

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 10, 2016

C. Huitema
Microsoft
March 9, 2016

Privacy Extensions for DNS-SD
draft-huitema-dnssd-privacy-00.txt

Abstract

DNS-SD allows discovery of services published in DNS or MDNS. The publication normally disclose information about the device publishing the services. There are use cases where devices want to communicate without disclosing their identity, for example two mobile devices visiting the same hotspot. We propose a method to obfuscate the identification information published by DNS-SD.

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 10, 2016.

Copyright Notice

Copyright (c) 2016 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	2
1.1. Requirements	3
2. Privacy implications of DNS-SD	3
2.1. Privacy implication of publishing instance names	3
2.2. Privacy implication of publishing node names	4
2.3. Privacy implication of publishing service attributes	4
2.4. Device fingerprinting	5
2.5. Privacy implication of discovering services	5
3. Design of DNS-SD privacy mitigations	5
3.1. Obfuscated instance names	5
3.2. Randomized host names	6
3.3. Timing of obfuscation and randomization	7
3.4. Fingerprint resistance	7
3.5. A note on Private DNS services	7
4. Privacy extensions for DNS-SD	8
4.1. Randomized Host Name	8
4.2. Instance Discovery Key	8
4.3. Composing Obfuscated Instance Names	9
4.4. De-Obfuscation of Instance Names	9
5. Security Considerations	10
6. IANA Considerations	10
7. Acknowledgments	10
8. References	11
8.1. Normative References	11
8.2. Informative References	11
Author's Address	12

1. Introduction

There are cases when nodes connected to a network want to provide or consume services without exposing their identity to the other parties connected to the same network. Consider for example a traveller wanting to upload pictures from a phone to a laptop when connected to the Wi-Fi network of an Internet cafe, or two travellers who want to share files between their laptops when waiting for their plane in an airport lounge.

We expect that these exchanges will start with a discovery procedure using DNS-SD [RFC6763]. One of the devices will publish the availability of a service, such as a picture library or a file store in our examples. The user of the other device will discover this service, and then connect to it.

When analysing these scenarios in Section 2, we find that the DNS-SD messages leak identifying information such as instance name, host name or service properties. We review the design constraint of a solution in Section 3, and describe the proposed solution in Section 4.

1.1. Requirements

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

2. Privacy implications of DNS-SD

DNS-Based Service Discovery (DNS-SD) is defined in [RFC6763]. It allows nodes to publish the availability of an instance of a service by inserting specific records in the DNS ([RFC1033], [RFC1034], [RFC1035]) or by publishing these records locally using multicast DNS (MDNS) [RFC6762]. The service availability will be described in three types of records:

PTR Record: Associate the service name in the domain with the "instance" name published by the node.

SRV Record: Provides the node name, port number, priority and weight associated with the service instance, in conformance with [RFC2782].

TXT Record: Provides a set of attribute-value pairs describing specific properties of the service instance.

In the remaining subsections, we will review the privacy issues related to publishing instance names, node names, service attributes and other data, as well as review the implications of using the discovery service as a client.

2.1. Privacy implication of publishing instance names

In the first phase of discovery, the client will obtain a copy of all the PTR records associated to a service in a given naming domain. Each record contains a domain name starting with an instance name. Instance names are free form description of the instance, and are meant to convey enough information so discovery clients can easily select the desired service. Section 4 of [RFC6763] give the following example for the instance names of a printer service:

```
Building 2, 1st Floor . example . com .  
Building 2, 2nd Floor . example . com .  
Building 2, 3rd Floor . example . com .  
Building 2, 4th Floor . example . com .
```

Nodes that use DNS-SD in a mobile environment will rely on the specificity of the instance name to identify the desired service. In our example of users wanting to upload pictures to a laptop in an Internet Cafe, the list of available services may look like:

```
Alice's notebook      . local .  
Bob's laptop          . local .  
Image store for Carol . local .
```

Alice will see the list on her phone and understand intuitively that she should pick the first item. The discovery will "just work." It will also reveal to anybody who cares that Alice is currently visiting the Internet Cafe.

2.2. Privacy implication of publishing node names

The SRV records contain the DNS name of the node publishing the service. Typical implementations construct this DNS name by concatenating the "host name" of the node with the name of the local domain. The privacy implications of this practice are reviewed in [I-D.ietf-intarea-hostname-practice]. Depending on naming practices, the host name is either a strong identifier of the device, or at a minimum a partial identifier. It enables tracking of the device, and by extension of the device's owner.

2.3. Privacy implication of publishing service attributes

The TXT records contain a set of attribute and value pairs characteristics of the service implementation. These attributes reveal some information about the devices that publishes the service. The amount of information will vary widely with the particular service and its implementation:

- o Some attributes like the paper size available in a printer, are the same on many devices, and thus only provides limited information to a tracker.
- o Attributes that have freeform values, such as the name of a directory, may reveal much more information.

Combinations of attributes have more information power than specific attributes, and can potentially be used for "fingerprinting" a specific device.

2.4. Device fingerprinting

The combination of information published in DNS-SD has the potential to provide a "fingerprint" of a specific device. Such information includes:

- o The list of services published by the device, which can be retrieved because the SRV records will point to the same host name.
- o The specific attributes describing these services.
- o The port numbers used by the services.
- o The values of the priority and weight attributes in the SRV records.

This combination of services and attribute will often be sufficient to identify the version of the software running on a device. If a device publishes many services with rich sets of attributes, the combination may be sufficient to identify the specific device.

2.5. Privacy implication of discovering services

The consumers of services engage in discovery, and in doing so do reveal some information such as the list of services that they are interested in and the domains in which they are looking for the services. When the clients select specific instances of services, they reveal their preference for these instances.

In first analysis, the leakage of information by clients looks benign compared to the disclosures made by the servers. There may be a concern when the client is attempting to use rare services.

3. Design of DNS-SD privacy mitigations

Ah Ah.

3.1. Obfuscated instance names

The privacy issues described in Section 2.1 can be solved by obfuscating the instance names. Instead of a user friendly description of the instance, the nodes will publish a random looking string of characters. To prevent tracking over time and location, different string values should be used at different locations, or at different times.

