Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: January 9, 2017

T. Pusateri
Seeking affiliation
S. Cheshire
Apple Inc.

July 8, 2016

DNS Push Notifications
draft-ietf-dnssd-push-08

Abstract

The Domain Name System (DNS) was designed to return matching records efficiently for queries for data that is relatively static. When those records change frequently, DNS is still efficient at returning the updated results when polled. 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

IMPORTANT NOTE: This document currently references the EDNS(0) TCP Keepalive option [RFC7828]. As a result of discussions about this document, the community came to the realization that DNS needs explicit session-level signaling, to complement the current EDNS(0) per-message signaling. As a result, work on DNS Session Signaling [I-D.bellis-dnsop-session-signal] is underway, and this document will be updated shortly to make use of those new Session Signaling mechanisms once they are agreed.

DNS records may be updated using DNS Update [RFC2136]. Other mechanisms such as a Hybrid Proxy [I-D.ietf-dnssd-hybrid] can also generate changes to a DNS zone. This document specifies a protocol for Unicast 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 "Key words for use in RFCs to Indicate Requirement Levels" [RFC2119].

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 to state changes following the Observer design pattern. 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 result in 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 new services and updates are sent 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 Hybrid Proxy [I-D.ietf-dnssd-hybrid]) 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.

DNS Long-Lived Queries (LLQ) [I-D.sekar-dns-llq] is an existing deployed solution to provide asynchronous change notifications. Even though it can be used over TCP, LLQ is defined primarily as a UDP-based protocol, and as such it defines its own equivalents of existing TCP features like the three-way handshake. This document builds on experience gained with the LLQ protocol, with an improved design that uses long-lived TCP connections instead of UDP (and therefore doesn't need to duplicate existing TCP functionality), and adopts the syntax and semantics of DNS Update messages [RFC2136] instead of inventing a new vocabulary of messages to communicate DNS zone changes.

Because DNS Push Notifications impose a certain load on the responding server (though less load than rapid polling of that server) DNS Push Notification clients SHOULD exercise restraint in issuing DNS Push Notification subscriptions. 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 of that subscription 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). Implementations MAY want to implement idle timeouts, so that if the user ceases interacting with the device, the display showing the result of the DNS Push Notification subscription is automatically dismissed after a certain period of inactivity. For example, if a user presses the "Print" button on their phone, and then leaves the phone showing the printer discovery screen until the phone goes to sleep, then the printer discovery screen should be automatically dismissed as the device goes to sleep. If the user does still intend to print, this will require them to press the "Print" button again when they wake their phone up.

A DNS Push Notification client MUST 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 it will be really fast if the user does choose to bring up an on-screen display of that data. 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, a client SHOULD NOT keep a connection to a server open indefinitely if it has no active subscriptions on that connection. After 30 seconds with no active subscriptions the client SHOULD close the idle connection, and, if needed in the future, open a new connection.

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 zone. 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 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 master, slave, or stealth name server [RFC1996]. 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 Queries.

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.

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 connection, provided that these are queries for names falling within the server's zone (the <zone> in the "_dns-push-tls._tcp.<zone>" SRV record). The RD (Recursion Desired) bit MUST be zero.

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 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 it is today with existing DNS servers and clients. A DNS Push Notification server MAY choose to implement only DNS class "IN".

4. Transport

Implementations of 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 TLS over TCP.

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. Because TCP SYN flooding attacks are possible with any protocol over TCP, implementers are encouraged to use industry best practices to guard against such attacks [IPJ.9-4-TCPSYN] [RFC4953].

Transport Layer Security (TLS) [RFC5246] is well understood and deployed across many protocols running over TCP. It is designed to prevent eavesdropping, tampering, or 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. Additional authentication of the SRV target using DNSSEC verification and DANE TLSA records [RFC7673] is strongly encouraged. See below in Section 7.2 for details.

A DNS Push Notification session begins with a client connecting to a DNS Push Notification server. Over that connection the client then issues DNS operation requests, such as SUBSCRIBE.

4.1. Client-Initiated Termination

An individual subscription is terminated by sending an UNSUBSCRIBE message for that specific subscription, or all subscriptions can be cancelled at once by the client closing the connection. When a client terminates an individual subscription (via UNSUBSCRIBE) or all subscriptions on that connection (by closing the connection) 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 connection via UNSUBSCRIBE, a client MAY close the connection immediately, or it may keep it open if it anticipates performing further operations on that connection in the future. If a client wishes to keep an idle connection open, it MUST continue to meet its keepalive obligations [RFC7828] or the server is entitled to close the connection (see below).

