Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: September 10, 2019

T. Pusateri
Unaffiliated
S. Cheshire
Apple Inc.

March 9, 2019

DNS Push Notifications
draft-ietf-dnssd-push-17

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

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 [DisProx] 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 Service Discovery [RFC6763] but is not limited to that use case, and 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. These words may also appear in this document in lower case as plain English words, absent their normative meanings.

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]. 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 notification of changes with reduced CPU usage and lower network traffic.

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

The DNS Long-Lived Queries (LLQ) mechanism [LLQ] is an existing deployed solution to provide asynchronous change notifications, 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 mostly-idle TCP connections, so 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 poor imitations of existing TCP features like the three-way handshake, flow control, and reliability.

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) [DSO] 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. Instead of inventing a new vocabulary of messages to communicate DNS zone changes as LLQ did, this specification borrows the established syntax and semantics of DNS Update messages [RFC2136].

3. Overview

The existing DNS Update protocol [RFC2136] provides a mechanism for clients to add or delete individual resource records (RRs) or entire resource record sets (RRSets) on the zone's server.

This specification adopts a simplified subset of these existing syntax and semantics, and uses them for DNS Push Notification messages going in the opposite direction, from server to client, to communicate changes to a name's records. The client subscribes for Push Notifications by connecting to the server and sending DNS 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 zone is any server capable of generating the correct change notifications for a name. It may be a primary, secondary, or stealth name server [RFC7719]. Consequently, 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 TCP connection to that server for both DNS Updates and DNS Push Notification Subscriptions.

Supporting DNS Updates and DNS Push Notifications on the same server is OPTIONAL. A DNS Push Notification server is NOT REQUIRED also to support DNS Update.

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.

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 on names falling within that zone (e.g., the <zone> in 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 Notification clients are NOT required to implement DNS Update Prerequisite processing. Prerequisites are used to perform tentative atomic test-and-set type operations when a client updates records on a server, and that concept has no applicability when it

comes to an authoritative server unilaterally informing a client of changes to DNS records.

This DNS Push Notification specification includes support for DNS classes, for completeness. However, in practice, it is anticipated that for the foreseeable future the only DNS class in use will be DNS class "IN", as is the reality today with existing DNS servers and clients. A DNS Push Notification server MAY choose to implement only DNS class "IN". If messages are received for a class other than "IN", and that class is not supported, an error with RCODE NOTIMPL (Not Implemented) should be returned.

DNS Push Notifications impose less load on the responding server than rapid polling would, but Push Notifications do still have a cost, so 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 cancelled 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 which, 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 in the display anyway) and reinstate its subscriptions when re-awakening 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 [DSO], a client must not keep a session to a server open indefinitely if it has no subscriptions (or other operations) active on that session. A client MAY close a session as soon as it becomes idle, and then if needed in the future, open a new session when required. Alternatively, a client MAY speculatively keep an idle 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) [DSO].

4. Transport

Other DNS operations like DNS Update [RFC2136] MAY use either User Datagram Protocol (UDP) [RFC0768] or Transmission Control Protocol (TCP) [RFC0793] as the transport protocol, in keeping with the historical precedent that DNS queries must first be sent over UDP [RFC1123]. This requirement to use 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 (DSO) [DSO] running over TLS over TCP [RFC7858].

Connection setup over TCP ensures return reachability and alleviates concerns of state overload at the server through anonymous subscriptions. 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) [RFC6824], TCP Fast Open (TFO) [RFC7413], Tail Loss Probe (TLP) [I-D.dukkipati-tcpm-tcp-loss-probe], and so on.

Transport Layer Security (TLS) [RFC8446] is well understood and deployed across many protocols running over TCP. It 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.

5. 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 bandwidth, 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.

6. Protocol Operation

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

For details of the DSO message format refer to the DNS Stateful Operations specification [DSO]. 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.

A typical DNS Push Notification client will immediately issue a DSO Keepalive operation to request a session timeout and/or keepalive interval longer than the 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 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 cancelling that subscription is simultaneously in flight from client to server.