Authorized parties should be able to "de-obfuscate" the names, while non-authorized third parties will not be. For example, if both Alice notebook and Bob's laptop use an obfuscation process, the list of available services should appear differently to them and to third parties. Alice's phone will be able to de-obfuscate the name of Alice's notebook, but not that of Bob's laptop. Bob's phone will do the opposite. Carol will do neither.

Alice will see something like:

```
GobbeldygookBlaBlaBla (Alice's notebook) . local .
Abracadabragooklybok . local .
Image store for Carol . local .
```

Bob will see:

```
GobbeldygookBlaBlaBla . local .
Abracadabragooklybok (Bob's laptop) . local .
Image store for Carol . local .
```

Carol will see:

```
GobbeldygookBlaBlaBla . local .
Abracadabragooklybok . local .
Image store for Carol . local .
```

In that example, Alice, Bob and Carol will be able to select the appropriate instance. It would probably be preferable to filter out the obfuscated instance names, to avoid confusing the user. In our example, Alice and Bob have updated their software to understand obfuscation, and they could easily filter out the obfuscated strings that they do not like. But Carol is not using this system, and we could argue that her experience is suboptimal.

The suboptimal experience with unmodified software could be avoided if the obfuscated service records were published using different service names, or using different domain names. This would of course make management a bit more complex, and is thus debatable.

3.2. Randomized host names

Instead of publishing their actual name in the SRV records, nodes could publish a randomized name. That the solution argued for in [I-D.ietf-intarea-hostname-practice].

Randomized host names will prevent some of the tracking. Host names are typically not visible by the users, and randomizing host names will probably not cause much usability issues.

3.3. Timing of obfuscation and randomization

It is important that obfuscation of instance names be performed at the right time, and that the obfuscated names change in synchrony with other identifiers, such as MAC Addresses, IP Addresses or host names. If the randomized host name changed but the instance name remained constant, an adversary would have no difficulty linking the old and new host names. Similarly, if IP or MAC addresses changed but host names remained constant, the adversary could link the new addresses to the old ones using the published name.

The problem is handled in [I-D.ietf-intarea-hostname-practice], which recommends to pick a new random host name at the time of connecting to a new network. The instance names should be obfuscated at the same time, or maybe use the randomized host name as input in the randomization process.

3.4. Fingerprint resistance

Difficult...

3.5. A note on Private DNS services

The DNS Private Exchange working group develops mechanisms to provide confidentiality to DNS transactions, addressing the problems outlined in [RFC7626]. The solutions being developed include DNS over TLS [I-D.ietf-dprive-dns-over-tls] and DNS over DTLS [I-D.ietf-dprive-dnsodtls].

We could imagine that DNS-SD nodes are configured to update and retrieve DNS records using DNS over TLS or DNS over DTLS, but a number of problems can arise:

- o Discovery queries are scoped by the domain name within which services are published. As nodes move and visit arbitrary networks, there is no guarantee that the domain services for these networks will be accessible using DNS over TLS or DNS over DTLS.
- o Information placed in the DNS is considered public. Even if the server does support DNS over TLS, third parties will still be able to discover the content of PTR, SRV and TXT records.
- o Neither DNS over TLS nor DNS over DTLS applies to MDNS.

In short, DNS over TLS and DNS over DTLS solve a different problem, and are not a solution for DNS-SD privacy.

4. Privacy extensions for DNS-SD

The proposed solution uses the following components:

- o The host names are randomized to prevent tracking.
- o Nodes provide an Instance Discovery Key to other nodes authorized to discover the service instance,
- o The Instance Discovery Key is combined with a random seed to obfuscate the instance names,
- o Nodes engaged in discovery attempt to de-obfuscate the instance names using the set of Instance Discovery Key that they know about,

These components are detailed in the following subsections.

4.1. Randomized Host Name

Nodes publishing services with DNS-SD and concerned about their privacy MUST use a randomized host name. The randomized name MUST be changed when network connectivity changes, to avoid the correlation issues described in Section 3.3. The randomized host name MUST be used in the SRV records describing the service instance, and the corresponding A or AAAA records MUST be made available through DNS or MDNS, within the same scope as the PTR, SRV and TXT records used by DNS-SD.

If the link-layer address of the network connection is properly obfuscated (e.g. using MAC Address Randomization), The Randomized Host Name MAY be computed using the algorithm described in section 3.7 of [I-D.ietf-dhc-anonymity-profile]. If this is not possible, the randomized host name SHOULD be constructed by simply picking a 48 bit random number meeting the Randomness Requirements for Security expressed in [RFC4075], and then use the hexadecimal representation of this number as the obfuscated host name.

4.2. Instance Discovery Key

The obfuscation and de-obfuscation of instance names is controlled by the Instance Discovery Key. Each device publishing a service instance configures an Instance Discovery Key associated with the service instance.

The Instance Key SHOULD be at least 16 bytes long (128 bits). Its content SHOULD meet the Randomness Requirements for Security expressed in [RFC4075].

4.3. Composing Obfuscated Instance Names

The obfuscated instance name is composed of two components, a seed and a hash, encoded in BASE64 ([RFC2045] section 6.8) and separated by a dot:

```
instance_name = <base64_seed> "." <base64_hash>
```

The seed is derived algorithmically from the randomized host name. If the randomized name changes, new instance names SHOULD be computed and the corresponding records SHOULD be published in order to meet the requirement defined in Section 3.3.

The complete instance name MUST be generated using the following process:

```
long_seed = HASH(randomized_host_name)
seed = first 12 bytes of long_seed
long_hash = HASH(seed | instance_discovery_key )
instance_hash = first 12 bytes of long_hash
instance_name = BASE64(seed) "." BASE64(instance_hash)
```

In this formula, HASH SHOULD be the function SHA256 defined in [RFC4055], unless otherwise specified. Implementers MAY eventually replace SHA256 with a stronger algorithm.

The algorithm produces seeds and hash that are encoded as 16 BASE64 characters. The resulting instance name is 33 characters long, which fits within the 63 characters limit defined in [RFC6763].

4.4. De-Obfuscation of Instance Names

De-obfuscation of instance names assumes that authorized nodes are provisioned with three elements for each discoverable instance:

- o the de-obfuscated instance name,
- o a copy of the instance_discovery_key,
- o optionally, the identifier of the HASH function used by the publisher.

A given node may be provisioned to discover many instances. For example, Alice's phone may know about Alice's laptop and Alice's desktop. It might also know of Bob's laptop, if Alice and Bob have agreed to share such information.