If a client plans to terminate one or more subscriptions on a connection and doesn't intend to keep that connection open, then as an efficiency optimization it MAY instead choose to simply close the connection, which implicitly terminates all subscriptions on that connection. 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 connection has been cancelled.

When closing a connection, 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 connection that it would not want lost (like DNS updates) then the client SHOULD do an orderly disconnect, sending 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.

In the first SUBSCRIBE response on a connection, the server MUST include an explicit EDNS(0) TCP Keepalive option. If the first SUBSCRIBE response does not include an explicit EDNS(0) TCP Keepalive option this is an error and the client MUST immediately close the TCP connection and not attempt any further DNS Push Notification requests to that server until one hour has passed. This situation may occur

if a client connects to a server that doesn't implement DNS Push Notifications at all, and it is important not to burden such servers with continuous retries.

Upon receiving an error response from the server, a client SHOULD NOT close the connection. An error relating to one particular operation on a connection does not necessarily imply that all other operations on that connection have also failed, or that future operations will fail. The client should assume that the server will make its own decision about whether or not to close the connection, based on the server's determination of whether the error condition pertains to this particular operation, or would also apply to any subsequent operations. If the server does not close the connection then the client SHOULD continue to use that connection for subsequent operations.

Upon receiving a Termination Message from the server (see below), a client MUST immediately close the connection.

4.2. Server-Initiated Termination

If a client makes a connection and then fails to send any DNS message that uses EDNS(0) TCP Keepalive [RFC7828] (either SUBSCRIBE, where Keepalive is implicit, or some other DNS message, with an explicit an EDNS(0) TCP Keepalive option) then after 30 seconds of inactivity the server SHOULD close the connection. If no data has been sent on the connection the server MAY abort the connection with a TCP RST. If data has been sent on the connection then the server SHOULD close the connection gracefully with a TCP FIN so that the data is reliably delivered.

In the response to the first successful SUBSCRIBE, the included EDNS(0) TCP Keepalive option specifies the idle timeout so that the client knows the frequency of traffic it must generate to keep the connection alive. If the idle timeout for that connection changes, then the server communicates this by placing an updated EDNS(0) TCP Keepalive option in a subsequent message to the client.

At both servers and clients, the generation or reception of any complete request, response, update, or keepalive message resets the keepalive timer for that connection.

In the absence of any requests, responses, or update messages on a connection, a client MUST generate keepalive traffic before the idle timeout expires, or the server is entitled to close the connection.

If a client disconnects from the network abruptly, without closing its connection, the server learns of this after failing to receive further traffic from that client. If no requests, responses, update messages or keepalive traffic occurs on a connection for 1.5 times the idle timeout, then this indicates that the client is probably no longer on the network, and the server SHOULD abort the connection with a TCP RST. The time before the server closes the connection is intentionally 50% longer than the time before the client is required to generate keepalive traffic, to allow for differences in clock rate and network propagation delays.

[We need to discuss the nature of "the required keepalives". Are they TCP-layer keepalives? DNS-layer keepalives? There is currently no DNS-layer keepalive or 'no-op' operation defined. What would that operation be? A DNS QUERY containing zero questions? A DNS SUBSCRIBE containing zero questions? An "empty" DNS message over the TCP connection (just a pair of zero bytes, signifying a zero-length message)? One benefit of TCP-layer keepalives is that they transmit fewer bytes, and involve less software overhead for processing those bytes. Another benefit is that it is more feasible to implement these in networking offload hardware, which can allow devices to meet

their TCP keepalive obligations while sleeping. This is particularly important for battery-powered devices like mobile phones and tablets. On the other hand, using TCP-layer keepalives requires an API for a client to tell the networking stack at what frequency to perform TCP-layer keepalives, and an API for a server to request the networking stack to inform it when TCP-layer keepalives are not received by the required deadline. TCP-layer keepalives also only verify liveness of the remote networking stack, whereas DNS-layer keepalives provide higher assurance of liveness of the remote server application software -- though this a limited benefit, since there is no reason to expect that DNS Push Notification server software will routinely become wedged and unresponsive.]

After sending an error response to a client, the server MAY close the connection with a TCP FIN, or may allow the connection to remain open. For error conditions that only affect the single operation in question, the server SHOULD return an error response to the client and leave the connection open for further operations. For error conditions that are likely to make all operations unsuccessful in the immediate future, the server SHOULD return an error response to the client and then close the connection with a TCP FIN.