6.1. Discovery

The first step in 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 DNS over TLS [RFC7858]. If the request for a Push Notification subscription is successful, then the recursive resolver will make a corresponding Push Notification subscription on the client's behalf (if the recursive resolver doesn't already have an active subscription for that name, type, and class), and pass on any results it receives back to the client. This is closely analogous to how a client sends normal DNS queries to its configured DNS recursive resolver, which issues queries on the client's behalf (if the recursive resolver doesn't already have appropriate answer(s) in its cache), and passes on any results it receives back to the client.

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 split-view DNS can create some additional complexity in the client software here; the techniques to handle VPN tunnels and split-view 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 does support DNS over TLS but is not listening on TCP port 853, or does support DNS over TLS on TCP port 853 but does not support DSO on that port, then the DSO Session session establishment will fail [DSO].

If the recursive resolver does support DSO but not Push Notification subscriptions, then it 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 an informative error code to the client so that the client can make an informed decision how to handle the error. If the recursive resolver is unable to establish a connection to the zone's DNS Push Notification server (perhaps because the required SRV record does not exist) the recursive resolver should return SERVFAIL. If the recursive resolver is able to establish a connection to the zone's DNS Push Notification server and some other error code is then received, the recursive resolver should pass on

this received error code back to the client. 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 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 which TCP port on the server is listening for connections, which need not be (and often is not) the typical TCP port 53 used for conventional DNS, or TCP port 853 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.foo.example.com (to discover Internet Printing Protocol (IPP) printers [RFC8010] [RFC8011] being advertised at "foo.example.com"). The client begins by sending an SOA query for _ipp._tcp.foo.example.com to the local recursive resolver. The goal is to determine the server authoritative for the name _ipp._tcp.foo.example.com. The closest enclosing DNS zone containing the name _ipp._tcp.foo.example.com could be example.com, or foo.example.com, or _tcp.foo.example.com, or even _ipp._tcp.foo.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 one label in it, 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 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 after a change in network attachment.

5. Once the SOA is known (either by virtue of being seen 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 below in Section 7.2 and in the specification for using 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 session, it SHOULD repeat the discovery process in order to determine the preferred DNS server for subscriptions at that time. However, the client device MUST respect the DNS TTL values on records it receives, and store them in its local cache with this lifetime. This means that, as long as the DNS TTL values on the authoritative records were set to reasonable values, repeated application of this discovery process can be completed nearly 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 [DSO]. This specification defines a primary DSO TLV for DNS Push Notification SUBSCRIBE Requests (tentatively DSO Type Code 0x40).

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 should immediately abort the connection with a TCP RST (or equivalent for other protocols).

6.2.1. SUBSCRIBE Request

A SUBSCRIBE request begins with the standard DSO 12-byte header [DSO], followed by the SUBSCRIBE primary TLV. A SUBSCRIBE request message 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 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 which it has not yet cancelled 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 [DSO]. 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 (tentatively 0x40).

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.

```
                                1  1  1  1  1  1
  0  1  2  3  4  5  6  7  8  9  0  1  2  3  4  5
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+  \
|                  MESSAGE ID                   |   \
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+    |
|QR| OPCODE(6) |         Z          |   RCODE   |    |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+    |
|             QDCOUNT (MUST BE ZERO)            |    |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+     > HEADER
|             ANCOUNT (MUST BE ZERO)            |    |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+    |
|             NSCOUNT (MUST BE ZERO)            |    |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+    |
|             ARCOUNT (MUST BE ZERO)            |   /
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+  /
|    DSO-TYPE = SUBSCRIBE (tentatively 0x40)    |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|   DSO-LENGTH (number of octets in DSO-DATA)   |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+  \
|                                               |   \
\                     NAME                      \    |
\                                               \    |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+     > DSO-DATA
|                     TYPE                      |    |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+    |
|                     CLASS                     |   /
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+  /
```

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 applies (e.g., "example.com" and "Example.com" are the same). If a server receives such a duplicate SUBSCRIBE message this is an error and the server MUST immediately terminate the connection with a TCP RST (or equivalent for other protocols).

DNS wildcarding is not supported. That is, a wildcard ("*") in a SUBSCRIBE message matches only a literal wildcard character ("*") in the zone, and nothing else.