To de-obfuscate the instance names, nodes performing discovery should obtain the list of PTR records published for the service and domain being searched and then do the following:

- o Test whether the instance name contains the base64 encoding of a seed and hash as defined in Section 4.3. If it is not in that form, the name is not considered obfuscated.
- o Retrieve the binary seed and hash from the base64 encoding.
- o For each known instance discovery key, compute whether the hash of the seed and key, and compare it to the published hash.
- o If there is a hash, the de-obfuscated name of the instance is the de-obfuscated name associated with the matching instance discovery key

5. Security Considerations

This document specifies a method to protect the privacy of service publishing nodes. This is especially useful when operating in a public space. Obfuscating the identity of the publishing nodes prevents some forms of "targeting" of high value nodes.

Obfuscating the identity of the publishing nodes does not provide any form of access control. It will not prevent attackers from trying to access the services.

The cost of the de-obfuscation algorithm scales as the product of the number of authorized publishers known by the client, times the number of obfuscated services published in the searched name domain. Attackers could potentially publish a large number of bogus instances of a service, forcing a high computation cost on discovery clients. While this potential denial of service attack is concerning, we note that this is merely an aggravation of a flooding attacks against DNS-SD.

6. IANA Considerations

This draft does not require any IANA action.

7. Acknowledgments

This draft results from initial discussions with Dave Thaler.

8. References

8.1. Normative References

- [RFC2045] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, DOI 10.17487/RFC2045, November 1996, <<http://www.rfc-editor.org/info/rfc2045>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC4055] Schaad, J., Kaliski, B., and R. Housley, "Additional Algorithms and Identifiers for RSA Cryptography for use in the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 4055, DOI 10.17487/RFC4055, June 2005, <<http://www.rfc-editor.org/info/rfc4055>>.
- [RFC4075] Kalusivalingam, V., "Simple Network Time Protocol (SNTP) Configuration Option for DHCPv6", RFC 4075, DOI 10.17487/RFC4075, May 2005, <<http://www.rfc-editor.org/info/rfc4075>>.
- [RFC6763] Cheshire, S. and M. Krochmal, "DNS-Based Service Discovery", RFC 6763, DOI 10.17487/RFC6763, February 2013, <<http://www.rfc-editor.org/info/rfc6763>>.

8.2. Informative References

- [I-D.ietf-dhc-anonymity-profile]
Huitema, C., Mrugalski, T., and S. Krishnan, "Anonymity profile for DHCP clients", draft-ietf-dhc-anonymity-profile-08 (work in progress), February 2016.
- [I-D.ietf-dprive-dns-over-tls]
Zi, Z., Zhu, L., Heidemann, J., Mankin, A., Wessels, D., and P. Hoffman, "Specification for DNS over TLS", draft-ietf-dprive-dns-over-tls-07 (work in progress), March 2016.
- [I-D.ietf-dprive-dnsodtls]
Reddy, T., Wing, D., and P. Patil, "DNS over DTLS (DNSoD)", draft-ietf-dprive-dnsodtls-04 (work in progress), January 2016.

- [I-D.ietf-intarea-hostname-practice]
Huitema, C. and D. Thaler, "Current Hostname Practice Considered Harmful", draft-ietf-intarea-hostname-practice-00 (work in progress), October 2015.
- [RFC1033] Lottor, M., "Domain Administrators Operations Guide", RFC 1033, DOI 10.17487/RFC1033, November 1987, <<http://www.rfc-editor.org/info/rfc1033>>.
- [RFC1034] Mockapetris, P., "Domain names - concepts and facilities", STD 13, RFC 1034, DOI 10.17487/RFC1034, November 1987, <<http://www.rfc-editor.org/info/rfc1034>>.
- [RFC1035] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, DOI 10.17487/RFC1035, November 1987, <<http://www.rfc-editor.org/info/rfc1035>>.
- [RFC2782] Gulbrandsen, A., Vixie, P., and L. Esibov, "A DNS RR for specifying the location of services (DNS SRV)", RFC 2782, DOI 10.17487/RFC2782, February 2000, <<http://www.rfc-editor.org/info/rfc2782>>.
- [RFC6762] Cheshire, S. and M. Krochmal, "Multicast DNS", RFC 6762, DOI 10.17487/RFC6762, February 2013, <<http://www.rfc-editor.org/info/rfc6762>>.
- [RFC7626] Bortzmeyer, S., "DNS Privacy Considerations", RFC 7626, DOI 10.17487/RFC7626, August 2015, <<http://www.rfc-editor.org/info/rfc7626>>.

Author's Address

Christian Huitema
Microsoft
Redmond, WA 98052
U.S.A.

Email: huitema@microsoft.com

Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: August 7, 2016

S. Cheshire
Apple Inc.
February 4, 2016

Hybrid Unicast/Multicast DNS-Based Service Discovery
draft-ietf-dnssd-hybrid-03

Abstract

Performing DNS-Based Service Discovery using purely link-local Multicast DNS enables discovery of services that are on the local link, but not (without some kind of proxy or similar special support) discovery of services that are outside the local link. Using a very large local link with thousands of hosts facilitates service discovery, but at the cost of large amounts of multicast traffic.

Performing DNS-Based Service Discovery using purely Unicast DNS is more efficient and doesn't require excessively large multicast domains, but requires that the relevant data be available in the Unicast DNS namespace. This can be achieved by manual DNS configuration (as has been done for many years at IETF meetings to advertise the IETF Terminal Room printer) but this is labor intensive, error prone, and requires a reasonable degree of DNS expertise. The Unicast DNS namespace can be populated with the required data automatically by the devices themselves, but that requires configuration of DNS Update keys on the devices offering the services, 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.

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 August 7, 2016.