If the server is overloaded and needs to shed load, it SHOULD send a Termination Message to the client and close the connection with a TCP FIN.

Apart from the cases described above, a server MUST NOT close a connection with a DNS Push Notification client, except in extraordinary error conditions. Closing the connection is the client's responsibility, to be done at the client's discretion, when it so chooses. A DNS Push Notification server only closes a DNS Push Notification connection under exceptional circumstances, such as when the server application software or underlying operating system is restarting, the server application terminated unexpectedly (perhaps due to a bug that makes it crash), or the server is undergoing maintenance procedures. When possible, a DNS Push Notification server SHOULD send a Termination Message (Section 6.6) informing the client of the reason for the connection being closed.

After a connection is closed by the server, the client SHOULD try to reconnect, to that server, or to another server supporting DNS Push Notifications for the zone. If reconnecting to the same server, and there was a Termination Message or error response containing a EDNS(0) TCP Keepalive option, the client MUST respect the indicated delay before attempting to reconnect.

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 connection to a DNS server, each record 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 connections to alternate DNS servers that support DNS Push Notifications for the zone and distribute record subscriptions at its discretion. In this way, both clients and servers can react to resource constraints. Token bucket rate limiting schemes are also effective in providing fairness by a server across numerous client requests.

6. Protocol Operation

A DNS Push Notification exchange begins with the client discovering the appropriate server, and then making a TLS/TCP connection to it. The client may then add and remove Push Notification subscriptions over this connection. 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.

The exchange between client and server terminates when either end closes the TCP connection with a TCP FIN or RST.

A client SHOULD NOT make multiple TLS/TCP connections to the same DNS Push Notification server. A client SHOULD share a single TLS/TCP connection for all requests to the same DNS Push Notification server. This shared connection should be used for all DNS Queries and DNS Push Notification Queries queries to that server, and for DNS Update requests too when the "_dns-update-tls._tcp.<zone>" SRV record indicates that the same server also handles DNS Update requests. This is to reduce unnecessary load on the DNS Push Notification server.

For the purposes here, the determination of "same server" is made by inspecting the target hostname and port, regardless of the name being queried, or what zone if falls within. A given server may support Push Notifications (and possibly DNS Updates too) for multiple DNS zones. When a client discovers that the DNS Push Notification server (and/or DNS Update server) for several different names (including names that fall within different zones) is the same target hostname and port, the client SHOULD use a single shared TCP connection for all relevant operations on those names. A client SHOULD NOT open multiple TCP connections to the same target host and port just because the names being queried (or updated) happen to fall within different zones.

Note that the "same server" determination described here is made using the target hostname given in the SRV record, not the IP address(es) that the hostname resolves to. If two different target hostnames happen to resolve to the same IP address(es), then the client SHOULD NOT recognize these as the "same server" for the purposes of using a single shared connection to that server. If an administrator wishes to use a single server for multiple zones and/or multiple roles (e.g., both DNS Push Notifications and DNS Updates),

and wishes to have clients use a single shared connection for operations on that server, then the administrator MUST use the same target hostname in the appropriate SRV records.

However, server implementers and operators should be aware that this connection sharing may not be possible in all cases. A single client device may be home to multiple independent client software instances that don't know about each other, so a DNS Push Notification server MUST be prepared to accept multiple connections from the same client IP address. This is undesirable from an efficiency standpoint, but may be unavoidable in some situations, so a DNS Push Notification server MUST be prepared to accept multiple connections from the same client IP address.

Clients SHOULD silently ignore unrecognized messages (both requests and responses) over the TLS/TCP connection. For example, UNSUBSCRIBE and RECONFIRM currently generate no response, but if future versions of this specification change that, existing clients SHOULD silently ignore these unexpected responses. This allows for backwards compatibility with future enhancements.

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 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 [I-D.ietf-dprive-dns-over-tls].

1. The client begins the discovery by sending a DNS query to the local resolver with record type SOA [RFC1035] for the name of the record it wishes to subscribe.

2. If the SOA record exists, it MUST be returned in the Answer Section of the response. If not, the local resolver SHOULD include the SOA record for the zone of the requested name in the Authority Section.

3. If no SOA record is returned, the client then strips off the leading label from the requested name. If the resulting name has at least one label in it, the client sends a new SOA query and processing continues at step 2 above. If the resulting name is empty (the root label) then this is a network configuration error and the client gives up. The client MAY retry the operation at a later time.

4. 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.

