

Internet Engineering Task Force  
Internet-Draft  
Intended status: Standards Track  
Expires: September 25, 2019

S. Cheshire  
Apple Inc.  
March 24, 2019

Discovery Proxy for Multicast DNS-Based Service Discovery  
draft-ietf-dnssd-hybrid-10

Abstract

This document specifies a network proxy that uses Multicast DNS to automatically populate the wide-area unicast Domain Name System namespace with records describing devices and services found on the local link.

Status of This Memo

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

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

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

This Internet-Draft will expire on September 25, 2019.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	3
2. Operational Analogy . . . . .	6
3. Conventions and Terminology Used in this Document . . . . .	7
4. Compatibility Considerations . . . . .	7
5. Discovery Proxy Operation . . . . .	8
5.1. Delegated Subdomain for Service Discovery Records . . . . .	9
5.2. Domain Enumeration . . . . .	11
5.2.1. Domain Enumeration via Unicast Queries . . . . .	11
5.2.2. Domain Enumeration via Multicast Queries . . . . .	13
5.3. Delegated Subdomain for LDH Host Names . . . . .	14
5.4. Delegated Subdomain for Reverse Mapping . . . . .	16
5.5. Data Translation . . . . .	18
5.5.1. DNS TTL limiting . . . . .	18
5.5.2. Suppressing Unusable Records . . . . .	19
5.5.3. NSEC and NSEC3 queries . . . . .	20
5.5.4. No Text Encoding Translation . . . . .	20
5.5.5. Application-Specific Data Translation . . . . .	21
5.6. Answer Aggregation . . . . .	23
6. Administrative DNS Records . . . . .	27
6.1. DNS SOA (Start of Authority) Record . . . . .	27
6.2. DNS NS Records . . . . .	28
6.3. DNS Delegation Records . . . . .	28
6.4. DNS SRV Records . . . . .	29
7. DNSSEC Considerations . . . . .	30
7.1. On-line signing only . . . . .	30
7.2. NSEC and NSEC3 Records . . . . .	30
8. IPv6 Considerations . . . . .	31
9. Security Considerations . . . . .	32
9.1. Authenticity . . . . .	32
9.2. Privacy . . . . .	32
9.3. Denial of Service . . . . .	32
10. IANA Considerations . . . . .	33
11. Acknowledgments . . . . .	33
12. References . . . . .	34
12.1. Normative References . . . . .	34
12.2. Informative References . . . . .	35
Appendix A. Implementation Status . . . . .	38
A.1. Already Implemented and Deployed . . . . .	38
A.2. Already Implemented . . . . .	38
A.3. Partially Implemented . . . . .	39
Author's Address . . . . .	39

## 1. Introduction

Multicast DNS [RFC6762] and its companion technology DNS-based Service Discovery [RFC6763] were created to provide IP networking with the ease-of-use and autoconfiguration for which AppleTalk was well known [RFC6760] [ZC] [Roadmap].

For a small home network consisting of just a single link (or a few physical links bridged together to appear as a single logical link from the point of view of IP) Multicast DNS [RFC6762] is sufficient for client devices to look up the ".local" host names of peers on the same home network, and to use Multicast DNS-Based Service Discovery (DNS-SD) [RFC6763] to discover services offered on that home network.

For a larger network consisting of multiple links that are interconnected using IP-layer routing instead of link-layer bridging, link-local Multicast DNS alone is insufficient because link-local Multicast DNS packets, by design, are not propagated onto other links.

Using link-local multicast packets for Multicast DNS was a conscious design choice [RFC6762]. Even when limited to a single link, multicast traffic is still generally considered to be more expensive than unicast, because multicast traffic impacts many devices, instead of just a single recipient. In addition, with some technologies like Wi-Fi [IEEE-11], multicast traffic is inherently less efficient and less reliable than unicast, because Wi-Fi multicast traffic is sent at lower data rates, and is not acknowledged [Mcast]. Increasing the amount of expensive multicast traffic by flooding it across multiple links would make the traffic load even worse.

Partitioning the network into many small links curtails the spread of expensive multicast traffic, but limits the discoverability of services. At the opposite end of the spectrum, using a very large local link with thousands of hosts enables better service discovery, but at the cost of larger amounts of multicast traffic.

Performing DNS-Based Service Discovery using purely Unicast DNS is more efficient and doesn't require large multicast domains, but does require that the relevant data be available in the Unicast DNS namespace. The Unicast DNS namespace in question could fall within a traditionally assigned globally unique domain name, or could use a private local unicast domain name such as ".home.arpa" [RFC8375].

In the DNS-SD specification [RFC6763], Section 10 ("Populating the DNS with Information") discusses various possible ways that a service's PTR, SRV, TXT and address records can make their way into the Unicast DNS namespace, including manual zone file configuration

[RFC1034] [RFC1035], DNS Update [RFC2136] [RFC3007] and proxies of various kinds.

Making the relevant data available in the Unicast DNS namespace by manual DNS configuration is one option. This option has been used for many years at IETF meetings to advertise the IETF Terminal Room printer. Details of this example are given in Appendix A of the Roadmap document [Roadmap]. However, this manual DNS configuration is labor intensive, error prone, and requires a reasonable degree of DNS expertise.

Populating the Unicast DNS namespace via DNS Update by the devices offering the services themselves is another option [RegProt] [DNS-UL]. However, this requires configuration of DNS Update keys on those devices, which has proven onerous and impractical for simple devices like printers and network cameras.

Hence, to facilitate efficient and reliable DNS-Based Service Discovery, a compromise is needed that combines the ease-of-use of Multicast DNS with the efficiency and scalability of Unicast DNS.

This document specifies a type of proxy called a "Discovery Proxy" that uses Multicast DNS [RFC6762] to discover Multicast DNS records on its local link, and makes corresponding DNS records visible in the Unicast DNS namespace.

In principle, similar mechanisms could be defined using other local service discovery protocols, to discover local information and then make corresponding DNS records visible in the Unicast DNS namespace. Such mechanisms for other local service discovery protocols could be addressed in future documents.

The design of the Discovery Proxy is guided by the previously published requirements document [RFC7558].

In simple terms, a descriptive DNS name is chosen for each link in an organization. Using a DNS NS record, responsibility for that DNS name is delegated to a Discovery Proxy physically attached to that link. Now, when a remote client issues a unicast query for a name falling within the delegated subdomain, the normal DNS delegation mechanism results in the unicast query arriving at the Discovery Proxy, since it has been declared authoritative for those names. Now, instead of consulting a textual zone file on disk to discover the answer to the query, as a traditional DNS server would, a Discovery Proxy consults its local link, using Multicast DNS, to find the answer to the question.

For fault tolerance reasons there may be more than one Discovery Proxy serving a given link.

Note that the Discovery Proxy uses a "pull" model. The local link is not queried using Multicast DNS until some remote client has requested that data. In the idle state, in the absence of client requests, the Discovery Proxy sends no packets and imposes no burden on the network. It operates purely "on demand".

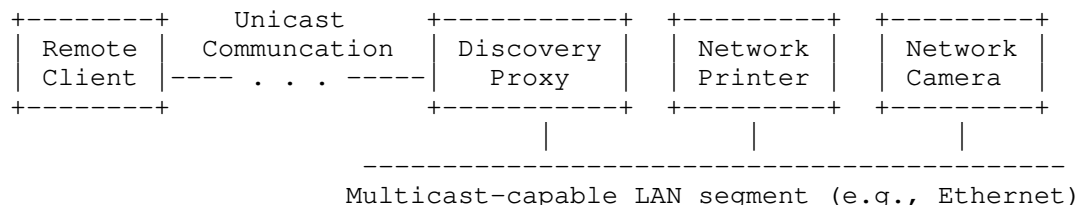
An alternative proposal that has been discussed is a proxy that performs DNS updates to a remote DNS server on behalf of the Multicast DNS devices on the local network. The difficulty with this is that Multicast DNS devices do not routinely announce their records on the network. Generally they remain silent until queried. This means that the complete set of Multicast DNS records in use on a link can only be discovered by active querying, not by passive listening. Because of this, a proxy can only know what names exist on a link by issuing queries for them, and since it would be impractical to issue queries for every possible name just to find out which names exist and which do not, there is no reasonable way for a proxy to programmatically learn all the answers it would need to push up to the remote DNS server using DNS Update. Even if such a mechanism were possible, it would risk generating high load on the network continuously, even when there are no clients with any interest in that data.

Hence, having a model where the query comes to the Discovery Proxy is much more efficient than a model where the Discovery Proxy pushes the answers out to some other remote DNS server.

A client seeking to discover services and other information achieves this by sending traditional DNS queries to the Discovery Proxy, or by sending DNS Push Notification subscription requests [Push].

How a client discovers what domain name(s) to use for its service discovery queries, (and consequently what Discovery Proxy or Proxies to use) is described in Section 5.2.

The diagram below illustrates a network topology using a Discovery Proxy to provide discovery service to a remote client.



## 2. Operational Analogy

A Discovery Proxy does not operate as a multicast relay, or multicast forwarder. There is no danger of multicast forwarding loops that result in traffic storms, because no multicast packets are forwarded. A Discovery Proxy operates as a *\*proxy\** for a remote client, performing queries on its behalf and reporting the results back.

A reasonable analogy is making a telephone call to a colleague at your workplace and saying, "I'm out of the office right now. Would you mind bringing up a printer browser window and telling me the names of the printers you see?" That entails no risk of a forwarding loop causing a traffic storm, because no multicast packets are sent over the telephone call.

A similar analogy, instead of enlisting another human being to initiate the service discovery operation on your behalf, is to log into your own desktop work computer using screen sharing, and then run the printer browser yourself to see the list of printers. Or log in using ssh and type "dns-sd -B \_ipp.\_tcp" and observe the list of discovered printer names. In neither case is there any risk of a forwarding loop causing a traffic storm, because no multicast packets are being sent over the screen sharing or ssh connection.

The Discovery Proxy provides another way of performing remote queries, except using a different protocol instead of screen sharing or ssh.

When the Discovery Proxy software performs Multicast DNS operations, the exact same Multicast DNS caching mechanisms are applied as when any other client software on that Discovery Proxy device performs Multicast DNS operations, whether that be running a printer browser client locally, or a remote user running the printer browser client via a screen sharing connection, or a remote user logged in via ssh running a command-line tool like "dns-sd", or a remote user sending DNS requests that cause a Discovery Proxy to perform discovery operations on its behalf.

### 3. Conventions and Terminology Used in this Document

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", when, and only when, they appear in all capitals, as shown here [RFC2119] [RFC8174].

The Discovery Proxy builds on Multicast DNS, which works between hosts on the same link. For the purposes of this document a set of hosts is considered to be "on the same link" if:

- o when any host from that set sends a packet to any other host in that set, using unicast, multicast, or broadcast, the entire link-layer packet payload arrives unmodified, and
- o a broadcast sent over that link, by any host from that set of hosts, can be received by every other host in that set.

The link-layer *\*header\** may be modified, such as in Token Ring Source Routing [IEEE-5], but not the link-layer *\*payload\**. In particular, if any device forwarding a packet modifies any part of the IP header or IP payload then the packet is no longer considered to be on the same link. This means that the packet may pass through devices such as repeaters, bridges, hubs or switches and still be considered to be on the same link for the purpose of this document, but not through a device such as an IP router that decrements the IP TTL or otherwise modifies the IP header.

### 4. Compatibility Considerations

No changes to existing devices are required to work with a Discovery Proxy.

Existing devices that advertise services using Multicast DNS work with Discovery Proxy.

Existing clients that support DNS-Based Service Discovery over Unicast DNS work with Discovery Proxy. Service Discovery over Unicast DNS was introduced in Mac OS X 10.4 in April 2005, as is included in Apple products introduced since then, including iPhone and iPad, as well as products from other vendors, such as Microsoft Windows 10.

An overview of the larger collection of related Service Discovery technologies, and how Discovery Proxy relates to those, is given in the Service Discovery Road Map document [Roadmap].

## 5. Discovery Proxy Operation

In a typical configuration, a Discovery Proxy is configured to be authoritative [RFC1034] [RFC1035] for four or more DNS subdomains, and authority for these subdomains is delegated to it via NS records:

A DNS subdomain for service discovery records.

This subdomain name may contain rich text, including spaces and other punctuation. This is because this subdomain name is used only in graphical user interfaces, where rich text is appropriate.

A DNS subdomain for host name records.

This subdomain name SHOULD be limited to letters, digits and hyphens, to facilitate convenient use of host names in command-line interfaces.

One or more DNS subdomains for IPv4 Reverse Mapping records.

These subdomains will have names that ends in "in-addr.arpa."

One or more DNS subdomains for IPv6 Reverse Mapping records.

These subdomains will have names that ends in "ip6.arpa."

In an enterprise network the naming and delegation of these subdomains is typically performed by conscious action of the network administrator. In a home network naming and delegation would typically be performed using some automatic configuration mechanism such as HNCP [RFC7788].

These three varieties of delegated subdomains (service discovery, host names, and reverse mapping) are described below in Section 5.1, Section 5.3 and Section 5.4.

How a client discovers where to issue its service discovery queries is described below in Section 5.2.



### 5.1. Delegated Subdomain for Service Discovery Records

In its simplest form, each link in an organization is assigned a unique Unicast DNS domain name, such as "Building 1.example.com" or "2nd Floor.Building 3.example.com". Grouping multiple links under a single Unicast DNS domain name is to be specified in a future companion document, but for the purposes of this document, assume that each link has its own unique Unicast DNS domain name. In a graphical user interface these names are not displayed as strings with dots as shown above, but something more akin to a typical file browser graphical user interface (which is harder to illustrate in a text-only document) showing folders, subfolders and files in a file system.

*example.com*	Building 1	1st Floor	Alice's printer
	Building 2	*2nd Floor*	Bob's printer
	*Building 3*	3rd Floor	Charlie's printer
	Building 4	4th Floor	
	Building 5		
	Building 6		

Figure 1: Illustrative GUI

Each named link in an organization has one or more Discovery Proxies which serve it. This Discovery Proxy function for each link could be performed by a device like a router or switch that is physically attached to that link. In the parent domain, NS records are used to delegate ownership of each defined link name

(e.g., "Building 1.example.com") to the one or more Discovery Proxies that serve the named link. In other words, the Discovery Proxies are the authoritative name servers for that subdomain. As in the rest of DNS-Based Service Discovery, all names are represented as-is using plain UTF-8 encoding, and, as described in Section 5.5.4, no text encoding translations are performed.

With appropriate VLAN configuration [IEEE-1Q] a single Discovery Proxy device could have a logical presence on many links, and serve as the Discovery Proxy for all those links. In such a configuration the Discovery Proxy device would have a single physical Ethernet [IEEE-3] port, configured as a VLAN trunk port, which would appear to software on that device as multiple virtual Ethernet interfaces, one connected to each of the VLAN links.

As an alternative to using VLAN technology, using a Multicast DNS Discovery Relay [Relay] is another way that a Discovery Proxy can have a 'virtual' presence on a remote link.

When a DNS-SD client issues a Unicast DNS query to discover services in a particular Unicast DNS subdomain (e.g., "\_printer.\_tcp.Building 1.example.com. PTR ?") the normal DNS delegation mechanism results in that query being forwarded until it reaches the delegated authoritative name server for that subdomain, namely the Discovery Proxy on the link in question. Like a conventional Unicast DNS server, a Discovery Proxy implements the usual Unicast DNS protocol [RFC1034] [RFC1035] over UDP and TCP. However, unlike a conventional Unicast DNS server that generates answers from the data in its manually-configured zone file, a Discovery Proxy generates answers using Multicast DNS. A Discovery Proxy does this by consulting its Multicast DNS cache and/or issuing Multicast DNS queries, as appropriate, according to the usual protocol rules of Multicast DNS [RFC6762], for the corresponding Multicast DNS name, type and class, with the delegated zone part of the name replaced with ".local" (e.g., in this case, "\_printer.\_tcp.local. PTR ?"). Then, from the received Multicast DNS data, the Discovery Proxy synthesizes the appropriate Unicast DNS response, with the ".local" top-level label replaced with the name of the delegated zone. How long the Discovery Proxy should wait to accumulate Multicast DNS responses before sending its unicast reply is described below in Section 5.6.

The existing Multicast DNS caching mechanism is used to minimize unnecessary Multicast DNS queries on the wire. The Discovery Proxy is acting as a client of the underlying Multicast DNS subsystem, and benefits from the same caching and efficiency measures as any other client using that subsystem.

Note that the contents of the delegated zone, generated as it is by performing ".local" Multicast DNS queries, mirrors the records available on the local link via Multicast DNS very closely, but not precisely. There is not a full bidirectional equivalence between the two. Certain records that are available via Multicast DNS may not have equivalents in the delegated zone, possibly because they are invalid or not relevant in the delegated zone, or because they are being suppressed because they are unusable outside the local link (see Section 5.5.2). Conversely, certain records that appear in the delegated zone may not have corresponding records available on the local link via Multicast DNS. In particular there are certain administrative SRV records (see Section 6) that logically fall within the delegated zone, but semantically represent metadata *about* the zone rather than records *within* the zone, and consequently these administrative records in the delegated zone do not have any corresponding counterparts in the Multicast DNS namespace of the local link.

## 5.2. Domain Enumeration

A DNS-SD client performs Domain Enumeration [RFC6763] via certain PTR queries, using both unicast and multicast. If it receives a Domain Name configuration via DHCP option 15 [RFC2132], then it issues unicast queries using this domain. It issues unicast queries using names derived from its IPv4 subnet address(es) and IPv6 prefix(es). These are described below in Section 5.2.1. It also issues multicast Domain Enumeration queries in the "local" domain [RFC6762]. These are described below in Section 5.2.2. The results of all the Domain Enumeration queries are combined for Service Discovery purposes.

### 5.2.1. Domain Enumeration via Unicast Queries

The administrator creates Domain Enumeration PTR records [RFC6763] to inform clients of available service discovery domains. Two varieties of such Domain Enumeration PTR records exist; those with names derived from the domain name communicated to the clients via DHCP, and those with names derived from IPv4 subnet address(es) and IPv6 prefix(es) in use by the clients. Below is an example showing the name-based variety:

b._dns-sd._udp.example.com.	PTR	Building 1.example.com.
	PTR	Building 2.example.com.
	PTR	Building 3.example.com.
	PTR	Building 4.example.com.
db._dns-sd._udp.example.com.	PTR	Building 1.example.com.
lb._dns-sd._udp.example.com.	PTR	Building 1.example.com.

The meaning of these records is defined in the DNS Service Discovery specification [RFC6763] but for convenience is repeated here. The "b" ("browse") records tell the client device the list of browsing domains to display for the user to select from. The "db" ("default browse") record tells the client device which domain in that list should be selected by default. The "db" domain MUST be one of the domains in the "b" list; if not then no domain is selected by default. The "lb" ("legacy browse") record tells the client device which domain to automatically browse on behalf of applications that don't implement UI for multi-domain browsing (which is most of them, at the time of writing). The "lb" domain is often the same as the "db" domain, or sometimes the "db" domain plus one or more others that should be included in the list of automatic browsing domains for legacy clients.

Note that in the example above, for clarity, space characters in names are shown as actual spaces. If this data is manually entered

into a textual zone file for authoritative server software such as BIND, care must be taken because the space character is used as a field separator, and other characters like dot ('.'), semicolon (';'), dollar ('\$'), backslash ('\'), etc., also have special meaning. These characters have to be escaped when entered into a textual zone file, following the rules in Section 5.1 of the DNS specification [RFC1035]. For example, a literal space in a name is represented in the textual zone file using '\032', so "Building 1.example.com." is entered as "Building\0321.example.com."

DNS responses are limited to a maximum size of 65535 bytes. This limits the maximum number of domains that can be returned for a Domain Enumeration query, as follows:

A DNS response header is 12 bytes. That's typically followed by a single qname (up to 256 bytes) plus qtype (2 bytes) and qclass (2 bytes), leaving 65275 for the Answer Section.

An Answer Section Resource Record consists of:

- o Owner name, encoded as a two-byte compression pointer
- o Two-byte rrtype (type PTR)
- o Two-byte rrclass (class IN)
- o Four-byte ttl
- o Two-byte rdlength
- o rdata (domain name, up to 256 bytes)

This means that each Resource Record in the Answer Section can take up to 268 bytes total, which means that the Answer Section can contain, in the worst case, no more than 243 domains.

In a more typical scenario, where the domain names are not all maximum-sized names, and there is some similarity between names so that reasonable name compression is possible, each Answer Section Resource Record may average 140 bytes, which means that the Answer Section can contain up to 466 domains.

It is anticipated that this should be sufficient for even a large corporate network or university campus.

### 5.2.2. Domain Enumeration via Multicast Queries

In the case where Discovery Proxy functionality is widely deployed within an enterprise (either by having a Discovery Proxy on each link, or by having a Discovery Proxy with a remote 'virtual' presence on each link using VLANs or Multicast DNS Discovery Relays [Relay]) this offers an additional way to provide Domain Enumeration data for clients.

A Discovery Proxy can be configured to generate Multicast DNS responses for the following Multicast DNS Domain Enumeration queries issued by clients:

b._dns-sd._udp.local.	PTR	?
db._dns-sd._udp.local.	PTR	?
lb._dns-sd._udp.local.	PTR	?

This provides the ability for Discovery Proxies to indicate recommended browsing domains to DNS-SD clients on a per-link granularity. In some enterprises it may be preferable to provide this per-link configuration data in the form of Discovery Proxy configuration, rather than populating the Unicast DNS servers with the same data (in the "ip6.arpa" or "in-addr.arpa" domains).

Regardless of how the network operator chooses to provide this configuration data, clients will perform Domain Enumeration via both unicast and multicast queries, and then combine the results of these queries.

### 5.3. Delegated Subdomain for LDH Host Names

DNS-SD service instance names and domains are allowed to contain arbitrary Net-Unicode text [RFC5198], encoded as precomposed UTF-8 [RFC3629].

Users typically interact with service discovery software by viewing a list of discovered service instance names on a display, and selecting one of them by pointing, touching, or clicking. Similarly, in software that provides a multi-domain DNS-SD user interface, users view a list of offered domains on the display and select one of them by pointing, touching, or clicking. To use a service, users don't have to remember domain or instance names, or type them; users just have to be able to recognize what they see on the display and touch or click on the thing they want.

In contrast, host names are often remembered and typed. Also, host names have historically been used in command-line interfaces where spaces can be inconvenient. For this reason, host names have traditionally been restricted to letters, digits and hyphens (LDH), with no spaces or other punctuation.

While we do want to allow rich text for DNS-SD service instance names and domains, it is advisable, for maximum compatibility with existing usage, to restrict host names to the traditional letter-digit-hyphen rules. This means that while a service name "My Printer.\_ipp.\_tcp.Building 1.example.com" is acceptable and desirable (it is displayed in a graphical user interface as an instance called "My Printer" in the domain "Building 1" at "example.com"), a host name "My-Printer.Building 1.example.com" is less desirable (because of the space in "Building 1").

To accomodate this difference in allowable characters, a Discovery Proxy SHOULD support having two separate subdomains delegated to it for each link it serves, one whose name is allowed to contain arbitrary Net-Unicode text [RFC5198], and a second more constrained subdomain whose name is restricted to contain only letters, digits, and hyphens, to be used for host name records (names of 'A' and 'AAAA' address records). The restricted names may be any valid name consisting of only letters, digits, and hyphens, including Punycode-encoded names [RFC3492].

For example, a Discovery Proxy could have the two subdomains "Building 1.example.com" and "bldg1.example.com" delegated to it. The Discovery Proxy would then translate these two Multicast DNS records:

```
My Printer._ipp._tcp.local. SRV 0 0 631 prnt.local.
prnt.local.                A    203.0.113.2
```

into Unicast DNS records as follows:

```
My Printer._ipp._tcp.Building 1.example.com.
                                SRV 0 0 631 prnt.bldg1.example.com.
prnt.bldg1.example.com.        A    203.0.113.2
```

Note that the SRV record name is translated using the rich-text domain name ("Building 1.example.com") and the address record name is translated using the LDH domain ("bldg1.example.com").