Copyright Notice

Copyright (c) 2016 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	4
2. Conventions and Terminology Used in this Document	5
3. Compatibility Considerations	5
4. Hybrid Proxy Operation	6
4.1. Delegated Subdomain for Service Discovery Records	7
4.2. Domain Enumeration	8
4.2.1. Domain Enumeration via Unicast Queries	8
4.2.2. Domain Enumeration via Multicast Queries	9
4.3. Delegated Subdomain for LDH Host Names	10
4.4. Delegated Subdomain for Reverse Mapping	12
4.5. Data Translation	13
4.5.1. DNS TTL limiting	13
4.5.2. Suppressing Unusable Records	14
4.5.3. Application-Specific Data Translation	15
4.6. Answer Aggregation	16
4.6.1. Discovery of LLQ and/or PUSH Notification Service	19
5. DNS SOA (Start of Authority) Record	20
6. Implementation Status	20
6.1. Already Implemented and Deployed	20
6.2. Already Implemented	21
6.3. Partially Implemented	21
6.4. Not Yet Implemented	21
7. IPv6 Considerations	22
8. Security Considerations	22
8.1. Authenticity	22
8.2. Privacy	22
8.3. Denial of Service	23
9. Intellectual Property Rights	23
10. IANA Considerations	23
11. Acknowledgments	23
12. References	23
12.1. Normative References	23
12.2. Informative References	24
Author's Address	25

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].

For a small network consisting of just a single link (or several physical links bridged together to appear as a single logical link to IP) Multicast DNS [RFC6762] is sufficient for client devices to look up the dot-local host names of peers on the same home network, and perform DNS-Based Service Discovery (DNS-SD) [RFC6763] of 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, do not cross between links. (This was a deliberate design choice for Multicast DNS, since even on a single link multicast traffic is expensive -- especially on Wi-Fi links -- and multiplying the amount of multicast traffic by flooding it across multiple links would make that problem even worse.) In this environment, Unicast DNS would be preferable to Multicast DNS. (Unicast DNS can be used either with a traditionally assigned globally unique domain name, or with a private local unicast domain name such as ".home" [HOME].)

To use Unicast DNS, the names of hosts and services need to be made available in the Unicast DNS namespace. 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.

This document specifies a type of proxy called a Hybrid 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.

2. 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" [RFC2119].

The Hybrid Proxy builds on Multicast DNS, which works between hosts on the same link. A set of hosts is considered to be "on the same link" if:

- o when any host A from that set sends a packet to any other host B 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 [802.5], 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.

3. Compatibility Considerations

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

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

Existing clients that support DNS-Based Service Discovery over Unicast DNS (Mac OS X 10.4 and later, including iPhone, iPad, and Bonjour for Windows) work with Hybrid Proxy.

4. Hybrid Proxy Operation

In a typical configuration, a Hybrid Proxy is configured to be authoritative [RFC1034] [RFC1035] for four 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.

A DNS subdomain for IPv6 Reverse Mapping records.

This subdomain name will be a name that ends in "ip6.arpa."

A DNS subdomain for IPv4 Reverse Mapping records.

This subdomain name will be a name that ends in "in-addr.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 [I-D.ietf-homenet-hncp].

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

4.1. Delegated Subdomain for Service Discovery Records

In its simplest form, each physical 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 a Hybrid Proxy which serves it. This Hybrid Proxy function could be performed by a router on that link, or, with appropriate VLAN configuration, a single Hybrid Proxy could have a logical presence on, and serve as the Hybrid Proxy for, many links. In the parent domain, NS records are used to delegate ownership of each defined link name (e.g., "Building 1.example.com") to the Hybrid Proxy that serves the named link. In other words, the Hybrid Proxy is the authoritative name server for that subdomain.

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 Hybrid Proxy on the link in question. Like a conventional Unicast DNS server, a Hybrid 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 Hybrid Proxy generates answers using Multicast DNS. A Hybrid Proxy does this by consulting its Multicast DNS cache and/or issuing Multicast DNS queries for the corresponding Multicast DNS name, type and class, (e.g., in this

case, "_printer._tcp.local. PTR ?"). Then, from the received Multicast DNS data, the Hybrid Proxy synthesizes the appropriate Unicast DNS response.

Naturally, the existing Multicast DNS caching mechanism is used to avoid issuing unnecessary Multicast DNS queries on the wire. The Hybrid 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.

4.2. Domain Enumeration

An DNS-SD client performs Domain Enumeration [RFC6763] via certain PTR queries. It issues unicast Domain Enumeration queries using its "home" domain (typically learned via DHCP) and using its IPv6 prefix and IPv4 subnet address. These are described below in Section 4.2.1. It also issues multicast Domain Enumeration queries in the "local" domain [RFC6762]. These are described below in Section 4.2.2. The results of all Domain Enumeration queries are combined for Service Discovery purposes.

4.2.1. Domain Enumeration via Unicast Queries

The administrator creates Domain Enumeration PTR records [RFC6763] to inform clients of available service discovery domains, e.g.,:

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 "b" ("browse") records tell the client device the list of browsing domains to display for the user to select from and the "db" ("default browse") record tells the client device which domain in that list should be 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, as of 2015). 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.

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.

4.2.2. Domain Enumeration via Multicast Queries

Since a Hybrid Proxy exists on many, if not all, the links in an enterprise, it offers an additional way to provide Domain Enumeration data for clients.

A Hybrid Proxy can be configured to generate Multicast DNS responses for the following Multicast DNS Domain Enumeration queries issues 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 Hybrid Proxies to provide configuration data on a per-link granularity to DNS-SD clients. In some enterprises it may be preferable to provide this per-link configuration data in the form of Hybrid Proxy configuration, rather than populating the Unicast DNS servers with the same data (in the "ip6.arpa" or "in-addr.arpa" domains).

4.3. Delegated Subdomain for LDH Host Names

The traditional rules for host names are more restrictive than those for DNS-SD service instance names and domains.

Users typically interact with DNS-SD by viewing a list of discovered service instance names on the 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 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, with no spaces or other punctuation.

While we still 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 Hybrid Proxy SHOULD support having separate subdomains delegated to it, 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).