5. If the zone in question does not offer DNS Push Notifications then SRV record MUST NOT exist and the SRV query will return a negative answer.

6. If the zone in question is set up to offer DNS Push Notifications then this SRV record MUST exist. 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 [RFC7673].

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 reachable or is not willing to accept a subscription request, then a subsequent server is to be contacted.

Each time a client makes a new DNS Push Notification subscription connection, it SHOULD repeat the discovery process in order to determine the preferred DNS server for subscriptions at that time.

6.2. DNS Push Notification SUBSCRIBE

A DNS Push Notification client indicates its desire to receive DNS Push Notifications for a given domain name by sending a SUBSCRIBE request over the established TCP connection to the server. A SUBSCRIBE request is formatted identically to a conventional DNS QUERY request [RFC1035], except that the opcode is SUBSCRIBE (6) instead of QUERY (0). If neither QTYPE nor QCLASS are ANY (255) then this is a specific subscription to changes for the given name, type and class. If one or both of QTYPE or QCLASS 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, were 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 DNS SUBSCRIBE, QTYPE and QCLASS 255 should be interpreted to mean "ALL", not "ANY". After accepting a subscription where one or both of QTYPE or QCLASS are 255, the server MUST send Push Notification Updates for ALL record changes that match the subscription, not just some of them.

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

The AA, TC, RD, RA, Z, AD, and CD bits, and the RCODE field, MUST be zero on transmission, and MUST be silently ignored on reception.

The ID field may be set to any value of the client's choosing, and the server MUST echo this value back in the response message. The client is not required to select unique ID values; it is permissible to use the same value (e.g., zero) for all operations. Since the name, qtype, and qclass are sufficient to uniquely identify a SUBSCRIBE operation on a connection, the name, qtype, and qclass in a SUBSCRIBE response are sufficient to correlate a response with its corresponding request. However, for convenience, the client may put any value it chooses in the ID field of the SUBSCRIBE request, and the server MUST echo that value back unchanged in the SUBSCRIBE response. Note that the ID field of Push Notification Update Messages is always zero, since a Push Notification Update Message could potentially match more than one subscription, or could relate to a subscription that the client has just cancelled with an UNSUBSCRIBE message.

Like a DNS QUERY request, a SUBSCRIBE request MUST contain exactly one question. Since SUBSCRIBE requests are sent over TCP, multiple SUBSCRIBE requests can be concatenated in a single TCP stream and packed efficiently into TCP segments, so the ability to pack multiple SUBSCRIBE operations into a single DNS message within that TCP stream would add extra complexity for little benefit.

ANCOUNT MUST be zero, and the Answer Section MUST be empty. Any records in the Answer Section MUST be silently ignored.

NSCOUNT MUST be zero, and the Authority Section MUST be empty. Any records in the Authority Section MUST be silently ignored.

ARCOUNT specifies the number of records in the Additional Data Section. Typically this is zero, but it may be nonzero in some cases, such as when the request includes an EDNS(0) OPT record.

If accepted, the subscription will stay in effect until the client revokes the subscription or until the connection between the client and the server is closed.

SUBSCRIBE requests on a given connection 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 TLS/TCP connection. For the purpose of this matching, the established DNS case-insensitivity for US-ASCII letters applies (e.g., "foo.com" and "Foo.com" are the same). If a server receives such a duplicate SUBSCRIBE message this is an error and the server MUST immediately close the TCP connection.

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 accept these requests and send Push Notifications if and when matches are found in the future.

Since all SUBSCRIBE operations are implicitly long-lived operations, the server MUST interpret a SUBSCRIBE request as if it contained an EDNS(0) TCP Keepalive option [RFC7828]. A client MUST NOT include an actual EDNS(0) TCP Keepalive option in the request, since it is automatic, and implied by the semantics of SUBSCRIBE. If a server

receives a SUBSCRIBE request that does contain an actual EDNS(0) TCP Keepalive option this is an error and the server MUST immediately close the TCP connection.

A SUBSCRIBE operation MAY include an explicit EDNS(0) [RFC6891] OPT record where necessary to carry additional EDNS(0) information other than a TCP Keepalive option.

The presence of a SUBSCRIBE operation on a connection indicates to the server that the client fully implements EDNS(0) [RFC6891], and can correctly understand any response that conforms to that specification. After receiving a SUBSCRIBE request, the server MAY include OPT record in any of its responses, as needed.

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

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

The AA, TC, RD, RA, Z, AD, and CD bits, MUST be zero on transmission, and MUST be silently ignored on reception.