A Discovery Proxy MAY support only a single rich text Net-Unicode domain, and use that domain for all records, including 'A' and 'AAAA' address records, but implementers choosing this option should be aware that this choice may produce host names that are awkward to use in command-line environments. Whether this is an issue depends on whether users in the target environment are expected to be using command-line interfaces.

A Discovery Proxy MUST NOT be restricted to support only a letter-digit-hyphen subdomain, because that results in an unnecessarily poor user experience.

As described above in Section 5.2.1, for clarity, space characters in names are shown as actual spaces. If this data were to be manually entered into a textual zone file (which it isn't) then spaces would need to be represented using '\032', so "My Printer.\_ipp.\_tcp.Building 1.example.com." would become "My\032Printer.\_ipp.\_tcp.Building\0321.example.com." Note that the '\032' representation does not appear in the network packets sent over the air. In the wire format of DNS messages, spaces are sent as spaces, not as '\032', and likewise, in a graphical user interface at the client device, spaces are shown as spaces, not as '\032'.

#### 5.4. Delegated Subdomain for Reverse Mapping

A Discovery Proxy can facilitate easier management of reverse mapping domains, particularly for IPv6 addresses where manual management may be more onerous than it is for IPv4 addresses.

To achieve this, in the parent domain, NS records are used to delegate ownership of the appropriate reverse mapping domain to the Discovery Proxy. In other words, the Discovery Proxy becomes the authoritative name server for the reverse mapping domain. For fault tolerance reasons there may be more than one Discovery Proxy serving a given link.

If a given link is using the IPv4 subnet 203.0.113/24, then the domain "113.0.203.in-addr.arpa" is delegated to the Discovery Proxy for that link.

For example, if a given link is using the IPv6 prefix 2001:0DB8:1234:5678/64, then the domain "8.7.6.5.4.3.2.1.8.b.d.0.1.0.0.2.ip6.arpa" is delegated to the Discovery Proxy for that link.

When a reverse mapping query arrives at the Discovery Proxy, it issues the identical query on its local link as a Multicast DNS query. The mechanism to force an apparently unicast name to be resolved using link-local Multicast DNS varies depending on the API set being used. For example, in the "dns\_sd.h" APIs (available on macOS, iOS, Bonjour for Windows, Linux and Android), using `kDNSServiceFlagsForceMulticast` indicates that the `DNSServiceQueryRecord()` call should perform the query using Multicast DNS. Other APIs sets have different ways of forcing multicast queries. When the host owning that IPv4 or IPv6 address responds with a name of the form "something.local", the Discovery Proxy rewrites that to use its configured LDH host name domain instead of "local", and returns the response to the caller.



For example, a Discovery Proxy with the two subdomains "113.0.203.in-addr.arpa" and "bldg1.example.com" delegated to it would translate this Multicast DNS record:

2.113.0.203.in-addr.arpa. PTR prnt.local.

into this Unicast DNS response:

2.113.0.203.in-addr.arpa. PTR prnt.bldg1.example.com.

Subsequent queries for the prnt.bldg1.example.com address record, falling as it does within the bldg1.example.com domain, which is delegated to the Discovery Proxy, will arrive at the Discovery Proxy, where they are answered by issuing Multicast DNS queries and using the received Multicast DNS answers to synthesize Unicast DNS responses, as described above.

Note that this design assumes that all addresses on a given IPv4 subnet or IPv6 prefix are mapped to hostnames using the Discovery Proxy mechanism. It would be possible to implement a Discovery Proxy that can be configured so that some address-to-name mappings are performed using Multicast DNS on the local link, while other address-to-name mappings within the same IPv4 subnet or IPv6 prefix are configured manually.

### 5.5. Data Translation

Generating the appropriate Multicast DNS queries involves, at the very least, translating from the configured DNS domain (e.g., "Building 1.example.com") on the Unicast DNS side to "local" on the Multicast DNS side.

Generating the appropriate Unicast DNS responses involves translating back from "local" to the appropriate configured DNS Unicast domain.

Other beneficial translation and filtering operations are described below.

#### 5.5.1. DNS TTL limiting

For efficiency, Multicast DNS typically uses moderately high DNS TTL values. For example, the typical TTL on DNS-SD PTR records is 75 minutes. What makes these moderately high TTLs acceptable is the cache coherency mechanisms built in to the Multicast DNS protocol which protect against stale data persisting for too long. When a service shuts down gracefully, it sends goodbye packets to remove its PTR records immediately from neighboring caches. If a service shuts down abruptly without sending goodbye packets, the Passive Observation Of Failures (POOF) mechanism described in Section 10.5 of the Multicast DNS specification [RFC6762] comes into play to purge the cache of stale data.

A traditional Unicast DNS client on a distant remote link does not get to participate in these Multicast DNS cache coherency mechanisms on the local link. For traditional Unicast DNS queries (those received without using Long-Lived Query [LLQ] or DNS Push Notification subscriptions [Push]) the DNS TTLs reported in the resulting Unicast DNS response MUST be capped to be no more than ten seconds.

Similarly, for negative responses, the negative caching TTL indicated in the SOA record [RFC2308] should also be ten seconds (Section 6.1).

This value of ten seconds is chosen based on user-experience considerations.

For negative caching, suppose a user is attempting to access a remote device (e.g., a printer), and they are unsuccessful because that device is powered off. Suppose they then place a telephone call and ask for the device to be powered on. We want the device to become available to the user within a reasonable time period. It is reasonable to expect it to take on the order of ten seconds for a simple device with a simple embedded operating system to power on.

Once the device is powered on and has announced its presence on the network via Multicast DNS, we would like it to take no more than a further ten seconds for stale negative cache entries to expire from Unicast DNS caches, making the device available to the user desiring to access it.

Similar reasoning applies to capping positive TTLs at ten seconds. In the event of a device moving location, getting a new DHCP address, or other renumbering events, we would like the updated information to be available to remote clients in a relatively timely fashion.

However, network administrators should be aware that many recursive (caching) DNS servers by default are configured to impose a minimum TTL of 30 seconds. If stale data appears to be persisting in the network to the extent that it adversely impacts user experience, network administrators are advised to check the configuration of their recursive DNS servers.

For received Unicast DNS queries that use LLQ [LLQ] or DNS Push Notifications [Push], the Multicast DNS record's TTL SHOULD be returned unmodified, because the Push Notification channel exists to inform the remote client as records come and go. For further details about Long-Lived Queries, and its newer replacement, DNS Push Notifications, see Section 5.6.

#### 5.5.2. Suppressing Unusable Records

A Discovery Proxy SHOULD offer a configurable option, enabled by default, to suppress Unicast DNS answers for records that are not useful outside the local link. When the option to suppress unusable records is enabled:

- o DNS A and AAAA records for IPv4 link-local addresses [RFC3927] and IPv6 link-local addresses [RFC4862] SHOULD be suppressed.
- o Similarly, for sites that have multiple private address realms [RFC1918], in cases where the Discovery Proxy can determine that the querying client is in a different address realm, private addresses SHOULD NOT be communicated to that client.
- o IPv6 Unique Local Addresses [RFC4193] SHOULD be suppressed in cases where the Discovery Proxy can determine that the querying client is in a different IPv6 address realm.
- o By the same logic, DNS SRV records that reference target host names that have no addresses usable by the requester should be suppressed, and likewise, DNS PTR records that point to unusable SRV records should be similarly be suppressed.

### 5.5.3. NSEC and NSEC3 queries

Multicast DNS devices do not routinely announce their records on the network. Generally they remain silent until queried. This means that the complete set of Multicast DNS records in use on a link can only be discovered by active querying, not by passive listening. Because of this, a Discovery Proxy can only know what names exist on a link by issuing queries for them, and since it would be impractical to issue queries for every possible name just to find out which names exist and which do not, a Discovery Proxy cannot programmatically generate the traditional NSEC [RFC4034] and NSEC3 [RFC5155] records which assert the nonexistence of a large range of names.

When queried for an NSEC or NSEC3 record type, the Discovery Proxy issues a qtype "ANY" query using Multicast DNS on the local link, and then generates an NSEC or NSEC3 response with a Type Bit Map signifying which record types do and do not exist for just the specific name queried, and no other names.

Multicast DNS NSEC records received on the local link MUST NOT be forwarded unmodified to a unicast querier, because there are slight differences in the NSEC record data. In particular, Multicast DNS NSEC records do not have the NSEC bit set in the Type Bit Map, whereas conventional Unicast DNS NSEC records do have the NSEC bit set.

### 5.5.4. No Text Encoding Translation

A Discovery Proxy does no translation between text encodings. Specifically, a Discovery Proxy does no translation between Punycode encoding [RFC3492] and UTF-8 encoding [RFC3629], either in the owner name of DNS records, or anywhere in the RDATA of DNS records (such as the RDATA of PTR records, SRV records, NS records, or other record types like TXT, where it is ambiguous whether the RDATA may contain DNS names). All bytes are treated as-is, with no attempt at text encoding translation. A client implementing DNS-based Service Discovery [RFC6763] will use UTF-8 encoding for its service discovery queries, which the Discovery Proxy passes through without any text encoding translation to the Multicast DNS subsystem. Responses from the Multicast DNS subsystem are similarly returned, without any text encoding translation, back to the requesting client.

#### 5.5.5. Application-Specific Data Translation

There may be cases where Application-Specific Data Translation is appropriate.

For example, AirPrint printers tend to advertise fairly verbose information about their capabilities in their DNS-SD TXT record. TXT record sizes in the range 500-1000 bytes are not uncommon. This information is a legacy from LPR printing, because LPR does not have in-band capability negotiation, so all of this information is conveyed using the DNS-SD TXT record instead. IPP printing does have in-band capability negotiation, but for convenience printers tend to include the same capability information in their IPP DNS-SD TXT records as well. For local mDNS use this extra TXT record information is inefficient, but not fatal. However, when a Discovery Proxy aggregates data from multiple printers on a link, and sends it via unicast (via UDP or TCP) this amount of unnecessary TXT record information can result in large responses. A DNS reply over TCP carrying information about 70 printers with an average of 700 bytes per printer adds up to about 50 kilobytes of data. Therefore, a Discovery Proxy that is aware of the specifics of an application-layer protocol such as AirPrint (which uses IPP) can elide unnecessary key/value pairs from the DNS-SD TXT record for better network efficiency.

Also, the DNS-SD TXT record for many printers contains an "adminurl" key something like "adminurl=http://printername.local/status.html". For this URL to be useful outside the local link, the embedded ".local" hostname needs to be translated to an appropriate name with larger scope. It is easy to translate ".local" names when they appear in well-defined places, either as a record's name, or in the rdata of record types like PTR and SRV. In the printing case, some application-specific knowledge about the semantics of the "adminurl" key is needed for the Discovery Proxy to know that it contains a name that needs to be translated. This is somewhat analogous to the need for NAT gateways to contain ALGs (Application-Specific Gateways) to facilitate the correct translation of protocols that embed addresses in unexpected places.

To avoid the need for application-specific knowledge about the semantics of particular TXT record keys, protocol designers are advised to avoid placing link-local names or link-local IP addresses in TXT record keys, if translation of those names or addresses would be required for off-link operation. In the printing case, the operational failure of failing to translate the "adminurl" key correctly is that, when accessed from a different link, printing will still work, but clicking the "Admin" UI button will fail to open the printer's administration page. Rather than duplicating the host name

from the service's SRV record in its "adminurl" key, thereby having the same host name appear in two places, a better design might have been to omit the host name from the "adminurl" key, and instead have the client implicitly substitute the target host name from the service's SRV record in place of a missing host name in the "adminurl" key. That way the desired host name only appears once, and it is in a well-defined place where software like the Discovery Proxy is expecting to find it.

Note that this kind of Application-Specific Data Translation is expected to be very rare. It is the exception, rather than the rule. This is an example of a common theme in computing. It is frequently the case that it is wise to start with a clean, layered design, with clear boundaries. Then, in certain special cases, those layer boundaries may be violated, where the performance and efficiency benefits outweigh the inelegance of the layer violation.

These layer violations are optional. They are done primarily for efficiency reasons, and generally should not be required for correct operation. A Discovery Proxy MAY operate solely at the mDNS layer, without any knowledge of semantics at the DNS-SD layer or above.

## 5.6. Answer Aggregation

In a simple analysis, simply gathering multicast answers and forwarding them in a unicast response seems adequate, but it raises the question of how long the Discovery Proxy should wait to be sure that it has received all the Multicast DNS answers it needs to form a complete Unicast DNS response. If it waits too little time, then it risks its Unicast DNS response being incomplete. If it waits too long, then it creates a poor user experience at the client end. In fact, there may be no time which is both short enough to produce a good user experience and at the same time long enough to reliably produce complete results.

Similarly, the Discovery Proxy -- the authoritative name server for the subdomain in question -- needs to decide what DNS TTL to report for these records. If the TTL is too long then the recursive (caching) name servers issuing queries on behalf of their clients risk caching stale data for too long. If the TTL is too short then the amount of network traffic will be more than necessary. In fact, there may be no TTL which is both short enough to avoid undesirable stale data and at the same time long enough to be efficient on the network.

Both these dilemmas are solved by use of DNS Long-Lived Queries (DNS LLQ) [LLQ] or its newer replacement, DNS Push Notifications [Push].

Clients supporting unicast DNS Service Discovery SHOULD implement DNS Push Notifications [Push] for improved user experience.

Clients and Discovery Proxies MAY support both DNS LLQ and DNS Push, and when talking to a Discovery Proxy that supports both, the client may use either protocol, as it chooses, though it is expected that only DNS Push will continue to be supported in the long run.

When a Discovery Proxy receives a query using DNS LLQ or DNS Push Notifications, it responds immediately using the Multicast DNS records it already has in its cache (if any). This provides a good client user experience by providing a near-instantaneous response. Simultaneously, the Discovery Proxy issues a Multicast DNS query on the local link to discover if there are any additional Multicast DNS records it did not already know about. Should additional Multicast DNS responses be received, these are then delivered to the client using additional DNS LLQ or DNS Push Notification update messages. The timeliness of such update messages is limited only by the timeliness of the device responding to the Multicast DNS query. If the Multicast DNS device responds quickly, then the update message is delivered quickly. If the Multicast DNS device responds slowly, then

the update message is delivered slowly. The benefit of using update messages is that the Discovery Proxy can respond promptly because it doesn't have to delay its unicast response to allow for the expected worst-case delay for receiving all the Multicast DNS responses. Even if a proxy were to try to provide reliability by assuming an excessively pessimistic worst-case time (thereby giving a very poor user experience) there would still be the risk of a slow Multicast DNS device taking even longer than that (e.g., a device that is not even powered on until ten seconds after the initial query is received) resulting in incomplete responses. Using update message solves this dilemma: even very late responses are not lost; they are delivered in subsequent update messages.

There are two factors that determine specifically how responses are generated:

The first factor is whether the query from the client used LLQ or DNS Push Notifications (used for long-lived service browsing PTR queries) or not (used for one-shot operations like SRV or address record queries). Note that queries using LLQ or DNS Push Notifications are received directly from the client. Queries not using LLQ or DNS Push Notifications are generally received via the client's configured recursive (caching) name server.

The second factor is whether the Discovery Proxy already has at least one record in its cache that positively answers the question.

- o Not using LLQ or Push Notifications; no answer in cache:  
Issue an mDNS query, exactly as a local client would issue an mDNS query on the local link for the desired record name, type and class, including retransmissions, as appropriate, according to the established mDNS retransmission schedule [RFC6762]. As soon as any Multicast DNS response packet is received that contains one or more positive answers to that question (with or without the Cache Flush bit [RFC6762] set), or a negative answer (signified via a Multicast DNS NSEC record [RFC6762]), the Discovery Proxy generates a Unicast DNS response packet containing the corresponding (filtered and translated) answers and sends it to the remote client. If after six seconds no Multicast DNS answers have been received, cancel the mDNS query and return a negative response to the remote client. Six seconds is enough time to transmit three mDNS queries, and allow some time for responses to arrive.  
DNS TTLs in responses MUST be capped to at most ten seconds.  
(Reasoning: Queries not using LLQ or Push Notifications are generally queries that expect an answer from only one device, so the first response is also the only response.)



- o Not using LLQ or Push Notifications; at least one answer in cache:  
Send response right away to minimise delay.  
DNS TTLs in responses MUST be capped to at most ten seconds.  
No local mDNS queries are performed.  
(Reasoning: Queries not using LLQ or Push Notifications are generally queries that expect an answer from only one device. Given RRSets TTL harmonisation, if the proxy has one Multicast DNS answer in its cache, it can reasonably assume that it has all of them.)
- o Using LLQ or Push Notifications; no answer in cache:  
As in the case above with no answer in the cache, perform mDNS querying for six seconds, and send a response to the remote client as soon as any relevant mDNS response is received.  
If after six seconds no relevant mDNS response has been received, return negative response to the remote client (for LLQ; not applicable for Push Notifications).  
(Reasoning: We don't need to rush to send an empty answer.)  
Whether or not a relevant mDNS response is received within six seconds, the query remains active for as long as the client maintains the LLQ or Push Notification state, and if mDNS answers are received later, LLQ or Push Notification messages are sent.  
DNS TTLs in responses are returned unmodified.
- o Using LLQ or Push Notifications; at least one answer in cache:  
As in the case above with at least one answer in cache, send response right away to minimise delay.  
The query remains active for as long as the client maintains the LLQ or Push Notification state, and results in transmission of mDNS queries, with appropriate Known Answer lists, to determine if further answers are available. If additional mDNS answers are received later, LLQ or Push Notification messages are sent.  
(Reasoning: We want UI that is displayed very rapidly, yet continues to remain accurate even as the network environment changes.)  
DNS TTLs in responses are returned unmodified.

The "negative responses" referred to above are "no error no answer" negative responses, not NXDOMAIN. This is because the Discovery Proxy cannot know all the Multicast DNS domain names that may exist on a link at any given time, so any name with no answers may have child names that do exist, making it an "empty nonterminal" name.

Note that certain aspects of the behavior described here do not have to be implemented overtly by the Discovery Proxy; they occur naturally as a result of using existing Multicast DNS APIs.

For example, in the first case above (no LLQ or Push Notifications, and no answers in the cache) if a new Multicast DNS query is requested (either by a local client, or by the Discovery Proxy on behalf of a remote client), and there is not already an identical Multicast DNS query active, and there are no matching answers already in the Multicast DNS cache on the Discovery Proxy device, then this will cause a series of Multicast DNS query packets to be issued with exponential backoff. The exponential backoff sequence in some implementations starts at one second and then doubles for each retransmission (0, 1, 3, 7 seconds, etc.) and in others starts at one second and then triples for each retransmission (0, 1, 4, 13 seconds, etc.). In either case, if no response has been received after six seconds, that is long enough that the underlying Multicast DNS implementation will have sent three query packets without receiving any response. At that point the Discovery Proxy cancels its Multicast DNS query (so no further Multicast DNS query packets will be sent for this query) and returns a negative response to the remote client via unicast.

The six-second delay is chosen to be long enough to give enough time for devices to respond, yet short enough not to be too onerous for a human user waiting for a response. For example, using the "dig" DNS debugging tool, the current default settings result in it waiting a total of 15 seconds for a reply (three transmissions of the query packet, with a wait of 5 seconds after each packet) which is ample time for it to have received a negative reply from a Discovery Proxy after six seconds.

The statement that for a one-shot query (i.e., no LLQ or Push Notifications requested), if at least one answer is already available in the cache then a Discovery Proxy should not issue additional mDNS query packets, also occurs naturally as a result of using existing Multicast DNS APIs. If a new Multicast DNS query is requested (either locally, or by the Discovery Proxy on behalf of a remote client), for which there are relevant answers already in the Multicast DNS cache on the Discovery Proxy device, and after the answers are delivered the Multicast DNS query is then cancelled immediately, then no Multicast DNS query packets will be generated for this query.

## 6. Administrative DNS Records

### 6.1. DNS SOA (Start of Authority) Record

The MNAME field SHOULD contain the host name of the Discovery Proxy device (i.e., the same domain name as the rdata of the NS record delegating the relevant zone(s) to this Discovery Proxy device).

The RNAME field SHOULD contain the mailbox of the person responsible for administering this Discovery Proxy device.

The SERIAL field MUST be zero.

Zone transfers are undefined for Discovery Proxy zones, and consequently the REFRESH, RETRY and EXPIRE fields have no useful meaning for Discovery Proxy zones. These fields SHOULD contain reasonable default values. The RECOMMENDED values are: REFRESH 7200, RETRY 3600, EXPIRE 86400.

The MINIMUM field (used to control the lifetime of negative cache entries) SHOULD contain the value 10. The value of ten seconds is chosen based on user-experience considerations (see Section 5.5.1).

In the event that there are multiple Discovery Proxy devices on a link for fault tolerance reasons, this will result in clients receiving inconsistent SOA records (different MNAME, and possibly RNAME) depending on which Discovery Proxy answers their SOA query. However, since clients generally have no reason to use the MNAME or RNAME data, this is unlikely to cause any problems.

## 6.2. DNS NS Records

In the event that there are multiple Discovery Proxy devices on a link for fault tolerance reasons, the parent zone MUST be configured with NS records giving the names of all the Discovery Proxy devices on the link.

Each Discovery Proxy device MUST be configured to answer NS queries for the zone apex name by giving its own NS record, and the NS records of its fellow Discovery Proxy devices on the same link, so that it can return the correct answers for NS queries.

The target host name in the RDATA of an NS record MUST NOT reference a name that falls within any zone delegated to a Discovery Proxy. Apart from the zone apex name, all other host names that fall within a zone delegated to a Discovery Proxy correspond to local Multicast DNS host names, which logically belong to the respective Multicast DNS hosts defending those names, not the Discovery Proxy. Generally speaking, the Discovery Proxy does not own or control the delegated zone; it is merely a conduit to the corresponding ".local" namespace, which is controlled by the Multicast DNS hosts on that link. If an NS record were to reference a manually-determined host name that falls within a delegated zone, that manually-determined host name may inadvertently conflict with a corresponding ".local" host name that is owned and controlled by some device on that link.

## 6.3. DNS Delegation Records

Since the Multicast DNS specification [RFC6762] states that there can be no delegation (subdomains) within a ".local" namespace, this implies that any name within a zone delegated to a Discovery Proxy (except for the zone apex name itself) cannot have any answers for any DNS queries for RRTYPEs SOA, NS, or DS. Consequently:

- o for any query for the zone apex name of a zone delegated to a Discovery Proxy, the Discovery Proxy MUST generate the appropriate immediate answers as described above, and
- o for any query for RRTYPEs SOA, NS, or DS, for any name within a zone delegated to a Discovery Proxy, other than the zone apex name, instead of translating the query to its corresponding Multicast DNS ".local" equivalent, a Discovery Proxy MUST generate an immediate negative answer.

#### 6.4. DNS SRV Records

There are certain special DNS records that logically fall within the delegated unicast DNS subdomain, but rather than mapping to their corresponding ".local" namesakes, they actually contain metadata pertaining to the operation of the delegated unicast DNS subdomain itself. They do not exist in the corresponding ".local" namespace of the local link. For these queries a Discovery Proxy MUST generate immediate answers, whether positive or negative, to avoid delays while clients wait for their query to be answered. For example, if a Discovery Proxy does not implement Long-Lived Queries [LLQ] then it MUST return an immediate negative answer to tell the client this without delay, instead of passing the query through to the local network as a query for "\_dns-llq.\_udp.local.", and then waiting unsuccessfully for answers that will not be forthcoming.

If a Discovery Proxy implements Long-Lived Queries [LLQ] then it MUST positively respond to "\_dns-llq.\_udp.<zone> SRV" queries, "\_dns-llq.\_tcp.<zone> SRV" queries, and "\_dns-llq-tls.\_tcp.<zone> SRV" queries as appropriate, else it MUST return an immediate negative answer for those queries.

If a Discovery Proxy implements DNS Push Notifications [Push] then it MUST positively respond to "\_dns-push-tls.\_tcp.<zone>" queries, else it MUST return an immediate negative answer for those queries.

A Discovery Proxy MUST return an immediate negative answer for "\_dns-update.\_udp.<zone> SRV" queries, "\_dns-update.\_tcp.<zone> SRV" queries, and "\_dns-update-tls.\_tcp.<zone> SRV" queries, since using DNS Update [RFC2136] to change zones generated dynamically from local Multicast DNS data is not possible.