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

```
My Printer._ipp._tcp.local. SRV 0 0 631 prnt.local.  
prnt.local.                A    10.0.1.2
```

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    10.0.1.2
```

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

4.4. Delegated Subdomain for Reverse Mapping

A Hybrid 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 Hybrid Proxy. In other words, the Hybrid Proxy becomes the authoritative name server for the reverse mapping domain.

For example, if a given link is using the IPv6 prefix 2001:0DB8/32, then the domain "8.b.d.0.1.0.0.2.ip6.arpa" is delegated to the Hybrid Proxy for that link.

If a given link is using the IPv4 subnet 10.1/16, then the domain "1.10.in-addr.arpa" is delegated to the Hybrid Proxy for that link.

When a reverse mapping query arrives at the Hybrid Proxy, it issues the identical query on its local link as a Multicast DNS query. (In the Apple "/usr/include/dns_sd.h" APIs, using ForceMulticast indicates that the `DNSServiceQueryRecord()` call should perform the query using Multicast DNS.) When the host owning that IPv6 or IPv4 address responds with a name of the form "something.local", the Hybrid Proxy rewrites that to use its configured LDH host name domain instead of "local" and returns the response to the caller.

For example, a Hybrid Proxy with the two subdomains "1.10.in-addr.arpa" and "bldg1.example.com" delegated to it would translate this Multicast DNS record:

```
3.2.1.10.in-addr.arpa. PTR prnt.local.
```

into this Unicast DNS response:

```
3.2.1.10.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 Hybrid Proxy, will arrive at the Hybrid 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.

4.5. 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 configured DNS Unicast domain.

Other beneficial translation and filtering operations are described below.

4.5.1. 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 neighbouring 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 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 any Long-Lived Query [I-D.sekar-dns-llq] or DNS Push Notification [I-D.ietf-dnssd-push] option) the DNS TTLs reported in the resulting Unicast DNS response SHOULD 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 5).

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 contain an LLQ or DNS Push Notification option, 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 4.6.

4.5.2. Suppressing Unusable Records

A Hybrid Proxy SHOULD suppress Unicast DNS answers for records that are not useful outside the local link. For example, DNS A and AAAA records for IPv6 link-local addresses [RFC4862] and IPv4 link-local addresses [RFC3927] should be suppressed. Similarly, for sites that have multiple private address realms [RFC1918], private addresses from one private address realm should not be communicated to clients in a different private address realm.

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.

4.5.3. 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 Hybrid 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 Hybrid 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 dot-local hostname needs to be translated to an appropriate name with larger scope. Dot-local names are easily translated 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 Hybrid 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.

As is the case with NAT ALGs, protocol designers are advised to avoid communicating names and addresses in nonstandard locations, because those "hidden" names and addresses are at risk of not being translated when necessary, resulting in operational failures. 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 Hybrid 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 Hybrid Proxy MAY operate solely at the mDNS layer, without any knowledge of semantics at the DNS-SD layer or above.

4.6. 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 Hybrid 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 Hybrid 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) [I-D.sekar-dns-llq] or its newer replacement, DNS Push Notifications [I-D.ietf-dnssd-push]. (Clients and Hybrid Proxies can support both DNS LLQ and DNS Push, and when talking to a Hybrid 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 Hybrid Proxy receives a query containing a DNS LLQ or DNS Push Notification option, 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 Hybrid 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 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 Hybrid 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 included an LLQ or DNS Push Notification option (typical with long-lived service browsing PTR queries) or not (typical with one-shot operations like SRV or address record queries). Note that queries containing the LLQ or PUSH option are received directly from the client (see Section 4.6.1). Queries containing no LLQ or PUSH option are generally received via the client's configured recursive (caching) name server.

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

- o No LLQ or PUSH option; 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 an NSEC record [RFC6762]), the Hybrid 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, return a negative response to the remote client.

DNS TTLs in responses are capped to at most ten seconds.

- o No LLQ or PUSH option; at least one answer in cache:
Send response right away to minimise delay.
DNS TTLs in responses are capped to at most ten seconds.
No local mDNS queries are performed.
(Reasoning: 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 Query contains LLQ or PUSH option; 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. (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 state, and if mDNS answers are received later, LLQ or PUSH update messages are sent.
DNS TTLs in responses are returned unmodified.
- o Query contains LLQ or PUSH option; 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 state, and if additional mDNS answers are received later, LLQ or PUSH update 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.

Note that the "negative responses" referred to above are "no error no answer" negative responses, not NXDOMAIN. This is because the Hybrid 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.

4.6.1. Discovery of LLQ and/or PUSH Notification Service

To issue LLQ or PUSH queries, clients need to communicate directly with the authoritative Hybrid Proxy. The procedure by which the client locates the authoritative Hybrid Proxy is described in the LLQ specification [I-D.sekar-dns-llq] and the DNS Push Notifications specification [I-D.ietf-dnssd-push].

Briefly, the procedure is as follows:

To discover the LLQ service for a given domain name, a client first performs DNS zone apex discovery, and then, having discovered <apex>, the client then issues a DNS query for the SRV record with the name `_dns-llq._udp.<apex>` to find the target host and port for the LLQ service for that zone. By default LLQ service runs on UDP port 5352, but since SRV records are used, the LLQ service can be offered on any port.

To discover the DNS Push Notification service for a given domain name, a client first performs DNS zone apex discovery, and then, having discovered <apex>, the client then issues a DNS query for the SRV record with the name `_dns-push-tls._tcp.<apex>` to find the target host and port for the DNS Push Notification service for that zone. By default DNS Push Notification service runs on TCP port 5352, but since SRV records are used, the DNS Push Notification service can be offered on any port.

A client performs DNS zone apex discovery using the procedure below:

1. The client issues a DNS query for the SOA record with the given domain name.
2. A conformant recursive (caching) name server will either send a positive response, or a negative response containing the SOA record of the zone apex in the Authority Section.
3. If the name server sends a negative response that does not contain the SOA record of the zone apex, the client trims the first label off the given domain name and returns to step 1 to try again.

By this method, the client iterates until it learns the name of the zone apex, or (in pathological failure cases) reaches the root and gives up.

Normal DNS caching is used to avoid repetitive queries on the wire.

5. DNS SOA (Start of Authority) Record

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

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

The SERIAL field SHOULD contain a sequence number that increments each time the Hybrid Proxy returns an SOA record to any client.
[Author's note: Or maybe it could just be zero?]

Since zone transfers are undefined for Hybrid Proxy zones, the REFRESH, RETRY and EXPIRE fields have no useful meaning for Hybrid 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 4.5.1).

[Author's note: Discussion of these recommendations is requested.]

6. 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.

6.1. 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 you Press Cmd-P on your Mac, or select AirPrint on your iPad or iPhone, and the Terminal room

printers appear, that is because your client is sending unicast DNS queries to the IETF DNS servers.

6.2. Already Implemented

A minimal portable Hybrid 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 also has an implementation that runs on any Unix/Linux. It has a RESTful interface for management and an experimental demo CLI and web interface.