The ID field MUST echo the value given in the ID field of the SUBSCRIBE request.

The Question Section MUST echo back the values provided by the client in the SUBSCRIBE request that generated this SUBSCRIBE response.

ANCOUNT MUST be zero, and the Answer Section MUST be empty.
Any records in the Answer Section MUST be silently ignored.
If the subscription was accepted and there are positive answers for the requested name, type and class, then these positive answers MUST be communicated to the client in an immediately following Push Notification Update, not in the Answer Section of the SUBSCRIBE response. This simplifying requirement is made so that there is only a single way that information is communicated to a DNS Push Notification client. Since a DNS Push Notification client has to parse information received via Push Notification Updates anyway, it is simpler if it does not also have to parse information received via the Answer Section of a SUBSCRIBE response.

NSCOUNT MUST be zero, and the Authority Section MUST be empty.
Any records in the Authority Section MUST be silently ignored.

ARCOUNT specifies the number of records in the Additional Data Section, e.g., the EDNS(0) OPT record.

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 resource exhaustion.
NXDOMAIN	3	NOT APPLICABLE. DNS Push Notification MUST NOT return NXDOMAIN errors in response to SUBSCRIBE requests.
NOTIMP	4	Server does not implement DNS Push Notifications.
REFUSED	5	Server refuses to process request for policy or security reasons.
NOTAUTH	9	Server is not authoritative for the requested name.

SUBSCRIBE Response codes

This document specifies only these RCODE values for SUBSCRIBE Responses. Servers sending SUBSCRIBE Responses SHOULD use one of these values. 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 RCODE value.

In the first SUBSCRIBE response on a connection, the server MUST include an explicit EDNS(0) TCP Keepalive option. If the first SUBSCRIBE response does not include an explicit EDNS(0) TCP Keepalive option this is an error and the client MUST immediately close the TCP connection. In this case the client should act as if the response contained an EDNS(0) TCP Keepalive option with a value of one hour, and not attempt any further DNS Push Notification requests to that server until one hour has passed. This situation may occur if a client connects to a server that doesn't implement DNS Push Notifications at all, and it is important not to burden such servers with continuous retries.

The server MAY include EDNS(0) TCP Keepalive options in subsequent messages, if the idle timeout changes. If the client receives subsequent messages that do not contain an explicit EDNS(0) TCP Keepalive option then the idle timeout for that connection remains unchanged at that time.

In an error response, with nonzero RCODE, the server MUST contain an EDNS(0) TCP Keepalive option specifying the delay before the client submits further requests to this server:

For RCODE = 1 (FORMERR) the delay may be any value selected by the implementer. A value of one minute is RECOMMENDED, to avoid high load from defective clients.

For RCODE = 2 (SERVFAIL), which occurs due to resource exhaustion, 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.

For RCODE = 4 (NOTIMP), which occurs on a server that 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. Note that a server that doesn't implement DNS Push Notifications will most likely not implement this retry delay mechanism using the EDNS(0) TCP Keepalive option either, and in this case the client will fall back to the case described above specifying how to handle SUBSCRIBE responses that do not contain an EDNS(0) TCP Keepalive option.

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.
This is a misconfiguration, since this server is listed in a "_dns-push-tls._tcp.<zone>" SRV record, but the server itself is not currently configured to support DNS Push Notifications. 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.
This is a misconfiguration, since this server is listed in a "_dns-push-tls._tcp.<zone>" SRV record, but the server itself is not currently configured to support DNS Push Notifications for that zone. 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 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 close the TCP connection with a FIN, or MAY allow it to remain open, depending on the nature of the error. Clients MUST correctly handle both cases.

6.3. DNS Push Notification UNSUBSCRIBE

To cancel an individual subscription without closing the entire connection, the client sends an UNSUBSCRIBE message over the established TCP connection to the server. The UNSUBSCRIBE message is formatted identically to the SUBSCRIBE message which created the subscription, with the exact same name, type and class, except that the opcode is UNSUBSCRIBE (7) instead of SUBSCRIBE (6).

A client MUST NOT send an UNSUBSCRIBE message that does not exactly match the name, type and class of an existing active subscription on that TLS/TCP connection. If a server receives such an UNSUBSCRIBE message this is an error and the server MUST immediately close the connection.

No response message is generated as a result of processing an UNSUBSCRIBE message.

Having being successfully revoked with a correctly-formatted UNSUBSCRIBE message, the previously referenced subscription is no longer active and the server MAY discard the state associated with it immediately, or later, at the server's discretion.