## 7. DNSSEC Considerations

### 7.1. On-line signing only

The Discovery Proxy acts as the authoritative name server for designated subdomains, and if DNSSEC is to be used, the Discovery Proxy needs to possess a copy of the signing keys, in order to generate authoritative signed data from the local Multicast DNS responses it receives. Off-line signing is not applicable to Discovery Proxy.

### 7.2. NSEC and NSEC3 Records

In DNSSEC NSEC [RFC4034] and NSEC3 [RFC5155] records are used to assert the nonexistence of certain names, also described as "authenticated denial of existence".

Since a Discovery Proxy only knows what names exist on the local link by issuing queries for them, and since it would be impractical to issue queries for every possible name just to find out which names exist and which do not, a Discovery Proxy cannot programmatically synthesize the traditional NSEC and NSEC3 records which assert the nonexistence of a large range of names. Instead, when generating a negative response, a Discovery Proxy programmatically synthesizes a single NSEC record assert the nonexistence of just the specific name queried, and no others. Since the Discovery Proxy has the zone signing key, it can do this on demand. Since the NSEC record asserts the nonexistence of only a single name, zone walking is not a concern, so NSEC3 is not necessary.

Note that this applies only to traditional immediate DNS queries, which may return immediate negative answers when no immediate positive answer is available. When used with a DNS Push Notification subscription [Push] there are no negative answers, merely the absence of answers so far, which may change in the future if answers become available.

## 8. IPv6 Considerations

An IPv4-only host and an IPv6-only host behave as "ships that pass in the night". Even if they are on the same Ethernet [IEEE-3], neither is aware of the other's traffic. For this reason, each link may have *\*two\** unrelated ".local." zones, one for IPv4 and one for IPv6. Since for practical purposes, a group of IPv4-only hosts and a group of IPv6-only hosts on the same Ethernet act as if they were on two entirely separate Ethernet segments, it is unsurprising that their use of the ".local." zone should occur exactly as it would if they really were on two entirely separate Ethernet segments.

It will be desirable to have a mechanism to 'stitch' together these two unrelated ".local." zones so that they appear as one. Such mechanism will need to be able to differentiate between a dual-stack (v4/v6) host participating in both ".local." zones, and two different hosts, one IPv4-only and the other IPv6-only, which are both trying to use the same name(s). Such a mechanism will be specified in a future companion document.

At present, it is RECOMMENDED that a Discovery Proxy be configured with a single domain name for both the IPv4 and IPv6 ".local." zones on the local link, and when a unicast query is received, it should issue Multicast DNS queries using both IPv4 and IPv6 on the local link, and then combine the results.

## 9. Security Considerations

### 9.1. Authenticity

A service proves its presence on a link by its ability to answer link-local multicast queries on that link. If greater security is desired, then the Discovery Proxy mechanism should not be used, and something with stronger security should be used instead, such as authenticated secure DNS Update [RFC2136] [RFC3007].

### 9.2. Privacy

The Domain Name System is, generally speaking, a global public database. Records that exist in the Domain Name System name hierarchy can be queried by name from, in principle, anywhere in the world. If services on a mobile device (like a laptop computer) are made visible via the Discovery Proxy mechanism, then when those services become visible in a domain such as "My House.example.com" that might indicate to (potentially hostile) observers that the mobile device is in my house. When those services disappear from "My House.example.com" that change could be used by observers to infer when the mobile device (and possibly its owner) may have left the house. The privacy of this information may be protected using techniques like firewalls, split-view DNS, and Virtual Private Networks (VPNs), as are customarily used today to protect the privacy of corporate DNS information.

The privacy issue is particularly serious for the IPv4 and IPv6 reverse zones. If the public delegation of the reverse zones points to the Discovery Proxy, and the Discovery Proxy is reachable globally, then it could leak a significant amount of information. Attackers could discover hosts that otherwise might not be easy to identify, and learn their hostnames. Attackers could also discover the existence of links where hosts frequently come and go.

The Discovery Proxy could also provide sensitive records only to authenticated users. This is a general DNS problem, not specific to the Discovery Proxy. Work is underway in the IETF to tackle this problem [RFC7626].

### 9.3. Denial of Service

A remote attacker could use a rapid series of unique Unicast DNS queries to induce a Discovery Proxy to generate a rapid series of corresponding Multicast DNS queries on one or more of its local links. Multicast traffic is generally more expensive than unicast traffic -- especially on Wi-Fi links -- which makes this attack particularly serious. To limit the damage that can be caused by such



attacks, a Discovery Proxy (or the underlying Multicast DNS subsystem which it utilizes) MUST implement Multicast DNS query rate limiting appropriate to the link technology in question. For today's 802.11b/g/n/ac Wi-Fi links (for which approximately 200 multicast packets per second is sufficient to consume approximately 100% of the wireless spectrum) a limit of 20 Multicast DNS query packets per second is RECOMMENDED. On other link technologies like Gigabit Ethernet higher limits may be appropriate. A consequence of this rate limiting is that a rogue remote client could issue an excessive number of queries, resulting in denial of service to other legitimate remote clients attempting to use that Discovery Proxy. However, this is preferable to a rogue remote client being able to inflict even greater harm on the local network, which could impact the correct operation of all local clients on that network.

#### 10. IANA Considerations

This document has no IANA Considerations.

#### 11. Acknowledgments

Thanks to Markus Stenberg for helping develop the policy regarding the four styles of unicast response according to what data is immediately available in the cache. Thanks to Anders Brandt, Ben Campbell, Tim Chown, Alissa Cooper, Spencer Dawkins, Ralph Droms, Joel Halpern, Ray Hunter, Joel Jaeggli, Warren Kumari, Ted Lemon, Alexey Melnikov, Kathleen Moriarty, Tom Pusateri, Eric Rescorla, Adam Roach, David Schinazi, Markus Stenberg, Dave Thaler, and Andrew Yourtchenko for their comments.

## 12. References

### 12.1. Normative References

- [RFC1034] Mockapetris, P., "Domain names - concepts and facilities", STD 13, RFC 1034, DOI 10.17487/RFC1034, November 1987, <<https://www.rfc-editor.org/info/rfc1034>>.
- [RFC1035] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, DOI 10.17487/RFC1035, November 1987, <<https://www.rfc-editor.org/info/rfc1035>>.
- [RFC1918] Rekhter, Y., Moskowitz, B., Karrenberg, D., de Groot, G., and E. Lear, "Address Allocation for Private Internets", BCP 5, RFC 1918, DOI 10.17487/RFC1918, February 1996, <<https://www.rfc-editor.org/info/rfc1918>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [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>>.
- [RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, DOI 10.17487/RFC3629, November 2003, <<https://www.rfc-editor.org/info/rfc3629>>.
- [RFC3927] Cheshire, S., Aboba, B., and E. Guttman, "Dynamic Configuration of IPv4 Link-Local Addresses", RFC 3927, DOI 10.17487/RFC3927, May 2005, <<https://www.rfc-editor.org/info/rfc3927>>.
- [RFC4034] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "Resource Records for the DNS Security Extensions", RFC 4034, DOI 10.17487/RFC4034, March 2005, <<https://www.rfc-editor.org/info/rfc4034>>.
- [RFC4862] Thomson, S., Narten, T., and T. Jinmei, "IPv6 Stateless Address Autoconfiguration", RFC 4862, DOI 10.17487/RFC4862, September 2007, <<https://www.rfc-editor.org/info/rfc4862>>.

- [RFC5155] Laurie, B., Sisson, G., Arends, R., and D. Blacka, "DNS Security (DNSSEC) Hashed Authenticated Denial of Existence", RFC 5155, DOI 10.17487/RFC5155, March 2008, <<https://www.rfc-editor.org/info/rfc5155>>.
- [RFC5198] Klensin, J. and M. Padlipsky, "Unicode Format for Network Interchange", RFC 5198, DOI 10.17487/RFC5198, March 2008, <<https://www.rfc-editor.org/info/rfc5198>>.
- [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>>.
- [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>>.
- [RFC8490] Bellis, R., Cheshire, S., Dickinson, J., Dickinson, S., Lemon, T., and T. Pusateri, "DNS Stateful Operations", RFC 8490, DOI 10.17487/RFC8490, March 2019, <<https://www.rfc-editor.org/info/rfc8490>>.
- [Push] Pusateri, T. and S. Cheshire, "DNS Push Notifications", draft-ietf-dnssd-push-19 (work in progress), March 2019.

## 12.2. Informative References

- [Roadmap] Cheshire, S., "Service Discovery Road Map", draft-cheshire-dnssd-roadmap-03 (work in progress), October 2018.
- [DNS-UL] Sekar, K., "Dynamic DNS Update Leases", draft-sekar-dns-ul-01 (work in progress), August 2006.
- [LLQ] Cheshire, S. and M. Krochmal, "DNS Long-Lived Queries", draft-sekar-dns-llq-03 (work in progress), March 2019.
- [RegProt] Cheshire, S. and T. Lemon, "Service Registration Protocol for DNS-Based Service Discovery", draft-sctl-service-registration-00 (work in progress), July 2017.
- [Relay] Cheshire, S. and T. Lemon, "Multicast DNS Discovery Relay", draft-sctl-dnssd-mdns-relay-04 (work in progress), March 2018.

- [Mcast] Perkins, C., McBride, M., Stanley, D., Kumari, W., and J. Zuniga, "Multicast Considerations over IEEE 802 Wireless Media", draft-ietf-mboned-ieee802-mcast-problems-04 (work in progress), November 2018.
- [RFC2132] Alexander, S. and R. Droms, "DHCP Options and BOOTP Vendor Extensions", RFC 2132, DOI 10.17487/RFC2132, March 1997, <<https://www.rfc-editor.org/info/rfc2132>>.
- [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>>.
- [RFC3007] Wellington, B., "Secure Domain Name System (DNS) Dynamic Update", RFC 3007, DOI 10.17487/RFC3007, November 2000, <<https://www.rfc-editor.org/info/rfc3007>>.
- [RFC3492] Costello, A., "Punycode: A Bootstring encoding of Unicode for Internationalized Domain Names in Applications (IDNA)", RFC 3492, DOI 10.17487/RFC3492, March 2003, <<https://www.rfc-editor.org/info/rfc3492>>.
- [RFC4193] Hinden, R. and B. Haberman, "Unique Local IPv6 Unicast Addresses", RFC 4193, DOI 10.17487/RFC4193, October 2005, <<https://www.rfc-editor.org/info/rfc4193>>.
- [RFC6760] Cheshire, S. and M. Krochmal, "Requirements for a Protocol to Replace the AppleTalk Name Binding Protocol (NBP)", RFC 6760, DOI 10.17487/RFC6760, February 2013, <<https://www.rfc-editor.org/info/rfc6760>>.
- [RFC7558] Lynn, K., Cheshire, S., Blanchet, M., and D. Migault, "Requirements for Scalable DNS-Based Service Discovery (DNS-SD) / Multicast DNS (mDNS) Extensions", RFC 7558, DOI 10.17487/RFC7558, July 2015, <<https://www.rfc-editor.org/info/rfc7558>>.
- [RFC7626] Bortzmeyer, S., "DNS Privacy Considerations", RFC 7626, DOI 10.17487/RFC7626, August 2015, <<https://www.rfc-editor.org/info/rfc7626>>.
- [RFC7788] Stenberg, M., Barth, S., and P. Pfister, "Home Networking Control Protocol", RFC 7788, DOI 10.17487/RFC7788, April 2016, <<https://www.rfc-editor.org/info/rfc7788>>.

- [RFC8375] Pfister, P. and T. Lemon, "Special-Use Domain 'home.arpa.'", RFC 8375, DOI 10.17487/RFC8375, May 2018, <<https://www.rfc-editor.org/info/rfc8375>>.
- [ohp] "Discovery Proxy (Hybrid Proxy) implementation for OpenWrt", <<https://github.com/sbyx/ohybridproxy/>>.
- [ZC] Cheshire, S. and D. Steinberg, "Zero Configuration Networking: The Definitive Guide", O'Reilly Media, Inc. , ISBN 0-596-10100-7, December 2005.
- [IEEE-1Q] "IEEE Standard for Local and metropolitan area networks -- Bridges and Bridged Networks", IEEE Std 802.1Q-2014, November 2014, <<http://standards.ieee.org/getieee802/download/802-1Q-2014.pdf>>.
- [IEEE-3] "Information technology - Telecommunications and information exchange between systems - Local and metropolitan area networks - Specific requirements - Part 3: Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications", IEEE Std 802.3-2008, December 2008, <<http://standards.ieee.org/getieee802/802.3.html>>.
- [IEEE-5] Institute of Electrical and Electronics Engineers, "Information technology - Telecommunications and information exchange between systems - Local and metropolitan area networks - Specific requirements - Part 5: Token ring access method and physical layer specification", IEEE Std 802.5-1998, 1995.
- [IEEE-11] "Information technology - Telecommunications and information exchange between systems - Local and metropolitan area networks - Specific requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications", IEEE Std 802.11-2007, June 2007, <<http://standards.ieee.org/getieee802/802.11.html>>.

## Appendix A. Implementation Status

Some aspects of the mechanism specified in this document already exist in deployed software. Some aspects are new. This section outlines which aspects already exist and which are new.

### A.1. Already Implemented and Deployed

Domain enumeration by the client (the "b.\_dns-sd.\_udp" queries) is already implemented and deployed.

Unicast queries to the indicated discovery domain is already implemented and deployed.

These are implemented and deployed in Mac OS X 10.4 and later (including all versions of Apple iOS, on all iPhone and iPads), in Bonjour for Windows, and in Android 4.1 "Jelly Bean" (API Level 16) and later.

Domain enumeration and unicast querying have been used for several years at IETF meetings to make Terminal Room printers discoverable from outside the Terminal room. When an IETF attendee presses Cmd-P on a Mac, or selects AirPrint on an iPad or iPhone, and the Terminal room printers appear, that is because the client is sending unicast DNS queries to the IETF DNS servers. A walk-through giving the details of this particular specific example is given in Appendix A of the Roadmap document [Roadmap].

### A.2. Already Implemented

A minimal portable Discovery Proxy implementation has been produced by Markus Stenberg and Steven Barth, which runs on OS X and several Linux variants including OpenWrt [ohp]. It was demonstrated at the Berlin IETF in July 2013.

Tom Pusateri has an implementation that runs on any Unix/Linux. It has a RESTful interface for management and an experimental demo CLI and web interface.

Ted Lemon also has produced a portable implementation of Discovery Proxy, which is available in the mDNSResponder open source code.

The Long-Lived Query mechanism [LLQ] referred to in this specification exists and is deployed, but was not standardized by the IETF. The IETF has developed a superior Long-Lived Query mechanism called DNS Push Notifications [Push], which is built on DNS Stateful Operations [RFC8490]. The pragmatic short-term deployment approach is for vendors to produce Discovery Proxies that implement both the

deployed Long-Lived Query mechanism [LLQ] (for today's clients) and the new DNS Push Notifications mechanism [Push] as the preferred long-term direction.

#### A.3. Partially Implemented

The current APIs make multiple domains visible to client software, but most client UI today lumps all discovered services into a single flat list. This is largely a chicken-and-egg problem. Application writers were naturally reluctant to spend time writing domain-aware UI code when few customers today would benefit from it. If Discovery Proxy deployment becomes common, then application writers will have a reason to provide better UI. Existing applications will work with the Discovery Proxy, but will show all services in a single flat list. Applications with improved UI will group services by domain.

#### Author's Address

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

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

Network Working Group  
Internet-Draft  
Intended status: Standards Track  
Expires: August 26, 2021

T. Lemon  
S. Cheshire  
Apple Inc.  
February 22, 2021

Multicast DNS Discovery Relay  
draft-ietf-dnssd-mdns-relay-04

## Abstract

This document complements the specification of the Discovery Proxy for Multicast DNS-Based Service Discovery. It describes a lightweight relay mechanism, a Discovery Relay, which, when present on a link, allows remote clients, not attached to that link, to perform mDNS discovery operations on that link.

## Status of This Memo

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

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

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

This Internet-Draft will expire on August 26, 2021.

## Copyright Notice

Copyright (c) 2021 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.



## Table of Contents

1. Introduction . . . . .	3
2. Terminology . . . . .	4
3. Protocol Overview . . . . .	6
3.1. Connections between Clients and Relays (overview) . . . . .	6
3.2. mDNS Messages On Multicast Links . . . . .	7
4. Connections between Clients and Relays (details) . . . . .	8
5. Traffic from Relays to Clients . . . . .	10
6. Traffic from Clients to Relays . . . . .	12
7. Discovery Proxy Behavior . . . . .	13
8. DSO TLVs . . . . .	14
8.1. mDNS Link Data Request . . . . .	14
8.2. mDNS Link Data Discontinue . . . . .	14
8.3. Link Identifier . . . . .	15
8.4. Encapsulated mDNS Message . . . . .	15
8.5. IP Source . . . . .	15
8.6. Link State Request . . . . .	16
8.7. Link State Discontinue . . . . .	16
8.8. Link Available . . . . .	16
8.9. Link Unavailable . . . . .	16
8.10. Link Prefix . . . . .	17
9. Provisioning . . . . .	18
9.1. Provisioned Objects . . . . .	19
9.1.1. Multicast Link . . . . .	20
9.1.2. Discovery Proxy . . . . .	21
9.1.3. Discovery Relay . . . . .	22
9.2. Configuration Files . . . . .	23
9.3. Discovery Proxy Private Configuration . . . . .	25
9.4. Discovery Relay Private Configuration . . . . .	25
10. Security Considerations . . . . .	26
11. IANA Considerations . . . . .	27
12. Acknowledgments . . . . .	27
13. References . . . . .	28
13.1. Normative References . . . . .	28
13.2. Informative References . . . . .	29
Authors' Addresses . . . . .	30

## 1. Introduction

This document defines a Discovery Relay. A Discovery Relay is a companion technology that works in conjunction with Discovery Proxies, and other clients.

The Discovery Proxy for Multicast DNS-Based Service Discovery [RFC8766] is a mechanism for discovering services on a subnetted network through the use of Discovery Proxies. Discovery Proxies issue Multicast DNS (mDNS) requests [RFC6762] on various multicast links in the network on behalf of a remote host performing DNS-Based Service Discovery [RFC6763].

In the original Discovery Proxy specification, it was imagined that for every multicast link on which services will be discovered, a host will be present running a full Discovery Proxy. This document introduces a lightweight Discovery Relay that can be used in conjunction with a central Discovery Proxy to provide discovery services on a multicast link without requiring a full Discovery Proxy on every multicast link.

The primary purpose of a Discovery Relay is providing remote virtual interface functionality to Discovery Proxies, and this document is written with that usage in mind. However, in principle, a Discovery Relay could be used by any properly authorized client. In the context of this specification, a Discovery Proxy is a client to the Discovery Relay. This document uses the terms "Discovery Proxy" and "Client" somewhat interchangeably; the term "Client" is used when we are talking about the communication between the Client and the Relay, and the term "Discovery Proxy" when we are referring specifically to a Discovery Relay Client that also happens to be a Discovery Proxy. One example of another kind of device that can be a client of a Discovery Relay is an Advertising Proxy [AdProx].

The Discovery Relay operates by listening for TCP connections from Clients. When a Client connects, the connection is authenticated and secured using TLS. The Client can then specify one or more multicast links from which it wishes to receive mDNS traffic. The Client can also send messages to be transmitted on its behalf on one or more of those multicast links. DNS Stateful Operations (DSO) [RFC8490] is used as a framework for conveying interface and IP header information associated with each message. DSO formats its messages using type-length-value (TLV) data structures. This document defines additional DSO TLV types, used to implement the Discovery Relay functionality.

The Discovery Relay functions essentially as a set of one or more remote virtual interfaces for the Client, one on each multicast link to which the Discovery Relay is connected. In a complex network, it

is possible that more than one Discovery Relay will be connected to the same multicast link; in this case, the Client ideally should only be using one such Relay Proxy per multicast link, since using more than one will generate duplicate traffic.

How such duplication is detected and avoided is out of scope for this document; in principle it could be detected using HNCP [RFC7788] or configured using some sort of orchestration software in conjunction with NETCONF [RFC6241] or CPE WAN Management Protocol [TR-069].

Use of a Discovery Relay can be considered similar to using Virtual LAN (VLAN) trunk ports to give a Discovery Proxy device a virtual presence on multiple links or broadcast domains. The difference is that while a VLAN trunk port operates at the link layer and delivers all link-layer traffic to the Discovery Proxy device, a Discovery Relay operates further up the network stack and selectively delivers only relevant Multicast DNS traffic. Also, VLAN trunk ports are generally only available within a single administrative domain and require link-layer configuration and connectivity, whereas the Discovery Relay protocol, which runs over TCP, can be used between any two devices with IP connectivity to each other.

## 2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in 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.

The following definitions may be of use:

**Client** A network service that uses a Discovery Relay to send and receive mDNS multicast traffic on a remote link, to enable it to communicate with mDNS Agents on that remote link.

**mDNS Agent** A host which sends and/or responds to mDNS queries directly on its local link(s). Examples include network cameras, networked printers, networked home electronics, etc.

**Discovery Proxy** A network service which receives well-formed questions using the DNS protocol, performs multicast DNS queries to find answers to those questions, and responds with those answers using the DNS protocol. A Discovery Proxy that can communicate with remote mDNS Agents, using the services of a Discovery Relay, is a Client of the Discovery Relay.

**Discovery Relay** A network service which relays mDNS messages received on a local link to a Client, and on behalf of that Client can transmit mDNS messages on a local link.

**multicast link** A maximal set of network connection points, such that any host connected to any connection point in the set may send a packet with a link-local multicast destination address (specifically the mDNS link-local multicast destination address [RFC6762]) that will be received by all hosts connected to all other connection points in the set. Note that it is becoming increasingly common for a multicast link to be smaller than its corresponding unicast link. For example it is becoming common to have multiple Wi-Fi access points on a shared Ethernet backbone, where the multiple Wi-Fi access points and their shared Ethernet backbone form a single unicast link (a single IPv4 subnet, or single IPv6 prefix) but not a single multicast link. Unicast packets sent directly between two hosts on that IPv4 subnet or IPv6 prefix, without passing through an intervening IP-layer router, are correctly delivered, but multicast packets are not forwarded between the various Wi-Fi access points. Given the slowness of Wi-Fi multicast [I-D.ietf-mboned-ieee802-mcast-problems], having a packet that may be of interest to only one or two end systems transmitted to hundreds of devices, across multiple Wi-Fi access points, is especially wasteful. Hence the common configuration decision to not forward multicast packets between Wi-Fi access points is very reasonable. This further motivates the need for technologies like Discovery Proxy and Discovery Relay to facilitate discovery on these networks.

**allow-list** A list of one or more IP addresses from which a Discovery Relay may accept connections.

**silently discard** When a message that is not supported or not permitted is received, and the required response to that message is to "silently discard" it, that means that no response is sent by the service that is discarding the message to the service that sent it. The service receiving the message may log the event, and may also count such events: "silently" does not preclude such behavior.

Take care when reading this document not to confuse the terms "Discovery Proxy" and "Discovery Relay". A Discovery Proxy [RFC8766] provides Multicast DNS discovery service to remote clients. A Discovery Relay is a simple software entity that provides virtual link connectivity to one or more Discovery Proxies or other Discovery Relay clients.

### 3. Protocol Overview

This document describes a way for a Client to communicate with mDNS agents on remote multicast links to which the client is not directly connected, using a Discovery Relay. As such, there are two parts to the protocol: connections between Clients and Discovery Relays, and communications between Discovery Relays and mDNS agents.

#### 3.1. Connections between Clients and Relays (overview)

Discovery Relays listen for incoming connection requests. Connections between Clients and Discovery Relays are established by Clients. Connections are authenticated and encrypted using TLS, with both client and server certificates. Connections are long-lived: a Client is expected to send many queries over a single connection, and Discovery Relays will forward all mDNS traffic from subscribed interfaces over the connection.