6.3. 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 Hybrid Proxy deployment becomes common, then application writers will have a reason to provide better UI. Existing applications will work with the Hybrid Proxy, but will show all services in a single flat list. Applications with improved UI will group services by domain.

The Long-Lived Query mechanism [I-D.sekar-dns-llq] referred to in this specification exists and is deployed, but has not been standardized by the IETF. The IETF is considering standardizing a superior Long-Lived Query mechanism called DNS Push Notifications [I-D.ietf-dnssd-push]. The pragmatic short-term deployment approach is for vendors to produce Hybrid Proxies that implement both the deployed Long-Lived Query mechanism [I-D.sekar-dns-llq] (for today's clients) and the new DNS Push Notifications mechanism [I-D.ietf-dnssd-push] as the preferred long-term direction.

The translating/filtering Hybrid Proxy specified in this document. Implementations are under development, and operational experience with these implementations has guided updates to this document.

6.4. Not Yet Implemented

Client implementations of the new DNS Push Notifications mechanism [I-D.ietf-dnssd-push] are currently underway.

A mechanism to 'stitch' together multiple ".local." zones so that they appear as one. Such a mechanism will be specified in a future companion document.

7. IPv6 Considerations

An IPv6-only host and an IPv4-only host behave as "ships that pass in the night". Even if they are on the same Ethernet, neither is aware of the other's traffic. For this reason, each physical link may have **two** unrelated ".local." zones, one for IPv6 and one for IPv4. Since for practical purposes, a group of IPv6-only hosts and a group of IPv4-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 IPv6-only and the other IPv4-only, which are both trying to use the same name(s). Such a mechanism will be specified in a future companion document.

8. Security Considerations

8.1. 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 Hybrid Proxy mechanism should not be used, and something with stronger security should be used instead, such as authenticated secure DNS Update [RFC2136] [RFC3007].

8.2. 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 Hybrid 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 and split-view DNS, as are customarily used today to protect the privacy of corporate DNS information.

8.3. Denial of Service

A remote attacker could use a rapid series of unique Unicast DNS queries to induce a Hybrid Proxy to generate a rapid series of corresponding Multicast DNS queries on one or more of its local links. Multicast traffic is expensive -- especially on Wi-Fi links -- which makes this attack particularly serious. To limit the damage that can be caused by such attacks, a Hybrid 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 Wi-Fi links the Multicast DNS subsystem SHOULD NOT issue more than 20 Multicast DNS query packets per second. On other link technologies like Gigabit Ethernet higher limits may be appropriate.

9. Intellectual Property Rights

Apple has submitted an IPR disclosure concerning the technique proposed in this document. Details are available on the IETF IPR disclosure page [IPR2119].

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 and Andrew Yourtchenko for their comments. [Partial list; more names to be added.]

12. References

12.1. Normative References

- [RFC1034] Mockapetris, P., "Domain names - concepts and facilities", STD 13, RFC 1034, DOI 10.17487/RFC1034, November 1987, <<http://www.rfc-editor.org/info/rfc1034>>.
- [RFC1035] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, DOI 10.17487/RFC1035, November 1987, <<http://www.rfc-editor.org/info/rfc1035>>.

- [RFC1918] Rekhter, Y., Moskowitz, B., Karrenberg, D., J. de Groot, G., and E. Lear, "Address Allocation for Private Internets", BCP 5, RFC 1918, DOI 10.17487/RFC1918, February 1996, <<http://www.rfc-editor.org/info/rfc1918>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC2308] Andrews, M., "Negative Caching of DNS Queries (DNS NCACHE)", RFC 2308, DOI 10.17487/RFC2308, March 1998, <<http://www.rfc-editor.org/info/rfc2308>>.
- [RFC3927] Cheshire, S., Aboba, B., and E. Guttman, "Dynamic Configuration of IPv4 Link-Local Addresses", RFC 3927, DOI 10.17487/RFC3927, May 2005, <<http://www.rfc-editor.org/info/rfc3927>>.
- [RFC4862] Thomson, S., Narten, T., and T. Jinmei, "IPv6 Stateless Address Autoconfiguration", RFC 4862, DOI 10.17487/RFC4862, September 2007, <<http://www.rfc-editor.org/info/rfc4862>>.
- [RFC5198] Klensin, J. and M. Padlipsky, "Unicode Format for Network Interchange", RFC 5198, DOI 10.17487/RFC5198, March 2008, <<http://www.rfc-editor.org/info/rfc5198>>.
- [RFC6762] Cheshire, S. and M. Krochmal, "Multicast DNS", RFC 6762, December 2012.
- [RFC6763] Cheshire, S. and M. Krochmal, "DNS-Based Service Discovery", RFC 6763, December 2012.
- [I-D.ietf-dnssd-push] Pusateri, T. and S. Cheshire, "DNS Push Notifications", draft-ietf-dnssd-push-03 (work in progress), November 2015.

12.2. Informative References

- [HOME] Cheshire, S., "Special Use Top Level Domain 'home'", draft-cheshire-homenet-dot-home (work in progress), November 2015.
- [IPR2119] "Apple Inc.'s Statement about IPR related to Hybrid Unicast/Multicast DNS-Based Service Discovery", <<https://datatracker.ietf.org/ipr/2119/>>.

- [ohp] "Hybrid Proxy implementation for OpenWrt",
<<https://github.com/sbyx/ohybridproxy/>>.
- [I-D.sekar-dns-llq]
Sekar, K., "DNS Long-Lived Queries",
draft-sekar-dns-llq-01 (work in progress), August 2006.
- [I-D.ietf-homenet-hncc]
Stenberg, M., Barth, S., and P. Pfister, "Home Networking
Control Protocol", draft-ietf-homenet-hncc-09 (work in
progress), August 2015.
- [RFC2136] Vixie, P., Ed., Thomson, S., Rekhter, Y., and J. Bound,
"Dynamic Updates in the Domain Name System (DNS UPDATE)",
RFC 2136, DOI 10.17487/RFC2136, April 1997,
<<http://www.rfc-editor.org/info/rfc2136>>.
- [RFC3007] Wellington, B., "Secure Domain Name System (DNS) Dynamic
Update", RFC 3007, DOI 10.17487/RFC3007, November 2000,
<<http://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, December 2012.
- [ZC] Cheshire, S. and D. Steinberg, "Zero Configuration
Networking: The Definitive Guide", O'Reilly Media, Inc. ,
ISBN 0-596-10100-7, December 2005.

