., "Discovery Proxy for Multicast DNS-Based Service Discovery", RFC 8766, DOI 10.17487/RFC8766, June 2020, <https://www.rfc-editor.org/info/rfc8766>.

[SD-API] Apple Inc., "dns_sd.h", <https://opensource.apple.com/source/mDNSResponder/mDNSResponder-878.70.2/mDNSShared/dns_sd.h.auto.html>.

[SYN] Eddy, W., "Defenses Against TCP SYN Flooding Attacks", The Internet Protocol Journal, Cisco Systems, Volume 9, Number 4, December 2006, <https://www.cisco.com/web/about/ac123/ac147/archived_issues/ipj_9-4/ipj_9-4.pdf>.

[TCPRACK] Cheng, Y., Cardwell, N., Dukkipati, N., and P. Jha, "RACK: a time-based fast loss detection algorithm for TCP", Work in Progress, Internet-Draft, draft-ietf-tcpm-rack-08, 9 March 2020, <https://tools.ietf.org/html/draft-ietf-tcpm-rack-08>.

[XEP0060] Millard, P., Saint-Andre, P., and R. Meijer, "Publish-Subscribe", XSF XEP 0060 , October 2019, <https://xmpp.org/extensions/xep-0060.html>.

Acknowledgments

The authors would like to thank Kiren Sekar and Marc Krochmal for previous work completed in this field.

This document has been improved due to comments from Ran Atkinson, Tim Chown, Sara Dickinson, Mark Delany, Ralph Droms, Jan Komissar, Eric Rescorla, Michael Richardson, David Schinazi, Manju Shankar Rao, Robert Sparks, Markus Stenberg, Andrew Sullivan, Michael Sweet, Dave Thaler, Brian Trammell, Bernie Volz, Éric Vyncke, Christopher Wood, Liang Xia, and Soraia Zlatkovic. Ted Lemon provided clarifying text that was greatly appreciated.

Authors' Addresses

Tom Pusateri
Unaffiliated
Raleigh, NC 27608
United States of America
Phone: +1 919 867 1330
Email: pusateri@bangj.com

Stuart Cheshire
Apple Inc.
One Apple Park Way
Cupertino, CA 95014
United States of America
Phone: +1 (408) 996-1010
Email: cheshire@apple.com