The stream encapsulated in TLS will carry DNS frames as in the DNS TCP protocol [RFC1035] Section 4.2.2. However, all messages will be DSO messages [RFC8490]. There will be four types of such messages between Discovery Relays and Clients:

- o Control messages from Client to Relay
- o Link status messages from Relay to Client
- o Encapsulated mDNS messages from Client to Relay
- o Encapsulated mDNS messages from Relay to Client

Clients can send four different control messages to Relays: Link State Request, Link State Discontinue, Link Data Request and Link Data Discontinue. The first two are used by the Client to request that the Relay report on the set of links that can be requested, and to request that it discontinue such reporting. The second two are used by the Client to indicate to the Discovery Relay that mDNS messages from one or more specified multicast links are to be relayed to the Client, and to subsequently stop such relaying.

Link Status messages from a Discovery Relay to the Client inform the Client that a link has become available, or that a formerly-available link is no longer available.

Encapsulated mDNS messages from a Discovery Relay to a Client are sent whenever an mDNS message is received on a multicast link to which the Discovery Relay has subscribed.

Encapsulated mDNS messages from a Client to a Discovery Relay cause the Discovery Relay to transmit the mDNS message on the specified multicast link to which the Discovery Relay host is directly attached.

During periods with no traffic flowing, Clients are responsible for generating any necessary keepalive traffic, as stated in the DSO specification [RFC8490].

### 3.2. mDNS Messages On Multicast Links

Discovery Relays listen for mDNS traffic on all configured multicast links that have at least one active subscription from a Client. When an mDNS message is received on a multicast link, it is forwarded on every open Client connection that is subscribed to mDNS traffic on that multicast link. In the event of congestion, where a particular Client connection has no buffer space for an mDNS message that would otherwise be forwarded to it, the mDNS message is not forwarded to it. Normal mDNS retry behavior is used to recover from this sort of packet loss. Discovery Relays are not expected to buffer more than a few mDNS packets. Excess mDNS packets are silently discarded. In practice this is not expected to be a issue. Particularly on networks like Wi-Fi, multicast packets are transmitted at rates ten or even a hundred times slower than unicast packets. This means that even at peak multicast packets rates, it is likely that a unicast TCP connection will be able to carry those packets with ease.

Clients send encapsulated mDNS messages they wish to have sent on their behalf on remote multicast link(s) on which the Client has an active subscription. A Discovery Relay will not transmit mDNS packets on any multicast link on which the Client does not have an active subscription, since it makes no sense for a Client to ask to have a query sent on its behalf if it's not able to receive the responses to that query.

#### 4. Connections between Clients and Relays (details)

When a Discovery Relay starts, it opens a passive TCP listener to receive incoming connection requests from Clients. This listener may be bound to one or more source IP addresses, or to the wildcard address, depending on the implementation. When a connection is received, the relay must first validate that it is a connection to an IP address to which connections are allowed. For example, it may be that only connections to ULAs are allowed, or to the IP addresses configured on certain interfaces. If the listener is bound to a specific IP address, this check is unnecessary.

If the relay is using an IP address allow-list, the next step is for the relay to verify that the source IP address of the connection is on its allow-list. If the connection is not permitted either because of the source address or the destination address, the Discovery Relay closes the connection. If possible, before closing the connection, the Discovery Relay first sends a TLS `user_canceled` alert ([RFC8446] Section 6.1). Discovery Relays SHOULD refuse to accept TCP connections to invalid destination addresses, rather than accepting and then closing the connection, if this is possible.

Otherwise, the Discovery Relay will attempt to complete a TLS handshake with the Client. Clients are required to send the `post_handshake_auth` extension ([RFC8446] Section 4.2.5). If a Discovery Relay receives a `ClientHello` message with no `post_handshake_auth` extension, the Discovery Relay rejects the connection with a `certificate_required` alert ([RFC8446] Section 6.2).

Once the TLS handshake is complete, the Discovery Relay MUST request post-handshake authentication ([RFC8446] Section 4.6.2). If the Client refuses to send a certificate, or the key presented does not match the key associated with the IP address from which the connection originated, or the `CertificateVerify` does not validate, the connection is dropped with the `TLS access_denied` alert ([RFC8446] Section 6.2).

Clients MUST validate server certificates. If the client is configured with a server IP address and certificate, it can validate the server by comparing the certificate offered by the server to the certificate that was provided: they should be the same. If the certificate includes a Distinguished Name that is a fully-qualified domain name, the client SHOULD present that domain name to the server in an SNI request.

Rather than being configured with an IP address and a certificate, the client may be configured with the server's FQDN. In this case, the client uses the server's FQDN as a Authentication Domain Name

[RFC8310] Section 7.1, and uses the authentication method described in [RFC8310] section 8.1, if the certificate is signed by a root authority the client trusts, or the method described in section 8.2 of the same document if not. If neither method is available, then a locally-configured copy of the server certificate can be used, as in the previous paragraph.

Once the connection is established and authenticated, it is treated as a DNS TCP connection [RFC7766].

Aliveness of connections between Clients and Relays is maintained as described in Section 4 of the DSO specification [RFC8490]. Clients must also honor the 'Retry Delay' TLV (section 5 of [RFC8490]) if sent by the Discovery Relay.

Clients SHOULD avoid establishing more than one connection to a specific Discovery Relay. However, there may be situations where multiple connections to the same Discovery Relay are unavoidable, so Discovery Relays MUST be willing to accept multiple connections from the same Client.

In order to know what links to request, the Client can be configured with a list of links supported by the Relay. However, in some networking contexts, dynamic changes in the availability of links are likely; therefore Clients may also use the Report Link Changes TLV to request that the Relay report on the availability of its links. In some contexts, for example when debugging, a Client may operate with no information about the set of links supported by a relay, simply relying on the relay to provide one.



## 5. Traffic from Relays to Clients

The mere act of connecting to a Discovery Relay does not result in any mDNS traffic being forwarded. In order to request that mDNS traffic from a particular multicast link be forwarded on a particular connection, the Client must send one or more DSO messages, each containing a single mDNS Link Data Request TLV (Section 8.1) indicating the multicast link from which traffic is requested.

When an mDNS Link Data Request message is received, the Discovery Relay validates that it recognizes the link identifier, and that forwarding is enabled for that link. If both checks are successful, it MUST send a response with RCODE=0 (NOERROR). If the link identifier is not recognized, it sends a response with RCODE=3 (NXDOMAIN/Name Error). If forwarding from that link to the Client is not enabled, it sends a response with RCODE=5 (REFUSED). If the relay cannot satisfy the request for some other reason, for example resource exhaustion, it sends a response with RCODE=2 (SERVFAIL).

If the requested link is valid, the Relay begins forwarding all mDNS messages from that link to the Client. Delivery is not guaranteed: if there is no buffer space, packets will be dropped. It is expected that regular mDNS retry processing will take care of retransmission of lost packets. The amount of buffer space is implementation dependent, but generally should not be more than the bandwidth delay product of the TCP connection [RFC7323]. The Discovery Relay should use the TCP\_NOTSENT\_LOWAT mechanism [NOTSENT][PRIO] or equivalent, to avoid building up a backlog of data in excess of the amount necessary to have in flight to fill the bandwidth delay product of the TCP connection.

Encapsulated mDNS messages from Relays to Clients are framed within DSO messages. Each DSO message can contain multiple TLVs, but only a single encapsulated mDNS message is conveyed per DSO message. Each forwarded mDNS message is sent in an Encapsulated mDNS Message TLV (Section 8.4). The source IP address and port of the message MUST be encoded in an IP Source TLV (Section 8.5). The multicast link on which the message was received MUST be encoded in a Link Identifier TLV (Section 8.3). As described in the DSO specification [RFC8490], a Client MUST silently ignore unrecognized Additional TLVs in mDNS messages, and MUST NOT discard mDNS messages that include unrecognized Additional TLVs.

A Client may discontinue listening for mDNS messages on a particular multicast link by sending a DSO message containing an mDNS Link Data Discontinue TLV (Section 8.2). The Discovery Relay MUST discontinue forwarding mDNS messages when the Link Data Discontinue request is received. However, messages from that link that had previously been

queued may arrive after the Client has discontinued its listening. The Client should silently discard such messages. The Discovery Relay does not respond to the Link Data Discontinue message other than to discontinue forwarding mDNS messages from the specified links.

## 6. Traffic from Clients to Relays

Like mDNS traffic from relays, each mDNS message sent by a Client to a Discovery Relay is communicated in an Encapsulated mDNS Message TLV (Section 8.4) within a DSO message. Each message MUST contain exactly one Link Identifier TLV (Section 8.3). The Discovery Relay will transmit the mDNS message to the mDNS port and multicast address on the link specified in the message using the specified IP address family.

Although the communication between Clients and Relays uses the DNS stream protocol and DNS Stateless Operations, there is no case in which a Client would legitimately send a DNS query (or anything else other than a DSO message) to a Relay. Therefore, if a Relay receives any message other than a DSO message, it MUST immediately abort that DSO session with a TCP reset (RST).

When defining this behavior, the working group considered making it possible to specify more than one link identifier in an mDNSMessage TLV. A superficial evaluation of this suggested that this might be a useful optimization, since when a query is issued, it will often be issued to all links. However, on many link types, like Wi-Fi, multicast traffic is expensive [I-D.ietf-mboned-ieee802-mcast-problems] and should be generated frugally, so providing convenient ways to generate additional multicast traffic was determined to be an unwise optimization. In addition, because of the way mDNS handles retries, it will almost never be the case that the exact same message will be sent on more than one link. Therefore, the complexity that this optimization adds is not justified by the potential benefit, and this idea has been abandoned.

## 7. Discovery Proxy Behavior

Discovery Proxies treat multicast links for which Discovery Relay service is being used as if they were virtual interfaces; in other words, a Discovery Proxy serving multiple remote multicast links using multiple remote Discovery Relays behaves the same as a Discovery Proxy serving multiple local multicast links using multiple local physical network interfaces. In this section we refer to multicast links served directly by the Discovery Proxy as locally-connected links, and multicast links served through the Discovery Relay as relay-connected links. A relay-connected link can be thought of as similar to a link that a Discovery Proxy connects to using a USB Ethernet interface, just with a very long USB cable (that runs over TCP).

When a Discovery Proxy receives a DNS query from a DNS client via unicast, it will generate corresponding mDNS query messages on the relevant multicast link(s) for which it is acting as a proxy. For locally-connected link(s), those query messages will be sent directly. For relay-connected link(s), the query messages will be sent through the Discovery Relay that is being used to serve that multicast link.

Responses from devices on locally-connected links are processed normally. Responses from devices on relay-connected links are received by the Discovery Relay, encapsulated, and forwarded to the Client; the Client then processes these messages using the link-identifying information included in the encapsulation.

In principle it could be the case that some device is capable of performing service discovery using Multicast DNS, but not using traditional unicast DNS. Responding to mDNS queries received from the Discovery Relay could address this use case. However, continued reliance on multicast is counter to the goals of the current work in service discovery, and to benefit from wide-area service discovery such client devices should be updated to support service discovery using unicast queries.

## 8. DSO TLVs

This document defines a modest number of new DSO TLVs.

### 8.1. mDNS Link Data Request

The mDNS Link Data Request TLV conveys a link identifier from which a Client is requesting that a Discovery Relay forward mDNS traffic. The link identifier comes from the provisioning configuration (see Section 9). The DSO-TYPE for this TLV is TBD-R. DSO-LENGTH is always 5. DSO-DATA is the 8-bit address family followed by the link identifier, a 32-bit unsigned integer in network (big endian) byte order, as described in Section 9. An address family value of 1 indicates IPv4 and 2 indicates IPv6, as recorded in the IANA Registry of Address Family Numbers [AdFam].

The mDNS Link Data Request TLV can only be used as a primary TLV, and requires an acknowledgement.

At most one mDNS Link Data Request TLV may appear in a DSO message. To request multiple link subscriptions, multiple separate DSO messages are sent, each containing a single mDNS Link Data Request TLV.

A Client MUST NOT request a link if it already has an active subscription to that link on the same DSO connection. If a Discovery Relay receives a duplicate link subscription request, it MUST immediately abort that DSO session with a TCP reset (RST).

### 8.2. mDNS Link Data Discontinue

The mDNS Link Data Discontinue TLV is used by Clients to unsubscribe to mDNS messages on the specified multicast link. DSO-TYPE is TBD-D. DSO-LENGTH is always 5. DSO-DATA is the 8-bit address family followed by the 32-bit link identifier, a 32-bit unsigned integer in network (big endian) byte order, as described in Section 9.

The mDNS Link Data Discontinue TLV can only be used as a DSO unidirectional message TLV, and is not acknowledged.

At most one mDNS Link Data Discontinue TLV may appear in a DSO message. To unsubscribe from multiple links, multiple separate DSO messages are sent, each containing a single mDNS Link Data Discontinue TLV.

### 8.3. Link Identifier

This option is used both in DSO messages from Discovery Relays to Clients that contain received mDNS messages, and from Clients to Discovery Relays that contain mDNS messages to be transmitted on the multicast link. In the former case, it indicates the multicast link on which the message was received; in the latter case, it indicates the multicast link on which the message should be transmitted. DSO-TYPE is TBD-L. DSO-LENGTH is always 5. DSO-DATA is the 8-bit address family followed by the link identifier, a 32-bit unsigned integer in network (big endian) byte order, as described in Section 9.

The Link Identifier TLV can only be used as an additional TLV. The Link Identifier TLV can only appear at most once in a Discovery Relay DSO message.

### 8.4. Encapsulated mDNS Message

The Encapsulated mDNS Message TLV is used to communicate an mDNS message that a Relay is forwarding from a multicast link to a Client, or that a Client is sending to a Relay for transmission on a multicast link. Only the application-layer payload of the mDNS message is carried in the DSO "Encapsulated mDNS Message" TLV, i.e., just the DNS message itself, beginning with the DNS Message ID, not the IP or UDP headers. The DSO-TYPE for this TLV is TBD-M. DSO-LENGTH is the length of the encapsulated mDNS message. DSO-DATA is the content of the encapsulated mDNS message.

The Encapsulated mDNS Message TLV can only be used as a DSO unidirectional message TLV, and is not acknowledged.

### 8.5. IP Source

The IP Source TLV is used to report the IP source address and port from which an mDNS message was received. This TLV is present in DSO messages from Discovery Relays to Clients that contain encapsulated mDNS messages. DSO-TYPE is TBD-S. DSO-LENGTH is either 6, for an IPv4 address, or 18, for an IPv6 address. DSO-DATA is the two-byte source port, followed by the 4- or 16-byte IP Address. Both port and address are in the canonical byte order (i.e., the same representation as used in the UDP and IP packet headers, with no byte swapping).

The IP Source TLV can only be used as an additional TLV. The IP Source TLV can only appear at most once in a Discovery Relay DSO message.

### 8.6. Link State Request

The Link State Request TLV requests that the Discovery Relay report link changes. When the relay is reporting link changes and a new link becomes available, it sends a Link Available message to the Client. When a link becomes unavailable, it sends a Link Unavailable message to the Client. If there are links available when the request is received, then for each such link the relay immediately sends a Link Available Message to the Client. DSO-TYPE is TBD-P. DSO-LENGTH is 0.

The mDNS Link State Request TLV can only be used as a primary TLV, and requires an acknowledgement. The acknowledgment does not contain a Link Available TLV: it is just a response to the Link State Request message.

### 8.7. Link State Discontinue

The Link State Discontinue TLV requests that the Discovery Relay stop reporting on the availability of links supported by the relay. This cancels the effect of a Link State Request TLV. DSO-TYPE is TBD-Q. DSO-LENGTH is 0.

The mDNS Link State Discontinue TLV can only be used as a DSO unidirectional message TLV, and is not acknowledged.

### 8.8. Link Available

The Link Available TLV is used by Discovery Relays to indicate to Clients that a new link has become available. The format is the same as the Link Identifier TLV. DSO-TYPE is TBD-V. The Link Available TLV may be accompanied by one or more Link Prefix TLVs which indicate IP prefixes the Relay knows to be present on the link.

The mDNS Link Available TLV can only be used as a DSO unidirectional message TLV, and is not acknowledged.

### 8.9. Link Unavailable

The Link Unavailable TLV is used by Discovery Relays to indicate to Clients that an existing link has become unavailable. The format is the same as the Link Identifier TLV. DSO-TYPE is TBD-U.

The mDNS Link Unavailable TLV can only be used as a DSO unidirectional message TLV, and is not acknowledged.

#### 8.10. Link Prefix

The Link Prefix TLV represents an IP address or prefix configured on a link. The length is 17 for an IPv6 address or prefix, and 5 for an IPv4 address or prefix. The TLV consists of a prefix length, between 0 and 32 for IPv4 or between 0 and 128 for IPv6, represented as a single byte. This is followed by the IP address, either four or sixteen bytes. DSO-TYPE is TBD-K.

The Link Prefix TLV can only be used as a secondary TLV.



## 9. Provisioning

In order for a Discovery Proxy to use Discovery Relays, it must be configured with sufficient information to identify multicast links on which service discovery is to be supported and, if it is not running on a host that is directly connected to those multicast links, connect to Discovery Relays supporting those multicast links.

A Discovery Relay must be configured both with a set of multicast links to which the host on which it is running is connected, on which mDNS relay service is to be provided, and also with a list of one or more Clients authorized to use it.

On a network supporting DNS Service Discovery using Discovery Relays, more than one different Discovery Relay implementation may be present. While it may be that only a single Discovery Proxy is present, that implementation will need to be able to be configured to interoperate with all of the Discovery Relays that are present. Consequently, it is necessary that a standard set of configuration parameters be defined for both Discovery Proxies and Discovery Relays.

DNS Service Discovery generally operates within a constrained set of links, not across the entire internet. This section assumes that what will be configured will be a limited set of links operated by a single entity or small set of cooperating entities, among which services present on each link should be available to users on that link and every other link. This could be, for example, a home network, a small office network, or even a network covering an entire building or small set of buildings. The set of Discovery Proxies and Discovery Relays within such a network will be referred to in this section as a 'Discovery Domain'.

Depending on the context, several different candidates for configuration of Discovery Proxies and Discovery Relays may be applicable. The simplest such mechanism is a manual configuration file, but regardless of provisioning mechanism, certain configuration information needs to be communicated to the devices, as outlined below.

In the example we provide here, we only refer to configuring of IP addresses, private keys and certificates. It is also possible to use FQDNs to identify servers; this then allows for the use of DANE ([RFC8310] Section 8.2) or PKIX authentication [RFC6125]. Which method is used is to some extent up to the implementation, but at a minimum, it should be possible to associate an IP address with a self-signed certificate, and it should be possible to validate both

self-signed and PKIX-authenticated certificates, with PKIX, DANE or a pre-configured trust anchor.

#### 9.1. Provisioned Objects

Three types of objects must be described in order for Discovery Proxies and Discovery Relays to be provisioned: Discovery Proxies, Multicast Links, and Discovery Relays. "Human-readable" below means actual words or proper names that will make sense to an untrained human being. "Machine-readable" means a name that will be used by machines to identify the entity to which the name refers. Each entity must have a machine-readable name and may have a human-readable name. No two entities can have the same human-readable name. Similarly, no two entities can have the same machine-readable name.

### 9.1.1. Multicast Link

The description of a multicast link consists of:

**link-identifier** A 32-bit identifier that uniquely identifies that link within the Discovery Domain. Each link **MUST** have exactly one such identifier. Link Identifiers do not have any special semantics, and are not intended to be human-readable.

**ldh-name** A fully-qualified domain name for the multicast link that is used to form an LDH domain name as described in section 5.3 of the Discovery Proxy specification [RFC8766]. This name is used to identify the link during provisioning, and must be present.

**hr-name** A human-readable user-friendly fully-qualified domain name for the multicast link. This name **MUST** be unique within the Discovery Domain. Each multicast link **MUST** have exactly one such name. The hr-name **MAY** be the same as the ldh-name. (The hr-name is allowed to contain spaces, punctuation and rich text, but it is not required to do so.)

The ldh-name and hr-name can be used to form the LDH and human-readable domain names as described in [RFC8766], section 5.3.

Note that the ldh-name and hr-name can be used in two different ways.

On a small home network with little or no human administrative configuration, link names may be directly visible to the user. For example, a search in 'home.arpa' on a small home network may discover services on both ethernet.home.arpa and wi-fi.home.arpa. In the case of a home user who has one Ethernet-connected printer and one Wi-Fi-connected printer, discovering that they have one printer on ethernet.home.arpa and another on wi-fi.home.arpa is understandable and meaningful.

On a large corporate network with hundreds of Wi-Fi access points, the individual link names of the hundreds of multicast links are less likely to be useful to end users. In these cases, Discovery Broker functionality [I-D.sctl-discovery-broker] may be used to translate the many link names to something more meaningful to users. For example, in a building with 50 Wi-Fi access points, each with their own link names, services on all the different physical links may be presented to the user as appearing in 'headquarters.example.com'. In this case, the individual link names can be thought of similar to MAC addresses or IPv6 addresses. They are used internally by the software as unique identifiers, but generally are not exposed to end users.

### 9.1.2. Discovery Proxy

The description of a Discovery Proxy consists of:

`name` a machine-readable name used to reference this Discovery Proxy in provisioning.

`hr-name` an optional human-readable name which can appear in provisioning, monitoring and debugging systems. Must be unique within a Discovery Domain.

`certificate` a certificate that identifies the Discovery Proxy. This certificate can be shared across services on the Discovery Proxy Host. The public key in the certificate is used both to uniquely identify the Discovery Proxy and to authenticate connections from it. The certificate should be signed by its own private key.

`private-key` the private key corresponding to the public key in the certificate.

`source-ip-addresses` a list of IP addresses that may be used by the Discovery Proxy when connecting to Discovery Relays. These addresses should be addresses that are configured on the Discovery Proxy Host. They should not be temporary addresses. All such addresses must be reachable within the Discovery Domain.

`public-ip-addresses` a list of IP addresses that a Discovery Proxy listens on to receive requests from clients. This is not used for interoperation with Discovery Relays, but is mentioned here for completeness: the list of addresses listened on for incoming client requests may differ from the 'source-ip-addresses' list of addresses used for issuing outbound connection requests to Discovery Relays. If any of these addresses are reachable from outside of the Discovery Domain, services in that domain will be discoverable outside of the domain.

`multicast links` a list of multicast links on which this Discovery Proxy is expected to provide service

The private key should never be distributed to other hosts; all of the other information describing a Discovery Proxy can be safely shared with Discovery Relays.

In some configurations it may make sense for the Discovery Relay not to have a list of links, but simply to support the set of all links available on relays to which the Discovery Proxy is configured to communicate.

### 9.1.3. Discovery Relay

The description of a Discovery Relay consists of:

`name` a required machine-readable identifier used to reference the relay

`hr-name` an optional human-readable name which can appear in provisioning, monitoring and debugging systems. Must be unique within a Discovery Domain.

`certificate` a certificate that identifies the Discovery Relay. This certificate can be shared across services on the Discovery Relay Host. Indeed, if a Discovery Proxy and Discovery Relay are running on the same host, the same certificate can be used for both. The public key in the certificate uniquely identifies the Discovery Relay and is used by a Discovery Relay Client (e.g., a Discovery Proxy) to verify that it is talking to the intended Discovery Relay after a TLS connection has been established. The certificate must either be signed by its own key, or have a signature chain that can be validated using PKIX authentication [RFC6125].

`private-key` the private key corresponding to the public key in the certificate.

`listen-tuple` a list of IP address/port tuples that may be used to connect to the Discovery Relay. The relay may be configured to listen on all addresses on a single port, but this is not required, so the port as well as the address must be specified.

`multicast links` a list of multicast links to which this relay is physically connected.

The private key should never be distributed to other hosts; all of the other information describing a Discovery Relay can be safely shared with Discovery Proxies.

In some cases a Relay may not be configured with a static list of links, but may simply discover links by monitoring the set of available interfaces on the host on which the Relay is running. In that case, the relay could be configured to identify links based on the names of network interfaces, or based on the set of available prefixes seen on those interfaces. The details of this sort of configuration are not specified in this document.

## 9.2. Configuration Files

For this discussion, we assume the simplest possible means of configuring Discovery Proxies and Discovery Relays: the configuration file. Any environment where changes will happen on a regular basis will either require some automatic means of generating these configuration files as the network topology changes, or will need to use a more automatic method for configuration, such as HNCP [RFC7788].