6.4. DNS Push Notification Update Messages

Once a subscription has been successfully established, the server generates Push Notification Updates to send to the client as appropriate. An initial Push Notification Update will be sent immediately in the case that the answer set was non-empty at the moment the subscription was established. Subsequent changes to the answer set are then communicated to the client in subsequent Push Notification Updates.

The format of Push Notification Updates borrows from the existing DNS Update [RFC2136] protocol, with some simplifications.

The following figure shows the existing DNS Update header format:

```
                                1  1  1  1  1  1
  0  1  2  3  4  5  6  7  8  9  0  1  2  3  4  5
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                      ID                       |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|QR|   Opcode  |          Z          |   RCODE   |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                    ZOCOUNT                    |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                    PRCOUNT                    |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                    UPCOUNT                    |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                    ADCOUNT                    |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

Figure 1

For DNS Push Notifications the following rules apply:

The QR bit MUST be zero, and the Opcode MUST be UPDATE (5). Messages received where this is not true are not Push Notification Update Messages and should be silently ignored for the purposes of Push Notification Update Message handling.

ID, the Z bits, and RCODE MUST be zero on transmission, and MUST be silently ignored on reception.

ZOCOUNT MUST be zero, and the Zone Section MUST be empty. Any records in the Zone Section MUST be silently ignored.

PRCOUNT MUST be zero, and the Prerequisite Section MUST be empty. Any records in the Prerequisite Section MUST be silently ignored.

UPCOUNT specifies the number of records in the Update Section.

ADCOUNT specifies the number of records in the Additional Data Section. Typically this is zero, but it may be nonzero in some cases, such as when the message includes an EDNS(0) OPT record.

The Update Section contains the relevant change information for the client, formatted identically to a DNS Update [RFC2136]. To recap:

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.

When processing the records received in a Push Notification Update Message, the receiving client MUST validate that the records being added or deleted correspond with at least one currently active subscription on that connection. 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. If the QTYPE in the SUBSCRIBE request was not ANY (255) then the TYPE of the record must match the QTYPE given in the SUBSCRIBE request. If the QCLASS in the SUBSCRIBE request was not ANY (255) then the CLASS of the record must match the QCLASS given in the SUBSCRIBE request. If a matching active subscription on that connection 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 TCP connection is not closed. This is to allow for the race condition where a client sends an outbound UNSUBSCRIBE while inbound Push Notification Updates 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 update is sent. For example, an update adding a given record may match both a SUBSCRIBE request with the same QTYPE and a different SUBSCRIBE request with QTYPE=ANY. It is not the case that two updates 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" update, not three separate "delete an individual RR from a name" updates. 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" update, not two separate "delete an RRset from a name" updates.

A server SHOULD combine multiple change notifications in a single Update Message when possible, even if those change notifications apply to different subscriptions. Conceptually, a Push Notification Update Message is a connection-level concept, not a subscription-level concept.

Push Notification Update Messages MAY contain an EDNS(0) TCP Keepalive option [RFC7828] if the idle timeout has changed since the last time the server sent an EDNS(0) TCP Keepalive option on this connection.

In the event that the server wishes to inform a client of a new idle timeout for the connection, the server MAY combine that with the next message it sends to the client, or the server MAY send an empty Push Notification Update Message (zero records in the Update Section) to carry the EDNS(0) TCP Keepalive option. Clients MUST correctly receive and process the EDNS(0) TCP Keepalive option in both cases.

Reception of a Push Notification Update Message does not directly generate a response back to the server. (Updates may indirectly generate other operations; e.g., a Push Notification Update Message declaring the appearance of a PTR record could lead to a query for the SRV record named in the rdata of that PTR record[RFC6763].

The TTL of an added record is stored by the client and decremented as time passes, with the caveat that for as long as a relevant subscription is active, the TTL does not decrement below 1 second. 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 TCP connection being closed) record aging resumes and records are removed from the local cache when their TTL reaches zero.

6.5. DNS RECONFIRM

Sometimes, particularly when used with a Hybrid Proxy [I-D.ietf-dnssd-hybrid], a DNS Zone may contain stale data. When a client encounters data that it believe may be stale (e.g., an SRV record referencing a target host+port that is not responding to connection requests) the client sends a DNS RECONFIRM message to request that the server re-verify that the data is still valid. For a Hybrid Proxy, this causes it to issue new Multicast DNS requests to ascertain whether the target device is still present. For other kinds of DNS server the RECONFIRM operation is currently undefined and SHOULD be silently ignored.

A RECONFIRM request is formatted similarly to a conventional DNS QUERY request [RFC1035], except that the opcode is RECONFIRM (8) instead of QUERY (0). QTYPE MUST NOT be the value ANY (255). QCLASS MUST NOT be the value ANY (255).

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

The AA, TC, RD, RA, Z, AD, and CD bits, the ID field, and the RCODE field, MUST be zero on transmission, and MUST be silently ignored on reception.

Like a DNS QUERY request, a RECONFIRM request MUST contain exactly one question. Since RECONFIRM requests are sent over TCP, multiple RECONFIRM requests can be concatenated in a single TCP stream and packed efficiently into TCP segments, so the ability to pack multiple RECONFIRM operations into a single DNS message within that TCP stream would add extra complexity for little benefit.

ANCOUNT MUST be nonzero, and the Answer Section MUST contain the rdata for the record(s) that the client believes to be in doubt.

NSCOUNT MUST be zero, and the Authority Section MUST be empty.
Any records in the Authority Section MUST be silently ignored.

ARCOUNT specifies the number of records in the Additional Data Section. Typically this is zero, but it may be nonzero in some cases, such as when the request includes an EDNS(0) OPT record.

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

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

No response message is generated as a result of processing a RECONFIRM message.

If the server receiving the RECONFIRM request determines that the records are in fact no longer valid, then subsequent DNS Push Notification Update Messages will be generated to inform interested clients. Thus, one client discovering that a previously-advertised printer is no longer present has the side effect of informing all other interested clients that the printer in question is now gone.

6.6. DNS Push Notification Termination Message

If a server is low on resources it MAY simply terminate a client connection with a TCP RST. However, the likely behaviour of the client may be simply to reconnect immediately, putting more burden on the server. Therefore, a server SHOULD instead choose to shed client load by (a) sending a DNS Push Notification Termination Message and then (b) immediately closing the client connection with a TCP FIN instead of RST, thereby facilitating reliable delivery of the Termination Message. Upon successful reception of the Termination Message the client is expected to close the connection. The server SHOULD set a timer and, if the client has not closed the connection within a reasonable time, the server SHOULD then terminate the TCP connection with a TCP RST. The RECOMMENDED time the server should wait before terminating the TCP connection with a TCP RST is ten seconds.

The format of a Termination Message is similar to a Push Notification Update.

The following figure shows the existing DNS Update header format:

```
                                  1  1  1  1  1  1
    0  1  2  3  4  5  6  7  8  9  0  1  2  3  4  5
  +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
  |                      ID                       |
  +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
  |QR|   Opcode  |          Z         |   RCODE   |
  +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
  |                    ZOCOUNT                    |
  +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
  |                    PRCOUNT                    |
  +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
  |                    UPCOUNT                    |
  +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
  |                    ADCOUNT                    |
  +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

Figure 2

For Termination Messages the following rules apply:

The QR bit MUST be zero, and the Opcode MUST be UPDATE (5).
Messages received where this is not true are not Termination Messages
and should be silently ignored.

ID and the Z bits MUST be zero on transmission,
and MUST be silently ignored on reception.

ZOCOUNT MUST be zero, and the Zone Section MUST be empty.
Any records in the Zone Section MUST be silently ignored.

PRCOUNT MUST be zero, and the Prerequisite Section MUST be empty.
Any records in the Prerequisite Section MUST be silently ignored.

UPCOUNT MUST be zero, and the Update Section MUST be empty.
Any records in the Update Section MUST be silently ignored.

ADCOUNT specifies the number of records in the Additional Data
Section, e.g., the EDNS(0) OPT record..

The RCODE MUST contain a nonzero code giving the reason for
termination, as indicated below:

Mnemonic	Value	Description
SERVFAIL	2	The server is overloaded due to resource exhaustion.
REFUSED	5	The server has been reconfigured and is no longer accepting DNS Push Notification requests for one or more of the currently subscribed names.

Termination Response codes

This document specifies only these two RCODE values for Termination Messages. Servers sending Termination Messages SHOULD use one of these two values. However, future circumstances may create situations where other RCODE values are appropriate in Termination Messages, so clients MUST be prepared to accept Termination Messages with any RCODE value. In particular, a Termination Message with RCODE value zero (NOERROR) is still a Termination Message and should be treated as such.