Aliasing is not supported. That is, a CNAME in a SUBSCRIBE message matches only a literal CNAME record in the zone, and nothing else.

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 any type and/or any class, 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=ANY queries less useful than people sometimes imagine.

When used in conjunction with SUBSCRIBE, TYPE 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

Each SUBSCRIBE request generates exactly one SUBSCRIBE response from the server.

A SUBSCRIBE response begins with the standard DSO 12-byte header [DSO], possibly followed by one or more optional TLVs, such as a Retry Delay TLV.

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

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.

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 SUBSCRIBE Responses with any other RCODE value.

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

a. the client is (at least partially) misconfigured,
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 7, a subsequent server can 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 are 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 TLV [DSO] 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 TLV, and the default values a client should assume if the server provides no Retry Delay 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 [DSO], 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 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 send a DNS Push Notification Retry Delay Operation TLV instructing the client to close the session, as described in the DSO specification [DSO]. Clients MUST correctly handle both cases.

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.

6.3.1. PUSH Message

A PUSH unidirectional message begins with the standard DSO 12-byte header [DSO], followed by the PUSH primary TLV. A PUSH message is illustrated in Figure 2.

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 [DSO]. 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 (tentatively 0x41).

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 Update records. A PUSH Message MUST contain at least one Update record. If a PUSH Message is received that contains zero Update records, this is a fatal error, and the receiver MUST immediately terminate the connection with a TCP RST (or equivalent for other protocols). The Update records are formatted in the customary way for Resource Records in DNS messages. Update records in a PUSH Message are interpreted according to the same rules as for DNS Update [RFC2136] messages, namely:

Delete all RRsets from a name:
TTL=0, CLASS=ANY, RDLENGTH=0, TYPE=ANY.

Delete an RRset from a name:
TTL=0, CLASS=ANY, RDLENGTH=0;
TYPE specifies the RRset being deleted.

Delete an individual RR from a name:
TTL=0, CLASS=NONE;

TYPE, RDLENGTH and RDATA specifies the RR being deleted.

Add to an RRset:
TTL, CLASS, TYPE, RDLENGTH and RDATA specifies the RR being added.

```
                                     1  1  1  1  1  1
       0  1  2  3  4  5  6  7  8  9  0  1  2  3  4  5
     +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+  \
     |           MESSAGE ID (MUST BE ZERO)           |   \
     +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+    |
     |QR| OPCODE(6) |         Z          |   RCODE   |    |
     +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+    |
     |             QDCOUNT (MUST BE ZERO)            |    |
     +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+     > HEADER
     |             ANCOUNT (MUST BE ZERO)            |    |
     +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+    |
     |             NSCOUNT (MUST BE ZERO)            |    |
     +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+    |
     |             ARCOUNT (MUST BE ZERO)            |   /
     +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+  /
     |      DSO-TYPE = PUSH (tentatively 0x41)       |
     +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
     |   DSO-LENGTH (number of octets in DSO-DATA)   |
     +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+  \
     \                     NAME                      \   \
     +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+    |
     |                     TYPE                      |    |
     +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+    |
     |                     CLASS                     |    |
     +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+    |
     |                      TTL                      |    |
     |                   (32 bits)                   |     > DSO-DATA
     +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+    |
     |                     RDLEN                     |    |
     +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+    |
     \                     RDATA                     \    |
     +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+    |
     :     NAME, TYPE, CLASS, TTL, RDLEN, RDATA      :    |
     :             Repeated As Necessary             :   /
     +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+  /
```

Figure 2: PUSH Message

When processing the records received in a PUSH Message, the receiving client MUST validate that the records being added or deleted correspond with at least one currently active subscription on that session. Specifically, the record name MUST match the name given in a SUBSCRIBE request, subject to the usual established DNS case-

insensitivity for US-ASCII letters. If the TYPE in the SUBSCRIBE request was not ANY (255) then the TYPE of the record must match the TYPE given in the SUBSCRIBE request. 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. If a matching active subscription on that session is not found, then that individual record addition/deletion is silently ignored. Processing of other additions and deletions 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.