There are many different ways to organize configuration files. This discussion assumes that multicast links, relays and proxies will be specified as objects, as described above, perhaps in a master file, and then the specific configuration of each proxy or relay will reference the set of objects in the master file, referencing objects by name. This approach is not required, but is simply shown as an example. In addition, the private keys for each proxy or relay must appear only in that proxy or relay's configuration file.

The master file contains a list of Discovery Relays, Discovery Proxies and Multicast Links. Each object has a name and all the other data associated with it. We do not formally specify the format of the file, but it might look something like this:

```
Relay upstairs
  certificate xxx
  listen-tuple 192.0.2.1 1917
  listen-tuple fd00::1 1917
  link upstairs-wifi
  link upstairs-wired
  client-allow-list main

Relay downstairs
  certificate yyy
  listen-tuple 192.51.100.1 2088
  listen-tuple fd00::2 2088
  link downstairs-wifi
  link downstairs-wired
  client-allow-list main

Proxy main
  certificate zzz
  address 203.1.113.1

Link upstairs-wifi
  id 1
  hr-name Upstairs Wifi

Link upstairs-wired
  id 2
  hr-name Upstairs Wired

Link downstairs-wifi
  id 3
  hr-name Downstairs Wifi

Link downstairs-wired
  id 4
  hr-name Downstairs Wired
```

### 9.3. Discovery Proxy Private Configuration

The Discovery Proxy configuration contains enough information to identify which Discovery Proxy is being configured, enumerate the list of multicast links it is intended to serve, and provide keying information it can use to authenticate to Discovery Relays. It may also contain custom information about the port and/or IP address(es) on which it will respond to DNS queries.

An example configuration, following the convention used in this section, might look something like this:

```
Proxy main
  private-key zzz
  subscribe upstairs-wifi
  subscribe downstairs-wifi
  subscribe upstairs-wired
  subscribe downstairs-wired
```

When combined with the master file, this configuration is sufficient for the Discovery Proxy to identify and connect to the Discovery Relays that serve the links it is configured to support.

### 9.4. Discovery Relay Private Configuration

The Discovery Relay configuration just needs to tell the Discovery Relay what name to use to find its configuration in the master file, and what the private key is corresponding to its certificate (public key) in the master file. For example:

```
Relay Downstairs
  private-key yyy
```



## 10. Security Considerations

Part of the purpose of the Multicast DNS Discovery Relay protocol is to place a simple relay, analogous to a BOOTP relay, into routers and similar devices that may not be updated frequently. The BOOTP [RFC0951] protocol has been around since 1985, and continues to be useful today. The BOOTP protocol uses no encryption, and in many enterprise networks this is considered acceptable. In contrast, the Discovery Relay protocol requires TLS 1.3. A concern is that after 20 or 30 years, TLS 1.3, or some of the encryption algorithms it uses, may become obsolete, rendering devices that require it unusable. Our assessment is that TLS 1.3 probably will be around for many years to come. TLS 1.0 [RFC2246] was used for about a decade, and similarly TLS 1.2 [RFC5246] was also used for about a decade. We expect TLS 1.3 [RFC8446] to have at least that lifespan. In addition, recent IETF efforts are pushing for better software update practices for devices like routers, for other security reasons, making it likely that in ten years time it will be less common to be using routers that haven't had a software update for ten years. However, authors of encryption specifications and libraries should be aware of the potential backwards compatibility issues if an encryption algorithm becomes deprecated. This specification RECOMMENDS that if an encryption algorithm becomes deprecated, then rather than remove that encryption algorithm entirely, encryption libraries should disable that encryption algorithm by default, but leave the code present with an option for client software to enable it in special cases, such as a recent Client talking to an ancient Discovery Relay. Using no encryption, like BOOTP, would eliminate this backwards compatibility concern, but we feel that in such a future hypothetical scenario, using even a weak encryption algorithm still makes passive eavesdropping and tampering harder, and is preferable to using no encryption at all.

## 11. IANA Considerations

The IANA is kindly requested to update the DSO Type Codes Registry [RFC8490] by allocating codes for each of the TBD type codes listed in the following table, and by updating this document, here and in Section 8. Each type code should list this document as its reference document.

DSO-TYPE	Status	Name
TBD-R	Standard	Link Data Request
TBD-D	Standard	Link Data Discontinue
TBD-L	Standard	Link Identifier
TBD-M	Standard	Encapsulated mDNS Message
TBD-S	Standard	IP Source
TBD-P	Standard	Link State Request
TBD-Q	Standard	Link State Discontinue
TBD-V	Standard	Link Available
TBD-U	Standard	Link Unavailable
TBD-K	Standard	Link Prefix

DSO Type Codes to be allocated

## 12. Acknowledgments

Thanks to Derek Atkins for the secdir early review.

## 13. References

### 13.1. Normative References

- [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>>.
- [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>>.
- [RFC6125] Saint-Andre, P. and J. Hodges, "Representation and Verification of Domain-Based Application Service Identity within Internet Public Key Infrastructure Using X.509 (PKIX) Certificates in the Context of Transport Layer Security (TLS)", RFC 6125, DOI 10.17487/RFC6125, March 2011, <<https://www.rfc-editor.org/info/rfc6125>>.
- [RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <<https://www.rfc-editor.org/info/rfc6241>>.
- [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>>.
- [RFC7323] Borman, D., Braden, B., Jacobson, V., and R. Scheffenegger, Ed., "TCP Extensions for High Performance", RFC 7323, DOI 10.17487/RFC7323, September 2014, <<https://www.rfc-editor.org/info/rfc7323>>.
- [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>>.
- [RFC7788] Stenberg, M., Barth, S., and P. Pfister, "Home Networking Control Protocol", RFC 7788, DOI 10.17487/RFC7788, April 2016, <<https://www.rfc-editor.org/info/rfc7788>>.

- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8310] Dickinson, S., Gillmor, D., and T. Reddy, "Usage Profiles for DNS over TLS and DNS over DTLS", RFC 8310, DOI 10.17487/RFC8310, March 2018, <<https://www.rfc-editor.org/info/rfc8310>>.
- [RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <<https://www.rfc-editor.org/info/rfc8446>>.
- [RFC8490] Bellis, R., Cheshire, S., Dickinson, J., Dickinson, S., Lemon, T., and T. Pusateri, "DNS Stateful Operations", RFC 8490, DOI 10.17487/RFC8490, March 2019, <<https://www.rfc-editor.org/info/rfc8490>>.
- [RFC8766] Cheshire, S., "Discovery Proxy for Multicast DNS-Based Service Discovery", RFC 8766, DOI 10.17487/RFC8766, June 2020, <<https://www.rfc-editor.org/info/rfc8766>>.

### 13.2. Informative References

- [AdFam] "IANA Address Family Numbers Registry", <<https://www.iana.org/assignments/address-family-numbers/>>.
- [AdProx] Cheshire, S. and T. Lemon, "Advertising Proxy for DNS-SD Service Registration Protocol", draft-sctl-advertising-proxy-00 (work in progress), July 2020.
- [I-D.ietf-mboned-ieee802-mcast-problems] Perkins, C., McBride, M., Stanley, D., Kumari, W., and J. Zuniga, "Multicast Considerations over IEEE 802 Wireless Media", draft-ietf-mboned-ieee802-mcast-problems-12 (work in progress), October 2020.
- [I-D.sctl-discovery-broker] Cheshire, S. and T. Lemon, "Service Discovery Broker", draft-sctl-discovery-broker-00 (work in progress), July 2017.
- [NOTSENT] Dumazet, E., "TCP\_NOTSENT\_LOWAT socket option", July 2013, <<https://lwn.net/Articles/560082/>>.

- [PRIO] Chan, W., "Prioritization Only Works When There's Pending Data to Prioritize", January 2014, <<https://insouciant.org/tech/prioritization-only-works-when-theres-pending-data-to-prioritize/>>.
- [RFC0951] Croft, W. and J. Gilmore, "Bootstrap Protocol", RFC 951, DOI 10.17487/RFC0951, September 1985, <<https://www.rfc-editor.org/info/rfc951>>.
- [RFC2246] Dierks, T. and C. Allen, "The TLS Protocol Version 1.0", RFC 2246, DOI 10.17487/RFC2246, January 1999, <<https://www.rfc-editor.org/info/rfc2246>>.
- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <<https://www.rfc-editor.org/info/rfc5246>>.
- [TR-069] Broadband Forum, "CPE WAN Management Protocol", November 2013, <[https://www.broadband-forum.org/technical/download/TR-069\\_Amendment-5.pdf](https://www.broadband-forum.org/technical/download/TR-069_Amendment-5.pdf)>.

#### Authors' Addresses

Ted Lemon  
Apple Inc.  
One Apple Park Way  
Cupertino, California 95014  
United States of America

Phone: +1 (408) 996-1010  
Email: [elemen@apple.com](mailto:elemen@apple.com)

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

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

Internet Engineering Task Force  
Internet-Draft  
Intended status: Standards Track  
Expires: April 15, 2020

T. Pusateri  
Unaffiliated  
S. Cheshire  
Apple Inc.  
October 13, 2019

DNS Push Notifications  
draft-ietf-dnssd-push-25

Abstract

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

Status of This Memo

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

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

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

This Internet-Draft will expire on April 15, 2020.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect

to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	3
1.1. Requirements Language . . . . .	3
1.2. Fatal Errors . . . . .	3
2. Motivation . . . . .	4
3. Overview . . . . .	5
4. State Considerations . . . . .	6
5. Transport . . . . .	7
6. Protocol Operation . . . . .	8
6.1. Discovery . . . . .	9
6.2. DNS Push Notification SUBSCRIBE . . . . .	13
6.2.1. SUBSCRIBE Request . . . . .	13
6.2.2. SUBSCRIBE Response . . . . .	16
6.3. DNS Push Notification Updates . . . . .	20
6.3.1. PUSH Message . . . . .	20
6.4. DNS Push Notification UNSUBSCRIBE . . . . .	26
6.4.1. UNSUBSCRIBE Message . . . . .	26
6.5. DNS Push Notification RECONFIRM . . . . .	28
6.5.1. RECONFIRM Message . . . . .	29
6.6. DNS Stateful Operations TLV Context Summary . . . . .	31
6.7. Client-Initiated Termination . . . . .	32
6.8. Client Fallback to Polling . . . . .	33
7. Security Considerations . . . . .	34
7.1. Security Services . . . . .	35
7.2. TLS Name Authentication . . . . .	35
7.3. TLS Early Data . . . . .	36
7.4. TLS Session Resumption . . . . .	36
8. IANA Considerations . . . . .	37
9. Acknowledgements . . . . .	37
10. References . . . . .	38
10.1. Normative References . . . . .	38
10.2. Informative References . . . . .	40
Authors' Addresses . . . . .	42

## 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.

### 1.2. Fatal Errors

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

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



## 2. Motivation

As the domain name system continues to adapt to new uses and changes in deployment, polling has the potential to burden DNS servers at many levels throughout the network. Other network protocols have successfully deployed a publish/subscribe model following the Observer design pattern [obs]. 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 address, and changes are sent to that multicast group address for all group members to receive. Therefore, Multicast DNS already has asynchronous change notification capability. When DNS 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 imitation 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) [RFC8490] running over TLS over TCP, and therefore doesn't need to reinvent existing TCP functionality. Using TCP also gives long-lived low-traffic connections better longevity through NAT gateways without depending on the gateway to support NAT Port Mapping Protocol (NAT-PMP) [RFC6886] or Port Control Protocol (PCP) [RFC6887], or resorting to excessive keepalive traffic.

### 3. Overview

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

The DNS Push Notification server for a DNS zone is any server capable of generating the correct change notifications for a name. It may be a primary, secondary, or stealth name server [RFC7719].

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

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

DNS Push Notifications impose less load on the responding server than rapid polling would, but Push Notifications do still have a cost, 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 on 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 [RFC8490], a client must not keep a DSO session to a server open indefinitely if it has no subscriptions (or other operations) active on that session. A client may close a DSO session immediately it becomes idle, and then if needed in the future, open a new session when required. Alternatively, a client may speculatively keep an idle DSO session open for some time, subject to the constraint that it must not keep a session open that has been idle for more than the session's idle timeout (15 seconds by default) [RFC8490].

Note that a DSO session that has an active DNS Push Notification subscription is not considered idle, even if there is no traffic flowing for an extended period of time. In this case the DSO inactivity timeout does not apply, because the session is not inactive, but the keepalive interval does still apply, to ensure generation of sufficient messages to maintain state in middleboxes (such as NAT gateways or firewalls) and for the client and server to periodically verify that they still have connectivity to each other. This is described in Section 6.2 of the DSO specification [RFC8490].

#### 4. State Considerations

Each DNS Push Notification server is capable of handling some finite number of Push Notification subscriptions. This number will vary from server to server and is based on physical machine characteristics, network 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.

## 5. 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 [RFC8490] running over TLS over TCP [RFC7858].

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

Use of TCP also allows DNS Push Notifications to take advantage of current and future developments in TCP, such as Multipath TCP (MPTCP) [RFC6824], TCP Fast Open (TFO) [RFC7413], the TCP RACK fast loss detection algorithm [I-D.ietf-tcpm-rack], and so on.

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

## 6. Protocol Operation

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

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

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

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

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

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

### 6.1. Discovery

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

The client begins by opening a DSO Session to its normal configured DNS recursive resolver and requesting a Push Notification subscription. This connection is made to TCP port 853, the default port for DNS-over-TLS [RFC7858]. If the request for a Push Notification subscription is successful, and the recursive resolver doesn't already have an active subscription for that name, type, and class, then the recursive resolver will make a corresponding Push Notification subscription on the client's behalf. Results received are relayed to the client. This is closely analogous to how a client sends a normal DNS query to its configured DNS recursive resolver which, if it doesn't already have appropriate answer(s) in its cache, issues an upstream query to satisfy the request.

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

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

If the recursive resolver does support DSO 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 SERVFAIL to the client. This includes being unable to establish a connection to the zone's DNS Push Notification server or establishing a connection but receiving a non success response code. In some cases, where the client has a pre-established trust relationship with the owner of the zone (that is not handled via the usual mechanisms for VPN software) the client may handle these failures by contacting the zone's DNS Push 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.headoffice.example.com` (to discover Internet Printing Protocol (IPP) printers [RFC8010] [RFC8011] being advertised in the head office of Example Company.). The client begins by sending an SOA query for `_ipp._tcp.headoffice.example.com` to the local recursive resolver. The goal is to determine the server authoritative for the name `_ipp._tcp.headoffice.example.com`. The closest enclosing DNS zone containing the name `_ipp._tcp.headoffice.example.com` could be `example.com`, or `headoffice.example.com`, or `_tcp.headoffice.example.com`, or even `_ipp._tcp.headoffice.example.com`. The client does not know in advance where the closest enclosing zone cut occurs, which is why it uses the iterative procedure described here to discover this information.
2. If the requested SOA record exists, it will be returned in the Answer section with a NOERROR response code, and the client has succeeded in discovering the information it needs.  
(This language is not placing any new requirements on DNS recursive resolvers. This text merely describes the existing operation of the DNS protocol [RFC1034] [RFC1035].)
3. If the requested SOA record does not exist, the client will get back a NOERROR/NODATA response or an NXDOMAIN/Name Error response. In either case, the local resolver would normally include the SOA record for the closest enclosing zone of the requested name in the Authority Section. If the SOA record is received in the Authority Section, then the client has succeeded in discovering the information it needs.  
(This language is not placing any new requirements on DNS recursive resolvers. This text merely describes the existing

operation of the DNS protocol regarding negative responses [RFC2308].)

4. If the client receives a response containing no SOA record, then it proceeds with the iterative approach. The client strips the leading label from the current query name, and if the resulting name has at least two labels in it, the client sends an SOA query for that new name, and processing continues at step 2 above, repeating the iterative search until either an SOA is received, or the query name consists of a single label, i.e., a Top Level Domain (TLD). In the case of a single-label name (TLD), this is a network configuration error, which should not happen, and the client gives up. The client may retry the operation at a later time, of the client's choosing, such 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, it SHOULD repeat the discovery process in order to determine the preferred DNS server for that subscription at that time. If a client already has a DSO session with that DNS server the client SHOULD reuse that existing DSO session for the new subscription, otherwise, a new DSO session is established. The client MUST respect the DNS TTL values on records it receives while performing the discovery process and store them in its local cache with this lifetime (as it will generally be do anyway for all DNS queries it performs). This means that, as long as the DNS TTL values on the authoritative records are set to reasonable values, repeated application of the discovery process can be completed 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 [RFC8490]. This specification defines a primary DSO TLV for DNS Push Notification SUBSCRIBE Requests (tentatively DSO Type Code 0x40).

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

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

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

### 6.2.1. SUBSCRIBE Request

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

The MESSAGE ID field MUST be set to a unique value, that the client is not using for any other active operation on this DSO session. For the purposes here, a MESSAGE ID is in use on this session if 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 [RFC8490]. The DNS OPCODE field contains the OPCODE value for DNS Stateful Operations (6). The four count fields must be zero, and the corresponding four sections must be empty (i.e., absent).

The DSO-TYPE is SUBSCRIBE (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.

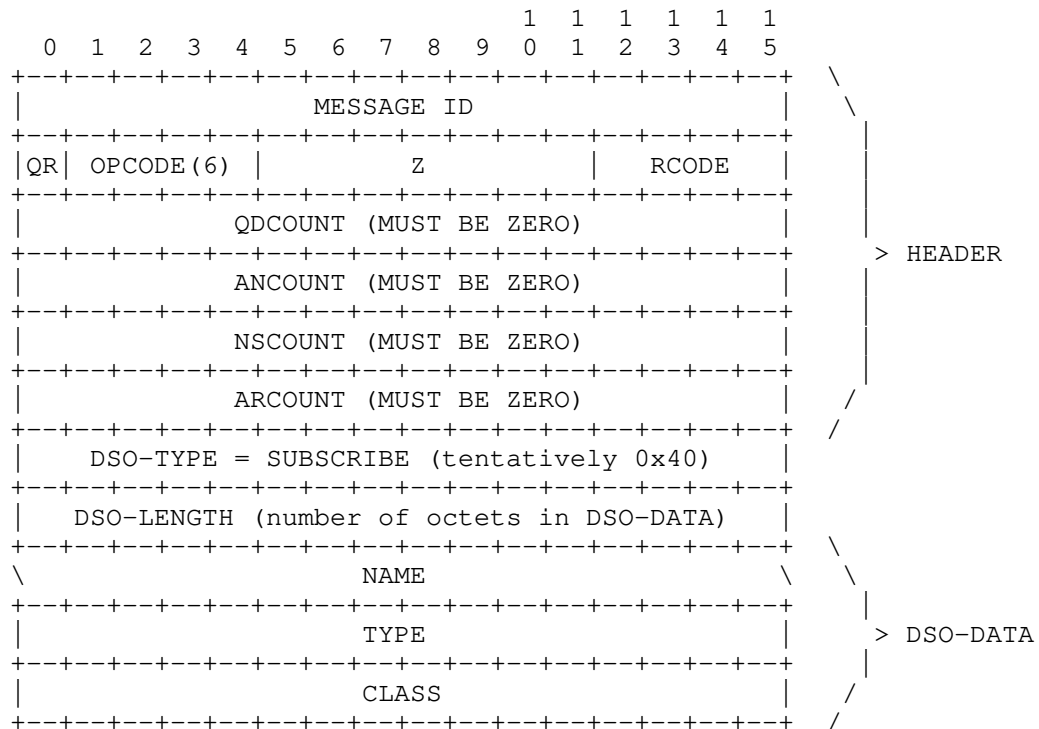


Figure 1: SUBSCRIBE Request

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

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

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

DNS wildcarding is not supported. That is, 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 no other records with the same owner name.

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

If neither TYPE nor CLASS are ANY (255) then this is a specific subscription to changes for the given NAME, TYPE and CLASS. If one or both of TYPE or CLASS are ANY (255) then this subscription matches 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 = 255 (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

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

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

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

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

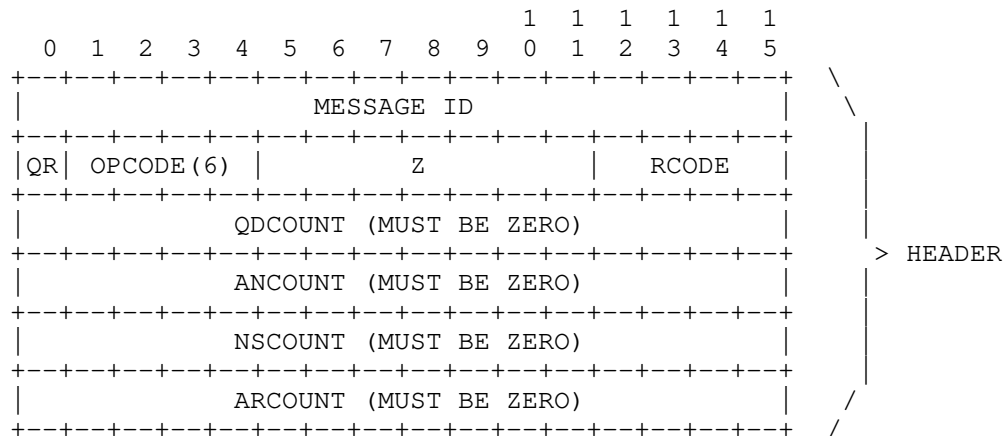


Figure 2: SUBSCRIBE Response

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

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

Table 1: SUBSCRIBE Response codes

This document specifies only these RCODE values for SUBSCRIBE Responses. Servers sending SUBSCRIBE Responses SHOULD use one of these values. Note that NXDOMAIN is not a valid RCODE in response to a SUBSCRIBE Request. However, future circumstances may create situations where other RCODE values are appropriate in SUBSCRIBE Responses, so clients MUST be prepared to accept 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, or
- b. the server resources are exhausted, or
- c. there is some other unknown failure on the server.

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

If the client has other successful subscriptions to this server, these subscriptions remain even though additional subscriptions may be refused. Neither the client nor the server 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 [RFC8490] to the response specifying a delay before the

client attempts this operation again. Recommended values for the delay for different RCODE values are given below. These recommended values apply both to the default values a server should place in the Retry Delay 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 [RFC8490], it is unlikely that the server will begin supporting DSO in the next few minutes, so the retry delay SHOULD be one hour. Notethat 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 [RFC8490]. 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.

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

#### 6.3.1. PUSH Message

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

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

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

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

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

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

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

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

There are three types of collective remove notification:

For collective remove notifications, if CLASS is not 255 (ANY) and TYPE is not 255 (ANY) then for the given name this removes all records of the specified type in the specified class.

For collective remove notifications, if CLASS is not 255 (ANY) and TYPE is 255 (ANY) then for the given name this removes all records of all types in the specified class.

For collective remove notifications, if CLASS is 255 (ANY), then for the given name this removes all records of all types in all classes. In this case TYPE MUST be set to zero on transmission, and MUST be silently ignored on reception.

## Summary of change notification types:

Remove all RRsets from a name, in all classes  
TTL = 0xFFFFFFFFE, RDLEN = 0, CLASS = 255 (ANY)

Remove all RRsets from a name, in given class:  
TTL = 0xFFFFFFFFE, RDLEN = 0, CLASS gives class, TYPE = 255 (ANY)

Remove specified RRset from a name, in given class:  
TTL = 0xFFFFFFFFE, RDLEN = 0  
CLASS and TYPE specify the RRset being removed

Remove an individual RR from a name:  
TTL = 0xFFFFFFFF  
CLASS, TYPE, RDLEN and RDATA specify the RR being removed

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

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

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

For efficiency, when generating a PUSH message, a server SHOULD include as many change notifications as it has immediately available to send, rather than sending each change notification as a separate DSO message. Once it has exhausted the list of change notifications immediately available to send, a server SHOULD then send the PUSH message immediately, rather than waiting to see if additional change notifications become available.