The Termination Message MUST contain an EDNS(0) TCP Keepalive option [RFC7828]. The client MUST wait for the time indicated in the EDNS(0) TCP Keepalive option's idle timeout before attempting any new connections to this server. A client that receives a Termination Message without an EDNS(0) TCP Keepalive option SHOULD treat it as equivalent to a TCP Keepalive option with a zero timeout value.

In the case where the server is rejecting some, but not all, of the existing subscriptions (perhaps because it has been reconfigured and is no longer authoritative for those names) with a REFUSED (5) RCODE, the EDNS(0) TCP Keepalive option's idle timeout MAY be zero, indicating that the client SHOULD attempt to re-establish its subscriptions immediately.

In the case where a server is terminating a large number of connections at once (e.g., if the system is restarting) and the server doesn't want to be inundated with a flood of simultaneous retries, it SHOULD send different EDNS(0) TCP Keepalive values to each client. These adjustments MAY be selected randomly, pseudorandomly, or deterministically (e.g., incrementing the time value by one tenth of a second for each successive client, yielding a post-restart reconnection rate of ten clients per second).

7. Security Considerations

TLS support is REQUIRED in DNS Push Notifications. There is no provision for opportunistic encryption using a mechanism like "STARTTLS".

DNSSEC is RECOMMENDED for DNS Push Notifications. 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.

7.3. TLS Compression

In order to reduce the chances of compression related attacks, TLS-level compression SHOULD be disabled when using TLS versions 1.2 and earlier. In the draft version of TLS 1.3 [I-D.ietf-tls-tls13], TLS-level compression has been removed completely.

7.4. TLS Session Resumption

TLS Session Resumption is permissible on DNS Push Notification servers. The server may keep TLS state with Session IDs [RFC5246] or operate in stateless mode by sending a Session Ticket [RFC5077] to the client for it to store. However, once the connection is closed, any existing subscriptions will be dropped. 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 connection. Use of TLS Session Resumption allows a new 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 the service name: "_dns-push-tls._tcp".
It is only applicable for the TCP protocol.
This name is to be published in the IANA Service Name Registry.

This document defines three DNS OpCodes: SUBSCRIBE with (tentative) value 6, UNSUBSCRIBE with (tentative) value 7, and RECONFIRM with (tentative) value 8.

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, and Soraia Zlatkovic.

10. References

10.1. Normative References

[I-D.bellis-dnsop-session-signal]
Bellis, R., Cheshire, S., Marcon, J., Mankin, A., and T. Pusateri, "DNS Session Signaling", draft-bellis-dnsop-session-signal-00 (work in progress), July 2016.

[I-D.ietf-tls-tls13]
Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", draft-ietf-tls-tls13-13 (work in progress), May 2016.

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

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

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

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

[RFC1123] Braden, R., Ed., "Requirements for Internet Hosts - Application and Support", STD 3, RFC 1123, DOI 10.17487/RFC1123, October 1989, <http://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, <http://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, <http://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, <http://www.rfc-editor.org/info/rfc2782>.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <http://www.rfc-editor.org/info/rfc5246>.

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

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

[RFC6895] Eastlake 3rd, D., "Domain Name System (DNS) IANA Considerations", BCP 42, RFC 6895, DOI 10.17487/RFC6895, April 2013, <http://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, <http://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, <http://www.rfc-editor.org/info/rfc7766>.

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

10.2. Informative References

[I-D.ietf-dnssd-hybrid]
Cheshire, S., "Hybrid Unicast/Multicast DNS-Based Service Discovery", draft-ietf-dnssd-hybrid-03 (work in progress), November 2015.

[I-D.ietf-dprive-dns-over-tls]
Zi, Z., Zhu, L., Heidemann, J., Mankin, A., Wessels, D., and P. Hoffman, "Specification for DNS over TLS", draft-ietf-dprive-dns-over-tls-09 (work in progress), March 2016.

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

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

[RFC1996] Vixie, P., "A Mechanism for Prompt Notification of Zone Changes (DNS NOTIFY)", RFC 1996, DOI 10.17487/RFC1996, August 1996, <http://www.rfc-editor.org/info/rfc1996>.

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

[RFC4953] Touch, J., "Defending TCP Against Spoofing Attacks", RFC 4953, DOI 10.17487/RFC4953, July 2007, <http://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, <http://www.rfc-editor.org/info/rfc5077>.

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

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

[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, <http://www.rfc-editor.org/info/rfc7525>.

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

Authors' Addresses

Tom Pusateri
Seeking affiliation
Hilton Head Island, SC
USA

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

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

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