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=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 deleted from a given name, and no other AAAA records exist for that name, the server SHOULD send a "delete an RRset from a name" PUSH message, not three separate "delete an individual RR from a name" PUSH messages. Similarly, when both an SRV and a TXT record are deleted from a given name, and no other records of any kind exist for that name, the server SHOULD send a "delete all RRsets from a name" PUSH message, not two separate "delete an RRset from a name" PUSH messages.

A server SHOULD combine multiple change notifications in a single PUSH message when possible, even if those change notifications apply to different subscriptions. Conceptually, a PUSH message is a session-level mechanism, not a subscription-level mechanism.

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 cancelled (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 UNSUBSCRIBE message is encoded as a DSO unidirectional message [DSO]. This specification defines a primary unidirectional DSO TLV for DNS Push Notification UNSUBSCRIBE Requests (tentatively DSO Type Code 0x42).

A server MUST NOT initiate an UNSUBSCRIBE request. If a server does send an UNSUBSCRIBE request over a DSO session initiated by a client, this is a fatal error and the client should immediately abort the connection with a TCP RST (or equivalent for other protocols).

6.4.1. UNSUBSCRIBE Request

An UNSUBSCRIBE request begins with the standard DSO 12-byte header [DSO], followed by the UNSUBSCRIBE primary TLV. An UNSUBSCRIBE request 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 server response to a UNSUBSCRIBE message.

The other header fields MUST be set as described in the DSO specification [DSO]. 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 (tentatively 0x42).

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 given in the MESSAGE ID field of an active SUBSCRIBE request. This is how the server knows which SUBSCRIBE request is being cancelled. After receipt of the UNSUBSCRIBE request, the SUBSCRIBE request is no longer active.

It is allowable for the client to issue an UNSUBSCRIBE request 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 request.

Consequently, it is possible for a server to receive an UNSUBSCRIBE request 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 request before it became aware that the SUBSCRIBE request had failed. Because of this, servers MUST silently ignore UNSUBSCRIBE requests that do not match any currently active subscription.

```
                                1  1  1  1  1  1
  0  1  2  3  4  5  6  7  8  9  0  1  2  3  4  5
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+  \
|           MESSAGE ID (MUST BE ZERO)           |   \
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+    |
|QR| OPCODE(6) |         Z          |   RCODE   |    |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+    |
|             QDCOUNT (MUST BE ZERO)            |    |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+     > HEADER
|             ANCOUNT (MUST BE ZERO)            |    |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+    |
|             NSCOUNT (MUST BE ZERO)            |    |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+    |
|             ARCOUNT (MUST BE ZERO)            |   /
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+  /
|   DSO-TYPE = UNSUBSCRIBE (tentatively 0x42)   |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                 DSO-LENGTH (2)                |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+  \
|               SUBSCRIBE MESSAGE ID            |   > DSO-DATA
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+  /
```

Figure 3: UNSUBSCRIBE Request

6.5. DNS Push Notification RECONFIRM

Sometimes, particularly when used with a Discovery Proxy [DisProx], 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 request 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 requests to ascertain whether the target device is still present. How the Discovery Proxy causes these new Multicast DNS requests to be issued depends on the details of the underlying Multicast DNS being used. For example, a Discovery Proxy built on Apple's dns_sd.h 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 otherwise need not cause any 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 request, 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.

6.5.1. RECONFIRM Request

A RECONFIRM request begins with the standard DSO 12-byte header [DSO], followed by the RECONFIRM primary TLV. A RECONFIRM request message is illustrated in Figure 4.

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 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 which it has not yet cancelled using UNSUBSCRIBE. In the RECONFIRM response the server MUST echo back the MESSAGE ID value unchanged.

The other header fields MUST be set as described in the DSO specification [DSO]. 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 (tentatively 0x43).