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

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

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

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

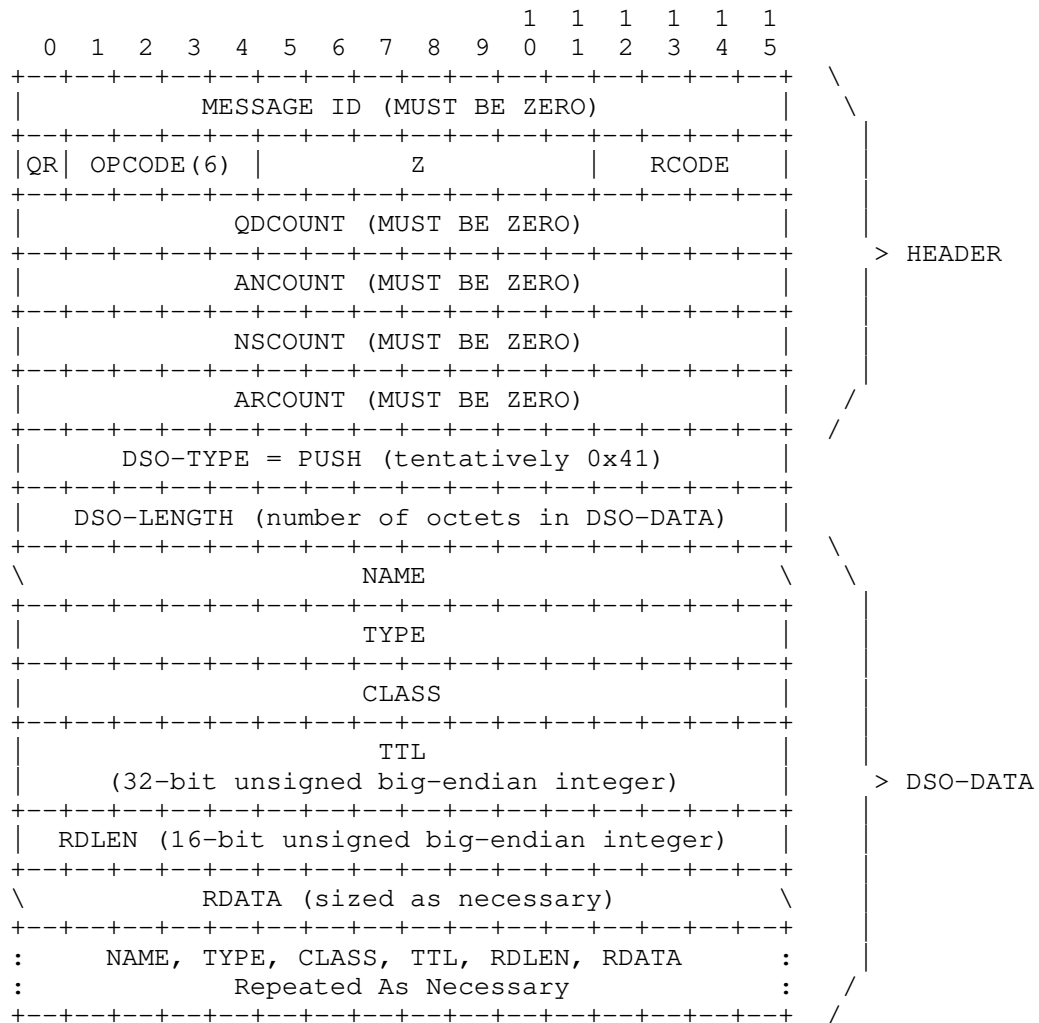


Figure 3: PUSH Message

When processing the records received in a PUSH Message, the receiving client MUST validate that the records being added or removed correspond with at least one currently active subscription on that session. Specifically, the record name MUST match the name given in the SUBSCRIBE request, subject to the usual established DNS case-insensitivity for US-ASCII letters. For individual additions and removals, if the TYPE in the SUBSCRIBE request was not ANY (255) then the TYPE of the record must match the TYPE given in the SUBSCRIBE request, and if the CLASS in the SUBSCRIBE request was not ANY (255) then the CLASS of the record must match the CLASS given in the

SUBSCRIBE request. For collective removals, at least one of the records being removed must match an active subscription. If a matching active subscription on that session is not found, then that particular addition/removal record is silently ignored. Processing of other additions and removal records in this message is not affected. The DSO session is not closed. This is to allow for the unavoidable race condition where a client sends an outbound UNSUBSCRIBE while inbound PUSH messages for that subscription from the server are still in flight.

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

The server SHOULD encode change notifications in the most efficient manner possible. For example, when three AAAA records are removed from a given name, and no other AAAA records exist for that name, the server SHOULD send a "remove an RRset from a name" PUSH message, not three separate "remove an individual RR from a name" PUSH messages. Similarly, when both an SRV and a TXT record are removed from a given name, and no other records of any kind exist for that name, the server SHOULD send a "remove all RRsets from a name" PUSH message, not two separate "remove 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 entity that initiates an UNSUBSCRIBE message is by definition the client. A server **MUST NOT** send an UNSUBSCRIBE message over an existing session from a client. If a server does send an UNSUBSCRIBE message over a DSO session initiated by a client, or an UNSUBSCRIBE message is sent with the QR bit set indicating that it is a response, this is a fatal error and the receiver **MUST** forcibly abort the connection immediately.

##### 6.4.1. UNSUBSCRIBE Message

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

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

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

The DSO-TYPE is UNSUBSCRIBE (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 previously 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 message, the SUBSCRIBE request is no longer active.

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

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

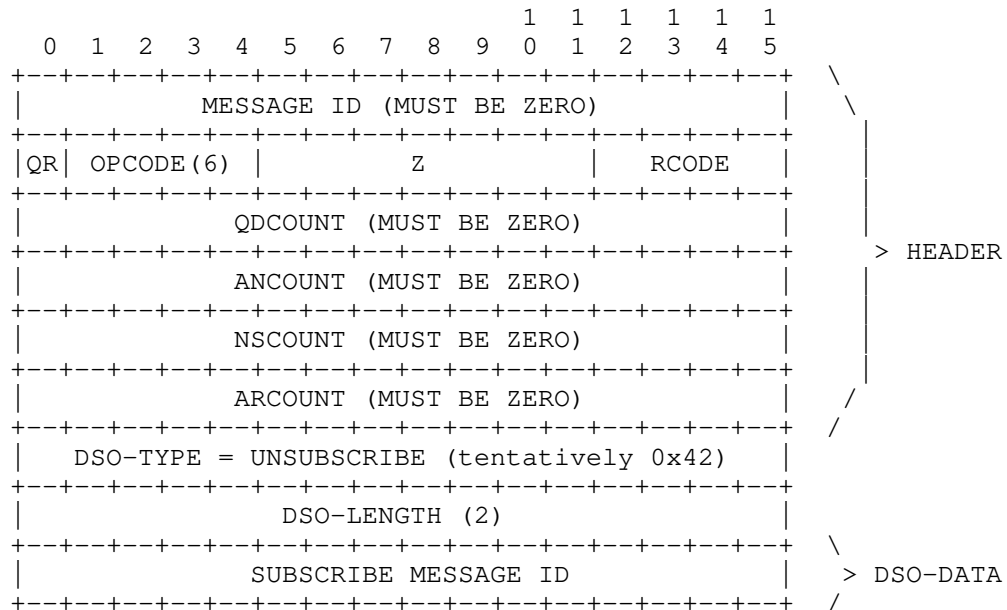


Figure 4: UNSUBSCRIBE Message



## 6.5. DNS Push Notification RECONFIRM

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

For other types of DNS server, the RECONFIRM operation is currently undefined, and SHOULD result in a NOERROR response, but 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 message, the server determines that the disputed records are in fact no longer valid, then subsequent DNS PUSH Messages will be generated to inform interested clients. Thus, one client discovering that a previously-advertised device (like a network printer) is no longer present has the side effect of informing all other interested clients that the device in question is now gone.

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

#### 6.5.1. RECONFIRM Message

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

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

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

The DSO-TYPE is RECONFIRM (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.

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

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 no other records with the same owner name.

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

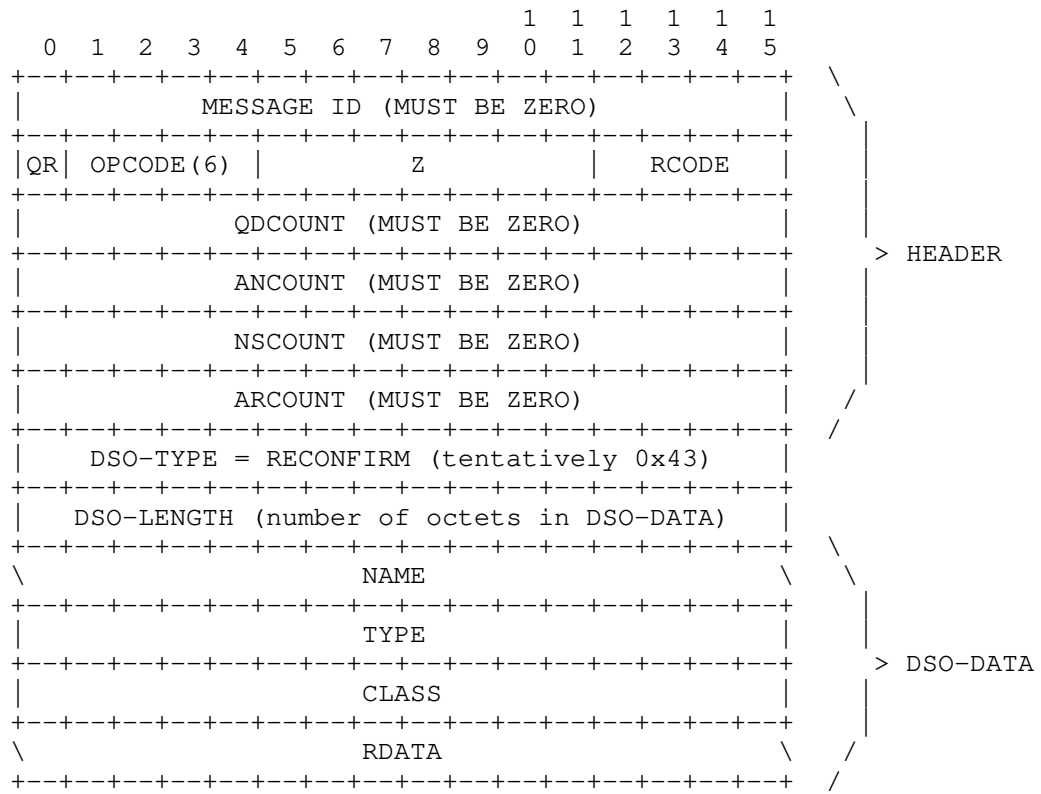


Figure 5: RECONFIRM Message

## 6.6. DNS Stateful Operations TLV Context Summary

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

The client TLV contexts are:

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

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

Table 2: DSO TLV Client Context Summary

The server TLV contexts are:

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

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

Table 3: DSO TLV Server Context Summary

## 6.7. Client-Initiated Termination

An individual subscription is terminated by sending an UNSUBSCRIBE TLV for that specific subscription, or all subscriptions can be 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 no longer interested in receiving those particular updates. It is informing the server that the server may release any state information it has been keeping with regards to these particular subscriptions.

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

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

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

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

## 6.8. Client Fallback to Polling

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

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

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

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

resolver avoids this race condition and unnecessary additional queries it causes.

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

## 7. Security Considerations

The Strict Privacy 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.

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

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

### 7.1. Security Services

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

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

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

**Authentication:** An end-point of the TLS communication is authenticated as the intended entity to communicate with.

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

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

### 7.2. TLS Name Authentication

As described in Section 6.1, the client discovers the DNS Push Notification server using an SRV lookup for the record name "\_dns-push-tls.\_tcp.<zone>". The server connection endpoint SHOULD then be authenticated using DANE TLSA records for the associated SRV record. This associates the target's name and port number with a trusted TLS certificate [RFC7673]. This procedure uses the TLS Server Name Indication (SNI) extension [RFC6066] to inform the server of the name the client has authenticated through the use of TLSA records. Therefore, if the SRV record passes DNSSEC validation and a TLSA record matching the target name is 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 Early Data

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

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

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

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

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

### 7.4. TLS Session Resumption

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

## 8. IANA Considerations

This document defines a new service name, only applicable for the TCP protocol, to be recorded in the IANA Service Type Registry [RFC6335] [SRVTYPE].

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

Table 4: IANA Service Type Assignments

This document defines four new DNS Stateful Operation TLV types to be recorded in the IANA DSO Type Code Registry [RFC8490] [DSOTYPE].

Name	Value	Early Data	Status	Definition
SUBSCRIBE	TBA (0x40)	OK	Standards Track	Section 6.2
PUSH	TBA (0x41)	NO	Standards Track	Section 6.3
UNSUBSCRIBE	TBA (0x42)	NO	Standards Track	Section 6.4
RECONFIRM	TBA (0x43)	NO	Standards Track	Section 6.5

Table 5: IANA DSO TLV Type Code Assignments

This document defines no new DNS OPCODEs or RCODEs.

## 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, Sara Dickinson, Mark Delany, Ralph Droms, Jan Komissar, Eric Rescorla, Michael Richardson, David Schinazi, Manju Shankar Rao, Robert Sparks, Markus Stenberg, Andrew Sullivan, Michael Sweet, Dave Thaler, Brian Trammell, Bernie Volz, Eric Vyncke, Christopher Wood, Liang Xia, and Soraia Zlatkovic. Ted Lemon provided clarifying text that was greatly appreciated.

## 10. References

### 10.1. Normative References

- [DSOTYPE] "DSO Type Code Registry",  
<<https://www.iana.org/assignments/dns-parameters/>>.
- [RFC0020] Cerf, V., "ASCII format for network interchange", STD 80, RFC 20, DOI 10.17487/RFC0020, October 1969,  
<<https://www.rfc-editor.org/info/rfc20>>.
- [RFC0768] Postel, J., "User Datagram Protocol", STD 6, RFC 768, DOI 10.17487/RFC0768, August 1980,  
<<https://www.rfc-editor.org/info/rfc768>>.
- [RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981,  
<<https://www.rfc-editor.org/info/rfc793>>.
- [RFC1034] Mockapetris, P., "Domain names - concepts and facilities", STD 13, RFC 1034, DOI 10.17487/RFC1034, November 1987,  
<<https://www.rfc-editor.org/info/rfc1034>>.
- [RFC1035] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, DOI 10.17487/RFC1035, November 1987, <<https://www.rfc-editor.org/info/rfc1035>>.
- [RFC1123] Braden, R., Ed., "Requirements for Internet Hosts - Application and Support", STD 3, RFC 1123, DOI 10.17487/RFC1123, October 1989,  
<<https://www.rfc-editor.org/info/rfc1123>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997,  
<<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC2136] Vixie, P., Ed., Thomson, S., Rekhter, Y., and J. Bound, "Dynamic Updates in the Domain Name System (DNS UPDATE)", RFC 2136, DOI 10.17487/RFC2136, April 1997,  
<<https://www.rfc-editor.org/info/rfc2136>>.
- [RFC2181] Elz, R. and R. Bush, "Clarifications to the DNS Specification", RFC 2181, DOI 10.17487/RFC2181, July 1997,  
<<https://www.rfc-editor.org/info/rfc2181>>.

- [RFC2782] Gulbrandsen, A., Vixie, P., and L. Esibov, "A DNS RR for specifying the location of services (DNS SRV)", RFC 2782, DOI 10.17487/RFC2782, February 2000, <<https://www.rfc-editor.org/info/rfc2782>>.
- [RFC6066] Eastlake 3rd, D., "Transport Layer Security (TLS) Extensions: Extension Definitions", RFC 6066, DOI 10.17487/RFC6066, January 2011, <<https://www.rfc-editor.org/info/rfc6066>>.
- [RFC6335] Cotton, M., Eggert, L., Touch, J., Westerlund, M., and S. Cheshire, "Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry", BCP 165, RFC 6335, DOI 10.17487/RFC6335, August 2011, <<https://www.rfc-editor.org/info/rfc6335>>.
- [RFC6895] Eastlake 3rd, D., "Domain Name System (DNS) IANA Considerations", BCP 42, RFC 6895, DOI 10.17487/RFC6895, April 2013, <<https://www.rfc-editor.org/info/rfc6895>>.
- [RFC7673] Finch, T., Miller, M., and P. Saint-Andre, "Using DNS-Based Authentication of Named Entities (DANE) TLSA Records with SRV Records", RFC 7673, DOI 10.17487/RFC7673, October 2015, <<https://www.rfc-editor.org/info/rfc7673>>.
- [RFC7766] Dickinson, J., Dickinson, S., Bellis, R., Mankin, A., and D. Wessels, "DNS Transport over TCP - Implementation Requirements", RFC 7766, DOI 10.17487/RFC7766, March 2016, <<https://www.rfc-editor.org/info/rfc7766>>.
- [RFC7858] Hu, Z., Zhu, L., Heidemann, J., Mankin, A., Wessels, D., and P. Hoffman, "Specification for DNS over Transport Layer Security (TLS)", RFC 7858, DOI 10.17487/RFC7858, May 2016, <<https://www.rfc-editor.org/info/rfc7858>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8310] Dickinson, S., Gillmor, D., and T. Reddy, "Usage Profiles for DNS over TLS and DNS over DTLS", RFC 8310, DOI 10.17487/RFC8310, March 2018, <<https://www.rfc-editor.org/info/rfc8310>>.
- [RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <<https://www.rfc-editor.org/info/rfc8446>>.

- [RFC8490] Bellis, R., Cheshire, S., Dickinson, J., Dickinson, S., Lemon, T., and T. Pusateri, "DNS Stateful Operations", RFC 8490, DOI 10.17487/RFC8490, March 2019, <<https://www.rfc-editor.org/info/rfc8490>>.
- [SRVTYPE] "Service Name and Transport Protocol Port Number Registry", <<http://www.iana.org/assignments/service-names-port-numbers/>>.

## 10.2. Informative References

- [BCP195] Sheffer, Y., Holz, R., and P. Saint-Andre, "Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", BCP 195, RFC 7525, May 2015, <<http://www.rfc-editor.org/info/bcp195>>.
- [DisProx] Cheshire, S., "Discovery Proxy for Multicast DNS-Based Service Discovery", draft-ietf-dnssd-hybrid-10 (work in progress), March 2019.
- [I-D.ietf-tcpm-rack] Cheng, Y., Cardwell, N., Dukkupati, N., and P. Jha, "RACK: a time-based fast loss detection algorithm for TCP", draft-ietf-tcpm-rack-05 (work in progress), April 2019.
- [LLQ] Cheshire, S. and M. Krochmal, "DNS Long-Lived Queries", draft-sekar-dns-llq-03 (work in progress), March 2019.
- [obs] "Observer Pattern", <[https://en.wikipedia.org/wiki/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>>.
- [RFC3123] Koch, P., "A DNS RR Type for Lists of Address Prefixes (APL RR)", RFC 3123, DOI 10.17487/RFC3123, June 2001, <<https://www.rfc-editor.org/info/rfc3123>>.
- [RFC4287] Nottingham, M., Ed. and R. Sayre, Ed., "The Atom Syndication Format", RFC 4287, DOI 10.17487/RFC4287, December 2005, <<https://www.rfc-editor.org/info/rfc4287>>.
- [RFC4953] Touch, J., "Defending TCP Against Spoofing Attacks", RFC 4953, DOI 10.17487/RFC4953, July 2007, <<https://www.rfc-editor.org/info/rfc4953>>.

- [RFC6281] Cheshire, S., Zhu, Z., Wakikawa, R., and L. Zhang, "Understanding Apple's Back to My Mac (BTMM) Service", RFC 6281, DOI 10.17487/RFC6281, June 2011, <<https://www.rfc-editor.org/info/rfc6281>>.
- [RFC6762] Cheshire, S. and M. Krochmal, "Multicast DNS", RFC 6762, DOI 10.17487/RFC6762, February 2013, <<https://www.rfc-editor.org/info/rfc6762>>.
- [RFC6763] Cheshire, S. and M. Krochmal, "DNS-Based Service Discovery", RFC 6763, DOI 10.17487/RFC6763, February 2013, <<https://www.rfc-editor.org/info/rfc6763>>.
- [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>>.
- [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>>.
- [RFC8010] Sweet, M. and I. McDonald, "Internet Printing Protocol/1.1: Encoding and Transport", STD 92, RFC 8010, DOI 10.17487/RFC8010, January 2017, <<https://www.rfc-editor.org/info/rfc8010>>.
- [RFC8011] Sweet, M. and I. McDonald, "Internet Printing Protocol/1.1: Model and Semantics", STD 92, RFC 8011, DOI 10.17487/RFC8011, January 2017, <<https://www.rfc-editor.org/info/rfc8011>>.
- [RFC8499] Hoffman, P., Sullivan, A., and K. Fujiwara, "DNS Terminology", BCP 219, RFC 8499, DOI 10.17487/RFC8499, January 2019, <<https://www.rfc-editor.org/info/rfc8499>>.

- [SD-API] "dns\_sd.h API",  
<[https://opensource.apple.com/source/mDNSResponder/mDNSResponder-878.70.2/mDNSShared/dns\\_sd.h.auto.html](https://opensource.apple.com/source/mDNSResponder/mDNSResponder-878.70.2/mDNSShared/dns_sd.h.auto.html)>.
- [SYN] Eddy, W., "Defenses Against TCP SYN Flooding Attacks", The Internet Protocol Journal, Cisco Systems, Volume 9, Number 4, December 2006.
- [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](mailto:pusateri@bangj.com)

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

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

Internet Engineering Task Force  
Internet-Draft  
Intended status: Standards Track  
Expires: 26 October 2022

T. Lemon  
S. Cheshire  
Apple Inc.  
24 April 2022

Service Registration Protocol for DNS-Based Service Discovery  
draft-ietf-dnssd-srp-13

## Abstract

The Service Registration Protocol for DNS-Based Service Discovery uses the standard DNS Update mechanism to enable DNS-Based Service Discovery using only unicast packets. This makes it possible to deploy DNS Service Discovery without multicast, which greatly improves scalability and improves performance on networks where multicast service is not an optimal choice, particularly 802.11 (Wi-Fi) and 802.15.4 (IoT) networks. DNS-SD Service registration uses public keys and SIG(0) to allow services to defend their registrations against attack.

## Status of This Memo

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

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

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

This Internet-Draft will expire on 26 October 2022.

## Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components



extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

## Table of Contents

1. Introduction . . . . .	3
2. Service Registration Protocol . . . . .	5
2.1. Protocol Variants . . . . .	5
2.1.1. Full-featured Hosts . . . . .	5
2.1.2. Constrained Hosts . . . . .	6
2.1.3. Why two variants? . . . . .	6
2.2. Protocol Details . . . . .	7
2.2.1. What to publish . . . . .	7
2.2.2. Where to publish it . . . . .	7
2.2.3. How to publish it . . . . .	8
2.2.3.1. How DNS-SD Service Registration differs from standard RFC2136 DNS Update . . . . .	8
2.2.4. How to secure it . . . . .	9
2.2.4.1. First-Come First-Served Naming . . . . .	9
2.2.5. Service Behavior . . . . .	9
2.2.5.1. Public/Private key pair generation and storage . . . . .	9
2.2.5.2. Name Conflict Handling . . . . .	10
2.2.5.3. Record Lifetimes . . . . .	10
2.2.5.4. Compression in SRV records . . . . .	11
2.2.5.5. Removing published services . . . . .	11
2.3. Validation and Processing of SRP Updates . . . . .	12
2.3.1. Validation of Adds and Deletes . . . . .	12
2.3.1.1. Service Discovery Instruction . . . . .	13
2.3.1.2. Service Description Instruction . . . . .	13
2.3.1.3. Host Description Instruction . . . . .	14
2.3.2. Valid SRP Update Requirements . . . . .	14
2.3.3. FCFS Name And Signature Validation . . . . .	15
2.3.4. Handling of Service Subtypes . . . . .	16
2.3.5. SRP Update response . . . . .	16
2.3.6. Optional Behavior . . . . .	16
3. TTL Consistency . . . . .	17
4. Maintenance . . . . .	18
4.1. Cleaning up stale data . . . . .	18
5. Security Considerations . . . . .	19
5.1. Source Validation . . . . .	19
5.2. SRP Server Authentication . . . . .	20
5.3. Required Signature Algorithm . . . . .	20
6. Privacy Considerations . . . . .	21
7. Delegation of 'service.arpa.' . . . . .	21
8. IANA Considerations . . . . .	21
8.1. Registration and Delegation of 'service.arpa' as a Special-Use Domain Name . . . . .	21

8.2. 'dnssd-srp' Service Name . . . . .	21
8.3. 'dnssd-srp-tls' Service Name . . . . .	22
8.4. Anycast Address . . . . .	22
9. Implementation Status . . . . .	23
10. Acknowledgments . . . . .	24
11. Normative References . . . . .	24
12. Informative References . . . . .	26
Appendix A. Testing using standard RFC2136-compliant servers . .	27
Appendix B. How to allow services to update standard RFC2136-compliant servers . . . . .	28
Appendix C. Sample BIND9 configuration for default.service.arpa. . . . .	28
Authors' Addresses . . . . .	29

## 1. Introduction

DNS-Based Service Discovery [RFC6763] is a component of Zero Configuration Networking [RFC6760] [ZC] [I-D.cheshire-dnssd-roadmap].