Author's Address

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

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

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".

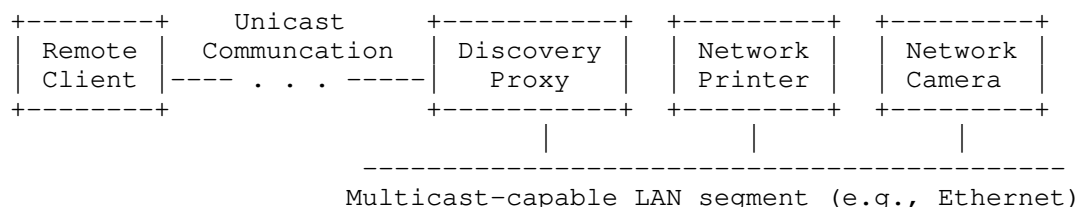
An alternative proposal that has been discussed is a proxy that performs DNS updates to a remote DNS server on behalf of the Multicast DNS devices on the local network. The difficulty with this is that Multicast DNS devices do not routinely announce their records on the network. Generally they remain 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 RRSset 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

DNSSD
Internet-Draft
Intended status: Informational
Expires: May 3, 2016

A. Sullivan
Dyn
October 31, 2015

On Interoperation of Labels Between DNS and Other Resolution Systems
draft-ietf-dnssd-mdns-dns-interop-02

Abstract

Despite its name, DNS-Based Service Discovery can use naming systems other than the Domain Name System when looking for services. Moreover, when it uses the DNS, DNS-Based Service Discovery uses the full capability of DNS, rather than using a subset of available octets. In order for DNS-SD to be used effectively in environments where multiple different name systems and conventions for their operation are in use, it is important to attend to differences in the underlying technology and operational environment. This memo presents an outline of the requirements for selection of labels for conventional DNS and other resolution systems when they are expected to interoperate in this manner.

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 May 3, 2016.

Copyright Notice

Copyright (c) 2015 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	2
1.1. Conventions and terms used in this document	3
2. Why there could be a problem at all	4
3. Requirements for a profile for label interoperation	5
4. DNS-SD portions	6
4.1. The <Instance> Portion of the Service Instance Name	6
4.2. The <Service> Portion of the Service Instance Name	7
4.3. The <Domain> Portion of the Service Instance Name	7
5. Acknowledgements	8
6. IANA Considerations	9
7. Security Considerations	9
8. Informative References	9
Appendix A. Change History	10
A.1. draft-ietf-dnssd-mdns-dns-interop-02	10
A.2. draft-ietf-dnssd-mdns-dns-interop-01	11
A.3. draft-ietf-dnssd-mdns-dns-interop-00	11
A.4. draft-sullivan-dnssd-mdns-dns-interop-01	11
A.5. draft-sullivan-dnssd-mdns-dns-interop-00	11
Author's Address	11

1. Introduction

DNS-Based Service Discovery (DNS-SD, [RFC6763]) specifies a mechanism for discovering services using queries to the Domain Name System (DNS, [RFC1034], [RFC1035]); and to any other system that uses domain names, such as Multicast DNS (mDNS, [RFC6762]). Conventional use of the DNS generally follows the host name rules [RFC0952] for labels -- the so-called LDH rule. That convention is the reason behind the development of Internationalized Domain Names for Applications (IDNA2008, [RFC5890], [RFC5891], [RFC5892], [RFC5893], [RFC5894], [RFC5895]). It is worth noting that the LDH rule is a convention, and not a rule of the DNS; this is made entirely plain by [RFC2181], section 11. Nevertheless, there is a widespread belief that in many circumstances domain names cannot be used in the DNS unless they cleave to the LDH rule.

At the same time, mDNS requires that labels be encoded in UTF-8, and permits a range of characters in labels that are not permitted by IDNA2008 or the LDH rule. For example, mDNS encourages the use of spaces and punctuation in mDNS names (see [RFC6763], section 4.1.3). It does not restrict which Unicode code points may be used in those labels, so long as the code points are UTF-8 in Net-Unicode [RFC5198] format.

Users of applications are, of course, frequently unconcerned with (not to say oblivious to) the name-resolution system(s) in service at any given moment, and are inclined simply to use the same domain names in different contexts. As a result, the same domain name might be tried using different name resolution technologies. If DNS-SD is to be used in an environment where multiple resolution systems (such as mDNS and DNS) are to be queried for services, then some parts of the domain names to be queried will need to be compatible with the rules and conventions for all the relevant technologies.

One approach to interoperability under these circumstances is to use a single operational convention (a "profile") for domain names under the different naming systems. This memo assumes such a use profile, and attempts to outline what is necessary to make it work without specifying any particular technology. It does assume, however, that the global DNS is eventually likely to be implicated. Given the general tendency of all resolution eventually to fall through to the DNS, that assumption does not seem controversial.

It is worth noting that users of DNS-SD do not use the service discovery names in the same way that users of other domain names might. Domain names often might as easily be entered as direct user input as by any other method. But the service discovery context generally assumes users are picking a service from a list. As a result, the sorts of application considerations that are appropriate to the general-purpose DNS name, and that resulted in the A-label/U-label split (see below) in IDNA2008, are not entirely the right approach for DNS-SD.

1.1. Conventions and terms used in this document

Wherever appropriate, this memo uses the terminology defined in Section 2 of [RFC5890]. In particular, the reader is assumed to be familiar with the terms "U-label", "LDH label", and "A-label" from that document. Similarly, the reader is assumed to be familiar with the U+NNNN notation for Unicode code points used in [RFC5890] and other documents dealing with Unicode code points. In the interests of brevity and consistency, the definitions are not repeated here.

Sometimes this memo refers to names in the DNS as though the LDH rule and IDNA2008 are strict requirements. They are not. DNS labels are, in principle, just collections of octets, and therefore in principle the LDH rule is not a constraint. In practice, applications sometimes intercept labels that do not conform to the LDH rule and apply IDNA and other transformations.

The DNS, perhaps unfortunately, has produced its own jargon. Unfamiliar DNS-related terms in this memo should be found in [I-D.ietf-dnsop-dns-terminology].