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

```
                                   1  1  1  1  1  1
     0  1  2  3  4  5  6  7  8  9  0  1  2  3  4  5
   +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+  \
   |                  MESSAGE ID                   |   \
   +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+    |
   |QR| OPCODE(6) |         Z            |   RCODE   |    |
   +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+    |
   |             QDCOUNT (MUST BE ZERO)            |    |
   +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+     > HEADER
   |             ANCOUNT (MUST BE ZERO)            |    |
   +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+    |
   |             NSCOUNT (MUST BE ZERO)            |    |
   +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+    |
   |             ARCOUNT (MUST BE ZERO)            |   /
   +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+  /
   |    DSO-TYPE = RECONFIRM (tentatively 0x43)    |
   +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
   |   DSO-LENGTH (number of octets in DSO-DATA)   |
   +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+  \
   \                     NAME                      \   \
   +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+    |
   |                     TYPE                      |    |
   +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+     > DSO-DATA
   |                     CLASS                     |    |
   +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+    |
   \                     RDATA                     \   /
   +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+  /
```

Figure 4: RECONFIRM Request

The DSO-DATA for a RECONFIRM request MUST contain exactly one record. The DSO-DATA for a RECONFIRM request has no count field to specify more than one record. Since RECONFIRM requests are sent over TCP, multiple RECONFIRM request messages can be concatenated in a single TCP stream and packed efficiently into TCP segments.

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

DNS wildcarding is not supported. That is, a wildcard ("*") in a RECONFIRM message matches only a literal wildcard character ("*") in the zone, and nothing else.

Aliasing is not supported. That is, a CNAME in a RECONFIRM message matches only a literal CNAME record in the zone, and nothing else.

6.5.2. RECONFIRM Response

Each RECONFIRM request generates exactly one RECONFIRM response from the server.

A RECONFIRM response begins with the standard DSO 12-byte header [DSO], possibly followed by one or more optional TLVs, such as a Retry Delay TLV. For suggested values for the Retry Delay TLV, see Section 6.2.2.

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

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

In the RECONFIRM response the RCODE confirms receipt of the reconfirmation request. Supported RCODEs are as follows:

Mnemonic	Value	Description
NOERROR	0	RECONFIRM accepted.
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	RECONFIRM operation not supported.

Table 2: RECONFIRM Response codes

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

Nonzero RCODE values signal some kind of error.

RCODE value FORMERR indicates a message format error, for example TYPE or CLASS being ANY (255).

RCODE value SERVFAIL indicates that the server has exhausted its resources or other serious problem occurred.

RCODE values NOTIMP indicates that the server does not support DSO, and DSO is required for RECONFIRM requests.

RCODE value REFUSED indicates that the server supports RECONFIRM requests but is currently not configured to accept them from this client.

RCODE value NOTAUTH indicates that the server is not authoritative for the requested name, and can do nothing to remedy the apparent error. Note that there may be future cases in which a server is able to pass on the RECONFIRM request to the ultimate source of the information, and in these cases the server should return NOERROR.

RCODE value DSOTYPENI indicates that the server does not support RECONFIRM requests.

Nonzero RCODE values SERVFAIL, REFUSED and DSOTYPENI are benign from the client's point of view. The client may log them to aid in debugging, but otherwise they require no special action.

Nonzero RCODE values other than these three indicate a serious problem with the client. After sending an error response other than one of these three, the server SHOULD send a DSO Retry Delay TLV to end the DSO session, as described in the DSO specification [DSO].

6.6. DNS Stateful Operations TLV Context Summary

This document defines four new DSO TLVs. As suggested in Section 8.2 of the DNS Stateful Operations specification [DSO], 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 3: 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 4: 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 cancelled 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 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 [DSO].

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 cancelled.

When closing a session, a client will generally do an abortive disconnect, sending a TCP RST. This immediately discards all remaining inbound and outbound data, which is appropriate if the client no longer has any interest in this data. In the BSD Sockets API, sending a TCP RST is achieved by setting the SO_LINGER option with a time of 0 seconds and then closing the socket.

If a client has performed operations on this session that it would not want lost (like DNS updates) then the client SHOULD do an orderly disconnect, sending a TLS close_notify followed by a TCP FIN. (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.)

7. Security Considerations

The Strict Privacy Usage 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 DNS over (D)TLS Usage Profiles document [RFC8310] for additional recommendations for various versions of TLS usage.

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 provided 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 end-point of the TLS communication is authenticated as the intended entity to communicate with.