This document describes an enhancement to DNS-Based Service Discovery [RFC6763] that allows services to register their services using the DNS protocol rather than using Multicast DNS [RFC6762] (mDNS). There is already a large installed base of DNS-SD clients that can discover services using the DNS protocol.

This document is intended for three audiences: implementors of software that provides services that should be advertised using DNS-SD, implementors of DNS servers that will be used in contexts where DNS-SD registration is needed, and administrators of networks where DNS-SD service is required. The document is intended to provide sufficient information to allow interoperable implementation of the registration protocol.

DNS-Based Service Discovery (DNS-SD) allows services to advertise the fact that they provide service, and to provide the information required to access that service. DNS-SD clients can then discover the set of services of a particular type that are available. They can then select a service from among those that are available and obtain the information required to use it. Although DNS-SD using the DNS protocol (as opposed to mDNS) can be more efficient and versatile, it is not common in practice, because of the difficulties associated with updating authoritative DNS services with service information.

Existing practice for updating DNS zones is to either manually enter new data, or else use DNS Update [RFC2136]. Unfortunately DNS Update requires either that the authoritative DNS server automatically trust updates, or else that the DNS Update client have some kind of shared

secret or public key that is known to the DNS server and can be used to authenticate the update. Furthermore, DNS Update can be a fairly chatty process, requiring multiple round trips with different conditional predicates to complete the update process.

The SRP protocol adds a set of default heuristics for processing DNS updates that eliminates the need for DNS update conditional predicates: instead, the SRP server has a set of default predicates that are applied to the update, and the update either succeeds entirely, or fails in a way that allows the registering service to know what went wrong and construct a new update.

SRP also adds a feature called First-Come, First-Served Naming, which allows the registering service to claim a name that is not yet in use, and, using SIG(0) [RFC2931], to authenticate both the initial claim and subsequent updates. This prevents name conflicts, since a second SRP service attempting to claim the same name will not possess the SIG(0) key used by the first service to claim it, and so its claim will be rejected and the second service will have to choose a new name.

Finally, SRP adds the concept of a 'lease,' similar to leases in Dynamic Host Configuration Protocol [RFC8415]. The SRP registration itself has a lease which may be on the order of an hour; if the registering service does not renew the lease before it has elapsed, the registration is removed. The claim on the name can have a longer lease, so that another service cannot claim the name, even though the registration has expired.

The Service Registration Protocol for DNS-SD (SRP), described in this document, provides a reasonably secure mechanism for publishing this information. Once published, these services can be readily discovered by DNS-SD clients using standard DNS lookups.

The DNS-SD specification [RFC6763], Section 10 ("Populating the DNS with Information"), briefly discusses ways that services can publish their information in the DNS namespace. In the case of mDNS, it allows services to publish their information on the local link, using names in the ".local" namespace, which makes their services directly discoverable by peers attached to that same local link.

RFC6763 also allows clients to discover services using the DNS protocol [RFC1035]. This can be done by having a system administrator manually configure service information in the DNS, but manually populating DNS authoritative server databases is costly and potentially error-prone, and requires a knowledgeable network administrator. Consequently, although all DNS-SD client implementations of which we are aware support DNS-SD using DNS queries, in practice it is used much less frequently than mDNS.

The Discovery Proxy [RFC8766] provides one way to automatically populate the DNS namespace, but is only appropriate on networks where services are easily advertised using mDNS. This document describes a solution more suitable for networks where multicast is inefficient, or where sleepy devices are common, by supporting both offering of services, and discovery of services, using unicast.

## 2. Service Registration Protocol

Services that implement SRP use DNS Update [RFC2136] [RFC3007] to publish service information in the DNS. Two variants exist, one for full-featured hosts, and one for devices designed for "Constrained-Node Networks" [RFC7228]. An SRP server is most likely an authoritative DNS server, or else is updating an authoritative DNS server. There is no requirement that the server that is receiving SRP requests be the same server that is answering queries that return records that have been registered.

### 2.1. Protocol Variants

#### 2.1.1. Full-featured Hosts

Full-featured hosts are either configured manually with a registration domain, or use the "dr.\_dns-sd.\_udp.<domain>" query ([RFC6763], Section 11) to learn the default registration domain from the network. RFC6763 says to discover the registration domain using either ".local" or a network-supplied domain name for <domain>. Services using SRP MUST use the domain name received through the DHCPv4 Domain Name option ([RFC2132], Section 3.17), if available, or the Neighbor Discovery DNS Search List option [RFC8106]. If the DNS Search List option contains more than one domain name, it MUST NOT be used. If neither option is available, the Service Registration protocol is not available on the local network.

Manual configuration of the registration domain can be done either by querying the list of available registration zones ("r.\_dns-sd.\_udp") and allowing the user to select one from the UI, or by any other means appropriate to the particular use case being addressed. Full-featured devices construct the names of the SRV, TXT, and PTR records

describing their service(s) as subdomains of the chosen service registration domain. For these names they then discover the zone apex of the closest enclosing DNS zone using SOA queries [RFC8765]. Having discovered the enclosing DNS zone, they query for the "\_dnssd-srp.\_tcp.<zone>" SRV record to discover the server to which they should send DNS updates. Hosts that support SRP Updates using TLS use the "\_dnssd-srp-tls.\_tcp.<zone>" SRV record instead.

#### 2.1.2. Constrained Hosts

For devices designed for Constrained-Node Networks [RFC7228] some simplifications are available. Instead of being configured with (or discovering) the service registration domain, the (proposed) special-use domain name (see [RFC6761]) "default.service.arpa" is used. The details of how SRP server(s) are discovered will be specific to the constrained network, and therefore we do not suggest a specific mechanism here.

SRP clients on constrained networks are expected to receive from the network a list of SRP servers with which to register. It is the responsibility of a Constrained-Node Network supporting SRP to provide one or more SRP server addresses. It is the responsibility of the SRP server supporting a Constrained-Node Network to handle the updates appropriately. In some network environments, updates may be accepted directly into a local "default.service.arpa" zone, which has only local visibility. In other network environments, updates for names ending in "default.service.arpa" may be rewritten internally to names with broader visibility.

#### 2.1.3. Why two variants?

The reason for these different assumptions is that low-power devices that typically use Constrained-Node Networks may have very limited battery power. The series of DNS lookups required to discover an SRP server and then communicate with it will increase the power required to advertise a service; for low-power devices, the additional flexibility this provides does not justify the additional use of power. It is also fairly typical of such networks that some network service information is obtained as part of the process of joining the network, and so this can be relied upon to provide nodes with the information they need.

Networks that are not constrained networks can have more complicated topologies at the Internet layer. Nodes connected to such networks can be assumed to be able to do DNSSD service registration domain discovery. Such networks are generally able to provide registration domain discovery and routing. By requiring the use of TCP, the possibility of off-network spoofing is eliminated.

## 2.2. Protocol Details

We will discuss several parts to this process: how to know what to publish, how to know where to publish it (under what name), how to publish it, how to secure its publication, and how to maintain the information once published.

### 2.2.1. What to publish

We refer to the DNS Update message sent by services using SRP as an SRP Update. Three types of updates appear in an SRP update: Service Discovery records, Service Description records, and Host Description records.

- \* Service Discovery records are one or more PTR RRs, mapping from the generic service type (or subtype) to the specific Service Instance Name.
- \* Service Description records are exactly one SRV RR, exactly one KEY RR, and one or more TXT RRs, all with the same name, the Service Instance Name ([RFC6763], Section 4.1). In principle Service Description records can include other record types, with the same Service Instance Name, though in practice they rarely do. The Service Instance Name MUST be referenced by one or more Service Discovery PTR records, unless it is a placeholder service registration for an intentionally non-discoverable service name.
- \* The Host Description records for a service are a KEY RR, used to claim exclusive ownership of the service registration, and one or more RRs of type A or AAAA, giving the IPv4 or IPv6 address(es) of the host where the service resides.

[RFC6763] describes the details of what each of these types of updates contains, with the exception of the KEY RR, which is defined in [RFC2539]. These RFCs should be considered the definitive source for information about what to publish; the reason for summarizing this here is to provide the reader with enough information about what will be published that the service registration process can be understood at a high level without first learning the full details of DNS-SD. Also, the "Service Instance Name" is an important aspect of first-come, first-serve naming, which we describe later on in this document.

### 2.2.2. Where to publish it

Multicast DNS uses a single namespace, ".local", which is valid on the local link. This convenience is not available for DNS-SD using the DNS protocol: services must exist in some specific unicast namespace.

As described above, full-featured devices are responsible for knowing in what domain they should register their services. Devices made for Constrained-Node Networks register in the (proposed) special use domain name [RFC6761] "default.service.arpa", and let the SRP server handle rewriting that to a different domain if necessary.

### 2.2.3. How to publish it

It is possible to issue a DNS Update that does several things at once; this means that it's possible to do all the work of adding a PTR resource record to the PTR RRset on the Service Name, and creating or updating the Service Instance Name and Host Description, in a single transaction.

An SRP Update takes advantage of this: it is implemented as a single DNS Update message that contains a service's Service Discovery records, Service Description records, and Host Description records.

Updates done according to this specification are somewhat different than regular DNS Updates as defined in RFC2136. The RFC2136 update process can involve many update attempts: you might first attempt to add a name if it doesn't exist; if that fails, then in a second message you might update the name if it does exist but matches certain preconditions. Because the registration protocol uses a single transaction, some of this adaptability is lost.

In order to allow updates to happen in a single transaction, SRP Updates do not include update prerequisites. The requirements specified in Section 2.3 are implicit in the processing of SRP Updates, and so there is no need for the service sending the SRP Update to put in any explicit prerequisites.

#### 2.2.3.1. How DNS-SD Service Registration differs from standard RFC2136 DNS Update

DNS-SD Service Registration is based on standard RFC2136 DNS Update, with some differences:

- \* It implements first-come first-served name allocation, protected using SIG(0) [RFC2931].
- \* It enforces policy about what updates are allowed.
- \* It optionally performs rewriting of "default.service.arpa" to some other domain.
- \* It optionally performs automatic population of the address-to-name reverse mapping domains.
- \* An SRP server is not required to implement general DNS Update prerequisite processing.

- \* Constrained-Node SRP clients are allowed to send updates to the generic domain "default.service.arpa"

#### 2.2.4. How to secure it

Traditional DNS update is secured using the TSIG protocol, which uses a secret key shared between the DNS Update client (which issues the update) and the server (which authenticates it). This model does not work for automatic service registration.

The goal of securing the DNS-SD Registration Protocol is to provide the best possible security given the constraint that service registration has to be automatic. It is possible to layer more operational security on top of what we describe here, but what we describe here is an improvement over the security of mDNS. The goal is not to provide the level of security of a network managed by a skilled operator.

##### 2.2.4.1. First-Come First-Served Naming

First-Come First-Serve naming provides a limited degree of security: a service that registers its service using DNS-SD Registration protocol is given ownership of a name for an extended period of time based on the key used to authenticate the DNS Update. As long as the registration service remembers the name and the key used to register that name, no other service can add or update the information associated with that. FCFS naming is used to protect both the Service Description and the Host Description.

#### 2.2.5. Service Behavior

##### 2.2.5.1. Public/Private key pair generation and storage

The service generates a public/private key pair. This key pair **MUST** be stored in stable storage; if there is no writable stable storage on the SRP client, the SRP client **MUST** be pre-configured with a public/private key pair in read-only storage that can be used. This key pair **MUST** be unique to the device. A device with rewritable storage should retain this key indefinitely. When the device changes ownership, it may be appropriate to erase the old key and install a new one. Therefore, the SRP client on the device **SHOULD** provide a mechanism to overwrite the key, for example as the result of a "factory reset."

When sending DNS updates, the service includes a KEY record containing the public portion of the key in each Host Description Instruction and each Service Description Instruction. Each KEY record **MUST** contain the same public key. The update is signed using



SIG(0), using the private key that corresponds to the public key in the KEY record. The lifetimes of the records in the update is set using the EDNS(0) Update Lease option [I-D.sekar-dns-ul].

The KEY record in Service Description updates MAY be omitted for brevity; if it is omitted, the SRP server MUST behave as if the same KEY record that is given for the Host Description is also given for each Service Description for which no KEY record is provided. Omitted KEY records are not used when computing the SIG(0) signature.

#### 2.2.5.2. Name Conflict Handling

Both Host Description records and Service Description Records can have names that result in name conflicts. Service Discovery records cannot have name conflicts. If any Host Description or Service Description record is found by the server to have a conflict with an existing name, the server will respond to the SRP Update with a YXDOMAIN rcode. In this case, the Service MUST either abandon the service registration attempt, or else choose a new name.

There is no specific requirement for how this is done; typically, however, the service will append a number to the preferred name. This number could be sequentially increasing, or could be chosen randomly. One existing implementation attempts several sequential numbers before choosing randomly. So for instance, it might try host.service.arpa, then host-1.service.arpa, then host-2.service.arpa, then host-31773.service.arpa.

#### 2.2.5.3. Record Lifetimes

The lifetime of the DNS-SD PTR, SRV, A, AAAA and TXT records [RFC6763] uses the LEASE field of the Update Lease option, and is typically set to two hours. This means that if a device is disconnected from the network, it does not appear in the user interfaces of devices looking for services of that type for too long.

The lifetime of the KEY records is set using the KEY-LEASE field of the Update Lease Option, and should be set to a much longer time, typically 14 days. The result of this is that even though a device may be temporarily unplugged, disappearing from the network for a few days, it makes a claim on its name that lasts much longer.

This means that even if a device is unplugged from the network for a few days, and its services are not available for that time, no other device can come along and claim its name the moment it disappears from the network. In the event that a device is unplugged from the network and permanently discarded, then its name is eventually cleaned up and made available for re-use.

#### 2.2.5.4. Compression in SRV records

Although [RFC2782] requires that the target name in the SRV record not be compressed, an SRP client SHOULD compress the target in the SRV record. The motivation for not compressing in RFC2782 is not stated, but is assumed to be because a caching resolver that does not understand the format of the SRV record might store it as binary data and thus return an invalid pointer in response to a query. This does not apply in the case of SRP: an SRP server needs to understand SRV records in order to validate the SRP Update. Compression of the target potentially saves substantial space in the SRP Update.

#### 2.2.5.5. Removing published services

##### 2.2.5.5.1. Removing all published services

To remove all the services registered to a particular host, the SRP client retransmits its most recent update with an Update Lease option that has a LEASE value of zero. If the registration is to be permanently removed, KEY-LEASE should also be zero. Otherwise, it should have the same value it had previously; this holds the name in reserve for when the SRP client is once again able to provide the service.

SRP clients are normally expected to remove all service instances when removing a host. However, in some cases a SRP client may not have retained sufficient state to know that some service instance is pointing to a host that it is removing. This method of removing services is intended for the case where the client is going offline and does not want its services advertised. Therefore, it is sufficient for the client to send the Host Description Instruction (Section 2.3.1.3).

To support this, when removing services based on the lease time being zero, an SRP server MUST remove all service instances pointing to a host when a host is removed, even if the SRP client doesn't list them explicitly. If the key lease time is nonzero, the SRP server MUST NOT delete the KEY records for these SRP clients.

##### 2.2.5.5.2. Removing some published services

In some use cases a client may need to remove some specific service, without removing its other services. This can be accomplished in one of two ways. To simply remove a specific service, the client sends a valid SRP Update where the Service Discovery Instruction (Section 2.3.1.1) contains a single Delete an RR from an RRset ([RFC2136], Section 2.5.4) update that deletes the PTR record whose target is the service instance name. The Service Description

Instruction (Section 2.3.1.2) in this case contains a single Delete all RRsets from a Name ([RFC2136], Section 2.5.3) update to the service instance name.

The second alternative is used when some service is being replaced by a different service with a different service instance name. In this case, the old service is deleted as in the first alternative. The new service is added, just as it would be in an update that wasn't deleting the old service. Because both the removal of the old service and the add of the new service consist of a valid Service Discovery Instruction and a valid Service Description Instruction, the update as a whole is a valid SRP Update, and will result in the old service being removed and the new one added, or, to put it differently, in the old service being replaced by the new service.

It is perhaps worth noting that if a service is being updated without the service instance name changing, that will look very much like the second alternative above. The difference is that because the target for the PTR record in the Service Discovery Instruction is the same for both the Delete An RR From An RRset update and the Add To An RRset update, these will be seen as a single Service Description Instruction, not as two Instructions. The same would be true of the Service Description Instruction.

Whichever of these two alternatives is used, the host lease will be updated with the lease time provided in the SRP update. In neither of these cases is it permissible to delete the host. All services must point to a host. If a host is to be deleted, this must be done using the method described in Section 2.2.5.5.1, which deletes the host and all services that have that host as their target.

## 2.3. Validation and Processing of SRP Updates

### 2.3.1. Validation of Adds and Deletes

The SRP server first validates that the DNS Update is a syntactically and semantically valid DNS Update according to the rules specified in RFC2136.

SRP Updates consist of a set of `_instructions_` that together add or remove one or more services. Each instruction consists of some combination of delete updates and add updates. When an instruction contains a delete and an add, the delete **MUST** precede the add.

The SRP server checks each instruction in the SRP Update to see that it is either a Service Discovery Instruction, a Service Description Instruction, or a Host Description Instruction. Order matters in DNS updates. Specifically, deletes must precede adds for records that

the deletes would affect; otherwise the add will have no effect. This is the only ordering constraint; aside from this constraint, updates may appear in whatever order is convenient when constructing the update.

Because the SRP Update is a DNS update, it MUST contain a single question that indicates the zone to be updated. Every delete and update in an SRP Update MUST be within the zone that is specified for the SRP Update.

#### 2.3.1.1. Service Discovery Instruction

An instruction is a Service Discovery Instruction if it contains

- \* exactly one "Add to an RRSet" or exactly one "Delete an RR from an RRSet" ([RFC2136], Section 2.5.1) RR update,
- \* which updates a PTR RR,
- \* the target of which is a Service Instance Name
- \* for which name a Service Description Instruction is present in the SRP Update
- \* if the Service Discovery Instruction is an "Add to an RRSet" instruction, the Service Description Instruction does not match if it does not contain an "Add to an RRset" update for the SRV RR describing that service.
- \* if the Service Discovery Instruction is a "Delete an RR from an RRSet" update, the Service Description Instruction does not match if it contains an "Add to an RRset" update.
- \* Service Discovery Instructions do not contain any other add or delete updates.

#### 2.3.1.2. Service Description Instruction

An instruction is a Service Description Instruction if, for the appropriate Service Instance Name, it contains

- \* exactly one "Delete all RRsets from a name" update for the service instance name ([RFC2136], Section 2.5.3),
- \* zero or one "Add to an RRset" SRV RR,
- \* zero or one "Add to an RRset" KEY RR that, if present, contains the public key corresponding to the private key that was used to sign the message (if present, the KEY MUST match the KEY RR given in the Host Description),
- \* zero or more "Add to an RRset" TXT RRs,
- \* If there is one "Add to an RRset" SRV update, there MUST be at least one "Add to an RRset" TXT update.
- \* the target of the SRV RR Add, if present points to a hostname for which there is a Host Description Instruction in the SRP Update, or

- \* if there is no "Add to an RRset" SRV RR, then either
  - the name to which the "Delete all RRsets from a name" applies does not exist, or
  - there is an existing KEY RR on that name, which matches the key with which the SRP Update was signed.
- \* Service Descriptions Instructions do not modify any other resource records.

An SRP server MUST correctly handle compressed names in the SRV target.

#### 2.3.1.3. Host Description Instruction

An instruction is a Host Description Instruction if, for the appropriate hostname, it contains

- \* exactly one "Delete all RRsets from a name" RR,
- \* one or more "Add to an RRset" RRs of type A and/or AAAA,
- \* A and/or AAAA records must be of sufficient scope to be reachable by all hosts that might query the DNS. If a link-scope address or IPv4 autoconfiguration address is provided by the SRP client, the SRP server MUST treat this as if no address records were received; that is, the Host Description is not valid.
- \* exactly one "Add to an RRset" RR that adds a KEY RR that contains the public key corresponding to the private key that was used to sign the message,
- \* there is a Service Instance Name Instruction in the SRP Update for which the SRV RR that is added points to the hostname being updated by this update.
- \* Host Description Instructions do not modify any other resource records.

#### 2.3.2. Valid SRP Update Requirements

An SRP Update MUST include zero or more Service Discovery Instructions. For each Service Discovery Instruction, there MUST be at least one Service Description Instruction. Note that in the case of SRP subtypes (Section 7.1 of [RFC6763]), it's quite possible that two Service Discovery Instructions might reference the same Service Description Instruction. For each Service Description Instruction there MUST be at least one Service Discovery Instruction with its service instance name as the target of its PTR record. There MUST be exactly one Host Description Instruction. Every Service Description Instruction must have that Host Description Instruction as the target of its SRV record. A DNS Update that does not meet these constraints is not an SRP Update.

A DNS Update that contains any additional adds or deletes that cannot be identified as Service Discovery, Service Description or Host Description Instructions is not an SRP Update. A DNS update that contains any prerequisites is not an SRP Update. Such messages should either be processed as regular RFC2136 updates, including access control checks and constraint checks, if supported, or else rejected with RCODE=REFUSED.

In addition, in order for an update to be a valid SRP Update, the target of every Service Discovery Instruction MUST be a Service Description Instruction that is present in the SRP Update. There MUST NOT be any Service Description Instruction to which no Service Discovery Instruction points. The target of the SRV record in every Service Description Instruction MUST be the single Host Description Instruction.

If the definitions of each of these instructions are followed carefully and the update requirements are validated correctly, many DNS Updates that look very much like SRP Updates nevertheless will fail to validate. For example, a DNS update that contains an Add to an RRset instruction for a Service Name and an Add to an RRset instruction for a Service Instance Name, where the PTR record added to the Service Name does not reference the Service Instance Name, is not a valid SRP Update message, but may be a valid RFC2136 update.

### 2.3.3. FCFS Name And Signature Validation

Assuming that a DNS Update message has been validated with these conditions and is a valid SRP Update, the server checks that the name in the Host Description Instruction exists. If so, then the server checks to see if the KEY record on that name is the same as the KEY record in the Host Description Instruction. The server performs the same check for the KEY records in any Service Description Instructions. For KEY records that were omitted from Service Description Instructions, the KEY from the Host Description Instruction is used. If any existing KEY record corresponding to a KEY record in the SRP Update does not match the KEY record in the SRP Update (whether provided or taken from the Host Description Instruction), then the server MUST reject the SRP Update with the YXDOMAIN RCODE.

Otherwise, the server validates the SRP Update using SIG(0) against the public key in the KEY record of the Host Description Instruction. If the validation fails, the server MUST reject the SRP Update with the REFUSED RCODE. Otherwise, the SRP Update is considered valid and authentic, and is processed according to the method described in RFC2136.

KEY record updates omitted from Service Description Instruction are processed as if they had been explicitly present: every Service Description that is updated MUST, after the SRP Update has been applied, have a KEY RR, and it must be the same KEY RR that is present in the Host Description to which the Service Description refers.

#### 2.3.4. Handling of Service Subtypes

SRP servers MUST treat the update instructions for a service type and all its subtypes as atomic. That is, when a service and its subtypes are being updated, whatever information appears in the SRP Update is the entirety of information about that service and its subtypes. If any subtype appeared in a previous update but does not appear in the current update, then the DNS server MUST remove that subtype.

Similarly, there is no mechanism for deleting subtypes. A delete of a service deletes all of its subtypes. To delete an individual subtype, an SRP Update must be constructed that contains the service type and all subtypes for that service.

#### 2.3.5. SRP Update response

The status that is returned depends on the result of processing the update, and can be either SUCCESS or SERVFAIL: all other possible outcomes should already have been accounted for when applying the constraints that qualify the update as an SRP Update.

#### 2.3.6. Optional Behavior

The server MAY add a Reverse Mapping that corresponds to the Host Description. This is not required because the Reverse Mapping serves no protocol function, but it may be useful for debugging, e.g. in annotating network packet traces or logs. In order for the server to add a reverse mapping update, it must be authoritative for the zone or have credentials to do the update. The SRP client MAY also do a reverse mapping update if it has credentials to do so.

The server MAY apply additional criteria when accepting updates. In some networks, it may be possible to do out-of-band registration of keys, and only accept updates from pre-registered keys. In this case, an update for a key that has not been registered should be rejected with the REFUSED RCODE.

There are at least two benefits to doing this rather than simply using normal SIG(0) DNS updates. First, the same registration protocol can be used in both cases, so both use cases can be addressed by the same service implementation. Second, the registration protocol includes maintenance functionality not present with normal DNS updates.

Note that the semantics of using SRP in this way are different than for typical RFC2136 implementations: the KEY used to sign the SRP Update only allows the SRP client to update records that refer to its Host Description. RFC2136 implementations do not normally provide a way to enforce a constraint of this type.

The server may also have a dictionary of names or name patterns that are not permitted. If such a list is used, updates for Service Instance Names that match entries in the dictionary are rejected with YXDOMAIN.

### 3. TTL Consistency

All RRs within an RRset are required to have the same TTL (Clarifications to the DNS Specification [RFC2181], Section 5.2). In order to avoid inconsistencies, SRP places restrictions on TTLs sent by services and requires that SRP servers enforce consistency.

Services sending SRP Updates MUST use consistent TTLs in all RRs within the SRP Update.

SRP servers MUST check that the TTLs for all RRs within the SRP Update are the same. If they are not, the SRP update MUST be rejected with a REFUSED RCODE.

Additionally, when adding RRs to an RRset, for example when processing Service Discovery records, the server MUST use the same TTL on all RRs in the RRset. How this consistency is enforced is up to the implementation.

TTLs sent in SRP Updates are advisory: they indicate the SRP client's guess as to what a good TTL would be. SRP servers may override these TTLs. SRP servers SHOULD ensure that TTLs are reasonable: neither too long nor too short. The TTL should never be longer than the lease time (Section 4.1). Shorter TTLs will result in more frequent data refreshes; this increases latency on the DNS-SD client side, increases load on any caching resolvers and on the authoritative server, and also increases network load, which may be an issue for constrained networks. Longer TTLs will increase the likelihood that data in caches will be stale. TTL minimums and maximums SHOULD be configurable by the operator of the SRP server.



## 4. Maintenance

### 4.1. Cleaning up stale data

Because the DNS-SD registration protocol is automatic, and not managed by humans, some additional bookkeeping is required. When an update is constructed by the SRP client, it **MUST** include an EDNS(0) Update Lease Option [I-D.sekar-dns-ul]. The Update Lease Option contains two lease times: the Lease Time and the Key Lease Time.

These leases are promises, similar to DHCP leases [RFC2131], from the SRP client that it will send a new update for the service registration before the lease time expires. The Lease time is chosen to represent the time after the update during which the registered records other than the KEY record should be assumed to be valid. The Key Lease time represents the time after the update during which the KEY record should be assumed to be valid.

The reasoning behind the different lease times is discussed in the section on first-come, first-served naming (Section 2.2.4.1). SRP servers may be configured with limits for these values. A default limit of two hours for the Lease and 14 days for the SIG(0) KEY are currently thought to be good choices. Constrained devices with limited battery that wake infrequently are likely to request longer leases; servers that support such devices may need to set higher limits. SRP clients that are going to continue to use names on which they hold leases should update well before the lease ends, in case the registration service is unavailable or under heavy load.

The lease time applies specifically to the host. All service instances, and all service entries for such service instances, depend on the host. When the lease on a host expires, the host and all services that reference it **MUST** be removed at the same time—it is never valid for a service instance to remain when the host it references has been removed. If the KEY record for the host is to remain, the KEY record for any services that reference it **MUST** also remain. However, the service PTR record **MUST** be removed, since it has no key associated with it, and since it is never valid to have a service PTR record for which there is no service instance on the target of the PTR record.

SRP Servers **SHOULD** also track a lease time per service instance. The reason for doing this is that a client may re-register a host with a different set of services, and not remember that some different service instance had previously been registered. In this case, when that service instance lease expires, the SRP server **SHOULD** remove the service instance (although the KEY record for the service instance **SHOULD** be retained until the key lease on that service expires).

This is beneficial because if the SRP client continues to renew the host, but never mentions the stale service again, the stale service will continue to be advertised.

The SRP server MUST include an EDNS(0) Update Lease option in the response if the lease time proposed by the service has been shortened or lengthened. The service MUST check for the EDNS(0) Update Lease option in the response and MUST use the lease times from that option in place of the options that it sent to the server when deciding when to update its registration. The times may be shorter or longer than those specified in the SRP Update; the SRP client must honor them in either case.

SRP clients should assume that each lease ends N seconds after the update was first transmitted, where N is the lease duration. Servers should assume that each lease ends N seconds after the update that was successfully processed was received. Because the server will always receive the update after the SRP client sent it, this avoids the possibility of misunderstandings.

SRP servers MUST reject updates that do not include an EDNS(0) Update Lease option. Dual-use servers MAY accept updates that don't include leases, but SHOULD differentiate between SRP Updates and other updates, and MUST reject updates that would otherwise be SRP Updates if they do not include leases.

Lease times have a completely different function than TTLs. On an authoritative DNS server, the TTL on a resource record is a constant: whenever that RR is served in a DNS response, the TTL value sent in the answer is the same. The lease time is never sent as a TTL; its sole purpose is to determine when the authoritative DNS server will delete stale records. It is not an error to send a DNS response with a TTL of 'n' when the remaining time on the lease is less than 'n'.

## 5. Security Considerations

### 5.1. Source Validation

SRP Updates have no authorization semantics other than first-come, first-served. This means that if an attacker from outside of the administrative domain of the server knows the server's IP address, it can in principle send updates to the server that will be processed successfully. Servers should therefore be configured to reject updates from source addresses outside of the administrative domain of the server.

For updates sent to an anycast IP address of an SRP server, this validation must be enforced by every router on the path from the Constrained-Device Network to the unconstrained portion of the network. For TCP updates, the initial SYN-SYN+ACK handshake prevents updates being forged by an off-network attacker. In order to ensure that this handshake happens, SRP servers relying on three-way-handshake validation **MUST NOT** accept TCP Fast Open payloads. If the network infrastructure allows it, an SRP server **MAY** accept TCP Fast Open payloads if all such packets are validated along the path, and the network is able to reject this type of spoofing at all ingress points.

Note that these rules only apply to the validation of SRP Updates. A server that accepts updates from SRP clients may also accept other DNS updates, and those DNS updates may be validated using different rules. However, in the case of a DNS service that accepts SRP updates, the intersection of the SRP Update rules and whatever other update rules are present must be considered very carefully.

For example, a normal, authenticated DNS update to any RR that was added using SRP, but that is authenticated using a different key, could be used to override a promise made by the SRP Server to an SRP client, by replacing all or part of the service registration information with information provided by an authenticated DNS update client. An implementation that allows both kinds of updates should not allow DNS Update clients that are using different authentication and authorization credentials to update records added by SRP clients.

## 5.2. SRP Server Authentication

This specification does not provide a mechanism for validating responses from DNS servers to SRP clients. In the case of Constrained Network/Constrained Node clients, such validation isn't practical because there's no way to establish trust. In principle, a KEY RR could be used by a non-constrained SRP client to validate responses from the server, but this is not required, nor do we specify a mechanism for determining which key to use.

## 5.3. Required Signature Algorithm

For validation, SRP servers **MUST** implement the ECDSA<sub>P256</sub>SHA256 signature algorithm. SRP servers **SHOULD** implement the algorithms specified in [RFC8624], Section 3.1, in the validation column of the table, that are numbered 13 or higher and have a "MUST", "RECOMMENDED", or "MAY" designation in the validation column of the table. SRP clients **MUST NOT** assume that any algorithm numbered lower than 13 is available for use in validating SIG(0) signatures.

## 6. Privacy Considerations

Because DNSSD SRP Updates can be sent off-link, the privacy implications of SRP are different than for multicast DNS responses. Host implementations that are using TCP SHOULD also use TLS if available. Server implementations MUST offer TLS support. The use of TLS with DNS is described in [RFC7858] and [RFC8310].

Hosts that implement TLS support SHOULD NOT fall back to TCP; since servers are required to support TLS, it is entirely up to the host implementation whether to use it.

Public keys can be used as identifiers to track hosts. SRP servers MAY elect not to return KEY records for queries for SRP registrations.

## 7. Delegation of 'service.arpa.'

In order to be fully functional, the owner of the 'arpa.' zone must add a delegation of 'service.arpa.' in the '.arpa.' zone [RFC3172]. This delegation should be set up as was done for 'home.arpa', as a result of the specification in Section 7 of [RFC8375].

## 8. IANA Considerations

### 8.1. Registration and Delegation of 'service.arpa' as a Special-Use Domain Name

IANA is requested to record the domain name 'service.arpa.' in the Special-Use Domain Names registry [SUDN]. IANA is requested, with the approval of IAB, to implement the delegation requested in Section 7.

IANA is further requested to add a new entry to the "Transport-Independent Locally-Served Zones" subregistry of the the "Locally-Served DNS Zones" registry [LSDZ]. The entry will be for the domain 'service.arpa.' with the description "DNS-SD Registration Protocol Special-Use Domain", listing this document as the reference.

### 8.2. 'dnssd-srp' Service Name

IANA is also requested to add a new entry to the Service Names and Port Numbers registry for dnssd-srp with a transport type of tcp. No port number is to be assigned. The reference should be to this document, and the Assignee and Contact information should reference the authors of this document. The Description should be as follows:

Availability of DNS Service Discovery Service Registration Protocol Service for a given domain is advertised using the "\_dnssd-srp.\_tcp.<domain>" SRV record gives the target host and port where DNSSD Service Registration Service is provided for the named domain.

### 8.3. 'dnssd-srp-tls' Service Name

IANA is also requested to add a new entry to the Service Names and Port Numbers registry for dnssd-srp with a transport type of tcp. No port number is to be assigned. The reference should be to this document, and the Assignee and Contact information should reference the authors of this document. The Description should be as follows:

Availability of DNS Service Discovery Service Registration Protocol Service for a given domain over TLS is advertised using the "\_dnssd-srp-tls.\_tcp.<domain>." SRV record gives the target host and port where DNSSD Service Registration Service is provided for the named domain.

### 8.4. Anycast Address

IANA is requested to allocate an IPv6 Anycast address from the IPv6 Special-Purpose Address Registry, similar to the Port Control Protocol anycast address, 2001:1::1. The value TBD should be replaced with the actual allocation in the table that follows. The values for the registry are:

Attribute	value
Address Block	2001:1::TBD/128
Name	DNS-SD Service Registration Protocol Anycast Address
RFC	[this document]
Allocation Date	[date of allocation]
Termination Date	N/A
Source	True
Destination	True
Forwardable	True
Global	True
Reserved-by-protocol	False

Table 1

## 9. Implementation Status

[Note to the RFC Editor: please remove this section prior to publication.]

This section records the status of known implementations of the protocol defined by this specification at the time of posting of this Internet-Draft, and is based on a proposal described in RFC 7942. The description of implementations in this section is intended to assist the IETF in its decision processes in progressing drafts to RFCs. Please note that the listing of any individual implementation here does not imply endorsement by the IETF. Furthermore, no effort has been spent to verify the information presented here that was supplied by IETF contributors. This is not intended as, and must not be construed to be, a catalog of available implementations or their features. Readers are advised to note that other implementations may exist.

According to RFC 7942, "this will allow reviewers and working groups to assign due consideration to documents that have the benefit of running code, which may serve as evidence of valuable experimentation

and feedback that have made the implemented protocols more mature. It is up to the individual working groups to use this information as they see fit".

There are two known independent implementations of SRP clients:

- \* SRP Client for OpenThread:  
<https://github.com/openthread/openthread/pull/6038>
- \* mDNSResponder open source project: <https://github.com/Abhayakara/mdnsresponder>

There are two related implementations of an SRP server. One acts as a DNS Update proxy, taking an SRP Update and applying it to the specified DNS zone using DNS update. The other acts as an Advertising Proxy [I-D.sctl-advertising-proxy]. Both are included in the mDNSResponder open source project mentioned above.

## 10. Acknowledgments

Thanks to Toke Høiland-Jørgensen, Jonathan Hui, Esko Dijk, Kangping Dong and Abtin Keshavarzian for their thorough technical reviews. Thanks to Kangping and Abtin as well for testing the document by doing an independent implementation. Thanks to Tamara Kemper for doing a nice developmental edit, Tim Wattenberg for doing a SRP client proof-of-concept implementation at the Montreal Hackathon at IETF 102, and Tom Pusateri for reviewing during the hackathon and afterwards.

## 11. Normative References

- [I-D.sekar-dns-ul]  
Cheshire, S. and T. Lemon, "An EDNS0 option to negotiate Leases on DNS Updates", Work in Progress, Internet-Draft, draft-sekar-dns-ul-03, 27 July 2021, <<https://datatracker.ietf.org/doc/html/draft-sekar-dns-ul-03>>.
- [RFC2132] Alexander, S. and R. Droms, "DHCP Options and BOOTP Vendor Extensions", RFC 2132, DOI 10.17487/RFC2132, March 1997, <<https://www.rfc-editor.org/info/rfc2132>>.
- [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>>.

- [RFC2539] Eastlake 3rd, D., "Storage of Diffie-Hellman Keys in the Domain Name System (DNS)", RFC 2539, DOI 10.17487/RFC2539, March 1999, <<https://www.rfc-editor.org/info/rfc2539>>.
- [RFC2931] Eastlake 3rd, D., "DNS Request and Transaction Signatures ( SIG(0)s )", RFC 2931, DOI 10.17487/RFC2931, September 2000, <<https://www.rfc-editor.org/info/rfc2931>>.
- [RFC3172] Huston, G., Ed., "Management Guidelines & Operational Requirements for the Address and Routing Parameter Area Domain ("arpa")", BCP 52, RFC 3172, DOI 10.17487/RFC3172, September 2001, <<https://www.rfc-editor.org/info/rfc3172>>.
- [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>>.
- [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>>.
- [RFC8106] Jeong, J., Park, S., Beloeil, L., and S. Madanapalli, "IPv6 Router Advertisement Options for DNS Configuration", RFC 8106, DOI 10.17487/RFC8106, March 2017, <<https://www.rfc-editor.org/info/rfc8106>>.
- [RFC8375] Pfister, P. and T. Lemon, "Special-Use Domain 'home.arpa.'", RFC 8375, DOI 10.17487/RFC8375, May 2018, <<https://www.rfc-editor.org/info/rfc8375>>.
- [RFC8624] Wouters, P. and O. Sury, "Algorithm Implementation Requirements and Usage Guidance for DNSSEC", RFC 8624, DOI 10.17487/RFC8624, June 2019, <<https://www.rfc-editor.org/info/rfc8624>>.
- [RFC8765] Pusateri, T. and S. Cheshire, "DNS Push Notifications", RFC 8765, DOI 10.17487/RFC8765, June 2020, <<https://www.rfc-editor.org/info/rfc8765>>.
- [SUDN] "Special-Use Domain Names Registry", July 2012, <<https://www.iana.org/assignments/special-use-domain-names/special-use-domain-names.xhtml>>.
- [LSDZ] "Locally-Served DNS Zones Registry", July 2011, <<https://www.iana.org/assignments/locally-served-dns-zones/locally-served-dns-zones.xhtml>>.



## 12. Informative References

- [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>>.
- [RFC2131] Droms, R., "Dynamic Host Configuration Protocol", RFC 2131, DOI 10.17487/RFC2131, March 1997, <<https://www.rfc-editor.org/info/rfc2131>>.
- [RFC2181] Elz, R. and R. Bush, "Clarifications to the DNS Specification", RFC 2181, DOI 10.17487/RFC2181, July 1997, <<https://www.rfc-editor.org/info/rfc2181>>.
- [RFC2782] Gulbrandsen, A., Vixie, P., and L. Esibov, "A DNS RR for specifying the location of services (DNS SRV)", RFC 2782, DOI 10.17487/RFC2782, February 2000, <<https://www.rfc-editor.org/info/rfc2782>>.
- [RFC3007] Wellington, B., "Secure Domain Name System (DNS) Dynamic Update", RFC 3007, DOI 10.17487/RFC3007, November 2000, <<https://www.rfc-editor.org/info/rfc3007>>.
- [RFC6760] Cheshire, S. and M. Krochmal, "Requirements for a Protocol to Replace the AppleTalk Name Binding Protocol (NBP)", RFC 6760, DOI 10.17487/RFC6760, February 2013, <<https://www.rfc-editor.org/info/rfc6760>>.
- [RFC6761] Cheshire, S. and M. Krochmal, "Special-Use Domain Names", RFC 6761, DOI 10.17487/RFC6761, February 2013, <<https://www.rfc-editor.org/info/rfc6761>>.
- [RFC6762] Cheshire, S. and M. Krochmal, "Multicast DNS", RFC 6762, DOI 10.17487/RFC6762, February 2013, <<https://www.rfc-editor.org/info/rfc6762>>.
- [RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <<https://www.rfc-editor.org/info/rfc7228>>.
- [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>>.

- [RFC8415] Mrugalski, T., Siodelski, M., Volz, B., Yourtchenko, A., Richardson, M., Jiang, S., Lemon, T., and T. Winters, "Dynamic Host Configuration Protocol for IPv6 (DHCPv6)", RFC 8415, DOI 10.17487/RFC8415, November 2018, <<https://www.rfc-editor.org/info/rfc8415>>.
- [RFC8766] Cheshire, S., "Discovery Proxy for Multicast DNS-Based Service Discovery", RFC 8766, DOI 10.17487/RFC8766, June 2020, <<https://www.rfc-editor.org/info/rfc8766>>.
- [I-D.cheshire-dnssd-roadmap]  
Cheshire, S., "Service Discovery Road Map", Work in Progress, Internet-Draft, draft-cheshire-dnssd-roadmap-03, 23 October 2018, <<https://datatracker.ietf.org/doc/html/draft-cheshire-dnssd-roadmap-03>>.
- [I-D.cheshire-edns0-owner-option]  
Cheshire, S. and M. Krochmal, "EDNS0 OWNER Option", Work in Progress, Internet-Draft, draft-cheshire-edns0-owner-option-01, 3 July 2017, <<https://datatracker.ietf.org/doc/html/draft-cheshire-edns0-owner-option-01>>.
- [I-D.sctl-advertising-proxy]  
Cheshire, S. and T. Lemon, "Advertising Proxy for DNS-SD Service Registration Protocol", Work in Progress, Internet-Draft, draft-sctl-advertising-proxy-02, 12 July 2021, <<https://datatracker.ietf.org/doc/html/draft-sctl-advertising-proxy-02>>.
- [ZC] Cheshire, S. and D.H. Steinberg, "Zero Configuration Networking: The Definitive Guide", O'Reilly Media, Inc. , ISBN 0-596-10100-7, December 2005.

#### Appendix A. Testing using standard RFC2136-compliant servers

It may be useful to set up a DNS server for testing that does not implement SRP. This can be done by configuring the server to listen on the anycast address, or advertising it in the `_dnssd-srp._tcp.<zone>` SRV and `_dnssd-srp-tls._tcp.<zone>` record. It must be configured to be authoritative for "default.service.arpa", and to accept updates from hosts on local networks for names under "default.service.arpa" without authentication, since such servers will not have support for FCFS authentication (Section 2.2.4.1).

A server configured in this way will be able to successfully accept and process SRP Updates from services that send SRP updates. However, no prerequisites will be applied, and this means that the

test server will accept internally inconsistent SRP Updates, and will not stop two SRP Updates, sent by different services, that claim the same name(s), from overwriting each other.

Since SRP Updates are signed with keys, validation of the SIG(0) algorithm used by the client can be done by manually installing the client public key on the DNS server that will be receiving the updates. The key can then be used to authenticate the client, and can be used as a requirement for the update. An example configuration for testing SRP using BIND 9 is given in Appendix C.

#### Appendix B. How to allow services to update standard RFC2136-compliant servers

Ordinarily SRP Updates will fail when sent to an RFC 2136-compliant server that does not implement SRP because the zone being updated is "default.service.arpa", and no DNS server that is not an SRP server should normally be configured to be authoritative for "default.service.arpa". Therefore, a service that sends an SRP Update can tell that the receiving server does not support SRP, but does support RFC2136, because the RCODE will either be NOTZONE, NOTAUTH or REFUSED, or because there is no response to the update request (when using the anycast address)

In this case a service MAY attempt to register itself using regular RFC2136 DNS updates. To do so, it must discover the default registration zone and the DNS server designated to receive updates for that zone, as described earlier, using the \_dns-update.\_udp SRV record. It can then make the update using the port and host pointed to by the SRV record, and should use appropriate prerequisites to avoid overwriting competing records. Such updates are out of scope for SRP, and a service that implements SRP MUST first attempt to use SRP to register itself, and should only attempt to use RFC2136 backwards compatibility if that fails. Although the owner name for the SRV record specifies the UDP protocol for updates, it is also possible to use TCP, and TCP should be required to prevent spoofing.

#### Appendix C. Sample BIND9 configuration for default.service.arpa.

```
zone "default.service.arpa." {  
    type master;  
    file "/etc/bind/master/service.db";  
    allow-update { key demo.default.service.arpa.; };  
};
```

Figure 1: Zone Configuration in named.conf

```

$ORIGIN .
$TTL 57600 ; 16 hours
default.service.arpa IN SOA      ns3.default.service.arpa.
                                postmaster.default.service.arpa. (
                                2951053287 ; serial
                                3600      ; refresh (1 hour)
                                1800      ; retry (30 minutes)
                                604800    ; expire (1 week)
                                3600      ; minimum (1 hour)
                                )
                                NS       ns3.default.service.arpa.
                                SRV 0 0 53 ns3.default.service.arpa.
$ORIGIN default.service.arpa.
$TTL 3600 ; 1 hour
_ipps._tcp PTR demo._ipps._tcp
$ORIGIN _ipps._tcp.default.service.arpa.
demo TXT "0"
SRV 0 0 9992 demo.default.service.arpa.
$ORIGIN _udp.default.service.arpa.
$TTL 3600 ; 1 hour
_dns-update PTR ns3.default.service.arpa.
$ORIGIN _tcp.default.service.arpa.
_dnssd-srp PTR ns3.default.service.arpa.
$ORIGIN default.service.arpa.
$TTL 300 ; 5 minutes
ns3 AAAA 2001:db8:0:1::1
$TTL 3600 ; 1 hour
demo AAAA 2001:db8:0:2::1
KEY 513 3 13 (
    qweEmaaQ0FAWok5//ftuQtZgiZoiFSUsm0srWREdywQU
    9dpvtOhrdKWUuPT3uEFF5TZU6B4q1z1I662GdaUwqg==
); alg = ECDSA256SHA256 ; key id = 15008
AAAA ::1

```

Figure 2: Example Zone file

## Authors' Addresses

Ted Lemon  
 Apple Inc.  
 One Apple Park Way  
 Cupertino, California 95014  
 United States of America  
 Email: mellon@fugue.com

Stuart Cheshire  
Apple Inc.  
One Apple Park Way  
Cupertino, California 95014  
United States of America  
Phone: +1 408 974 3207  
Email: cheshire@apple.com