Deployment recommendations on the appropriate key lengths and cypher suites are beyond the scope of this document. Please refer to TLS Recommendations [RFC7525] 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 Sever 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 useable, 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 Session Resumption

TLS Session Resumption is permissible on DNS Push Notification servers. The server may keep TLS state with Session IDs [RFC8446] or operate in stateless mode by sending a Session Ticket [RFC5077] to the client for it to store. 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 to be published in the IANA Registry Service Types [RFC6335][ST] that is only applicable for the TCP protocol.

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

Table 5: IANA Service Type Assignments

This document also defines four new DNS Stateful Operation TLV types to be recorded in the IANA DSO Type Code Registry.

Name	Value	Definition
SUBSCRIBE	TBA (tentatively 0x40)	Section 6.2
PUSH	TBA (tentatively 0x41)	Section 6.3
UNSUBSCRIBE	TBA (tentatively 0x42)	Section 6.4
RECONFIRM	TBA (tentatively 0x43)	Section 6.5

Table 6: IANA DSO TLV Type Code Assignments

9. Acknowledgements

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

This draft has been improved due to comments from Ran Atkinson, Tim Chown, Mark Delany, Ralph Droms, Bernie Volz, Jan Komissar, Manju Shankar Rao, Markus Stenberg, Dave Thaler, Soraia Zlatkovic, Sara Dickinson, and Andrew Sullivan. Ted Lemon provided clarifying text that was greatly appreciated.

10. References

10.1. Normative References

[DSO] Bellis, R., Cheshire, S., Dickinson, J., Dickinson, S., Lemon, T., and T. Pusateri, "DNS Stateful Operations", draft-ietf-dnsop-session-signal-18 (work in progress), October 2018.

[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>.

[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>.

[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>.

[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>.

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

10.2. Informative References

[DisProx] Cheshire, S., "Discovery Proxy for Multicast DNS-Based Service Discovery", draft-ietf-dnssd-hybrid-08 (work in progress), March 2018.

[I-D.dukkipati-tcpm-tcp-loss-probe] Dukkipati, N., Cardwell, N., Cheng, Y., and M. Mathis, "Tail Loss Probe (TLP): An Algorithm for Fast Recovery of Tail Losses", draft-dukkipati-tcpm-tcp-loss-probe-01 (work in progress), February 2013.

[LLQ] Sekar, K., "DNS Long-Lived Queries", draft-sekar-dns-llq-01 (work in progress), August 2006.

[obs] "Observer Pattern", <https://en.wikipedia.org/wiki/Observer_pattern>.

[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>.

[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>.

[RFC5077] Salowey, J., Zhou, H., Eronen, P., and H. Tschofenig, "Transport Layer Security (TLS) Session Resumption without Server-Side State", RFC 5077, DOI 10.17487/RFC5077, January 2008, <https://www.rfc-editor.org/info/rfc5077>.

[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>.

[RFC6824] Ford, A., Raiciu, C., Handley, M., and O. Bonaventure, "TCP Extensions for Multipath Operation with Multiple Addresses", RFC 6824, DOI 10.17487/RFC6824, January 2013, <https://www.rfc-editor.org/info/rfc6824>.

[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>.

[RFC7525] 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, DOI 10.17487/RFC7525, May 2015, <https://www.rfc-editor.org/info/rfc7525>.

[RFC7719] Hoffman, P., Sullivan, A., and K. Fujiwara, "DNS Terminology", RFC 7719, DOI 10.17487/RFC7719, December 2015, <https://www.rfc-editor.org/info/rfc7719>.

[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>.

[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>.

[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>.

[SYN] Eddy, W., "Defenses Against TCP SYN Flooding Attacks", The Internet Protocol Journal, Cisco Systems, Volume 9, Number 4, December 2006.

[XEP0060] Millard, P., Saint-Andre, P., and R. Meijer, "Publish-Subscribe", XSF XEP 0060, July 2010.

Authors' Addresses

Tom Pusateri
Unaffiliated
Raleigh, NC 27608
USA

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

Stuart Cheshire
Apple Inc.
One Apple Park Way
Cupertino, CA 95014
USA

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