The term "owner name" (common to the DNS vernacular; see above) is used here to apply not just to the domain names to be looked up in the DNS, but to any name that might be looked up either in the DNS or using other technologies. It therefore includes names that might not actually exist anywhere. In addition, what follows depends on the idea that not every domain name may be looked up in the DNS. For instance, names ending in "local." (in the presentation format) are not ordinarily looked up in the DNS, but instead by querying mDNS.

DNS-SD specifies three portions of the owner name for a DNS-SD resource record. These are the <Instance> portion, the <Service> portion, and the <Domain>. The owner name made of these three parts is called the Service Instance Name. It is worth observing that a portion may be more than one label long. See [RFC6763], section 4.1. Further discussion of the parts is found in Section 4.

Throughout this memo, mDNS is used liberally as the alternative resolution mechanism to DNS. This is for convenience rather than rigour: any alternative name resolution to DNS could present the same friction with the prevailing operational conventions of the global DNS. It so happens that mDNS is the overwhelmingly successful alternative as of this writing, so it is used in order to make the issues plainer to the reader. Other alternative resolution mechanisms may in general be read wherever mDNS appears in the text, except where details of the mDNS specification appear.

2. Why there could be a problem at all

One might reasonably wonder why there is a problem to be solved at all. After all, DNS labels permit any octet whatsoever, and anything that can be useful with DNS-SD cannot use any names that are outside the protocol strictures of the DNS.

The reason for the trouble is twofold. First, and least troublesome, is the possibility of resolvers that are attempting to offer IDNA service system-wide. Given the design of IDNA2008, it is reasonable to suppose that on some systems high-level name resolution libraries

will perform the U-label/A-label transformation automatically, saving applications from these details. If this were the main problem, however, it would presumably be self-correcting; for the right answer would be, "Don't use those libraries for DNS-SD," and DNS-SD would not work reliably in cases where such libraries were in use. This would be unfortunate; but given that DNS-SD in Internet contexts is as of this writing not in ubiquitous use, it should not represent a fatal issue.

The greater problem is that the "infrastructure" types of DNS service -- the root zone, the top-level domains, and so on -- have embraced IDNA and refuse registration of raw UTF-8 into their zones. As of this writing there is (perhaps unfortunately) no reliable way to discover where these sorts of DNS services end. Nevertheless, some client programs (notably web browsers) have adopted a number of different policies about how domain names will be looked up and presented to users given the policies of the relevant DNS zone operators. None of these policies permit raw UTF-8. Since it is anticipated that DNS-SD when used with the DNS will be inside domain names beneath those kinds of "infrastructure" domains, the implications of IDNA2008 must be a consideration.

For further exploration of issues relating to encoding of domain names generally, the reader should consult [RFC6055].

3. Requirements for a profile for label interoperation

Any interoperability between DNS (including prevailing operational conventions) and other resolution technologies will require interoperability across the portions of a DNS-SD Service Instance Name that are implicated in regular DNS lookups. Only some portions are implicated. In any case, if a given portion is implicated, the profile will need to apply to all labels in that portion.

In addition, because DNS-SD Service Instance Names can be used in a domain name slot, care must be taken by DNS-SD-aware resolvers to handle the different portions as outlined here, so that DNS-SD portions that do not use IDNA2008 will not be treated as U-labels and will not accidentally undergo IDNA processing.

Because the profile will need to apply to names that might need to interoperate with names in the public DNS, and because other resolution mechanisms (such as mDNS) could permit labels that IDNA does not, the profile might reduce the labels that could be used with those other resolution mechanisms. One consequence of this is that some recommendations from [RFC6763] will not really be possible to implement using names subject to the profile. In particular, [RFC6763], section 4.1.3 recommends that labels always be stored and

communicated as UTF-8, even in the DNS. Because of the way the public DNS is currently operated (see Section 2), the advice to store and transmit labels as UTF-8 in the DNS is likely either to encounter problems or result in unnecessary traffic to the public DNS (or both). In particular, many labels in the <Domain> part of a Service Instance Name is unlikely to be found in its UTF-8 form in the public DNS tree for zones that are using IDNA2008. By contrast, for example, mDNS normally uses UTF-8.

U-labels cannot contain upper case letters. That restriction extends to ASCII-range upper case letters that work fine in LDH-labels. It may be confusing that the character "A" works in the DNS when none of the characters in the label has a diacritic, but does not work when there is such a diacritic in the label. Labels in mDNS names (or other resolution technologies) may contain upper case characters, so the profile will need either to restrict the use of upper case or come up with a reliable and predictable (to users) convention for case folding even in the presence of diacritics.

4. DNS-SD portions

Service Instance Names are made up of three portions.

4.1. The <Instance> Portion of the Service Instance Name

[RFC6763] is clear that the <Instance> portion of the Service Instance Name is intended for presentation to users, and therefore virtually any character is permitted in it. There are two ways that a profile might address this portion.

The first way would be to treat this portion as likely to be intercepted by system-wide IDNA-aware resolvers, or likely subject to strict IDNA conformance requirements for publication in the relevant zone. In this case, the portion would need to be made subject to the profile, thereby curtailing what characters may appear in this portion. This approach permits DNS-SD to use any standard system resolver but presents inconsistencies with the DNS-SD specification and with DNS-SD that is exclusively mDNS-based. Therefore, this strategy is rejected.

Instead, DNS-SD implementations can intercept the <Instance> portion of a Service Instance Name and ensure that those labels are never handed to IDNA-aware resolvers that might attempt to convert these labels into A-labels. Under this approach, the DNS-SD <Instance> portion works as it always does, but at the cost of using special resolution code built into the DNS-SD system. A practical consequence of this is that zone operators need to be prepared not to apply the LDH rule to all labels, and may need to make special

concessions to ensure that the <Instance> portion can contain spaces, upper and lower case, and any UTF-8 code point; or else to prepare a user interface to handle the exceptions that would otherwise be generated. Automatic conversion to A-labels is not acceptable.

4.2. The <Service> Portion of the Service Instance Name

DNS-SD includes a <Service> component in the Service Instance Name. This component is not really user-facing data, but is instead control data embedded in the Service Instance Name. This component includes so-called "underscore labels", which are labels prepended with U+005F (_). The underscore label convention was established by DNS SRV ([RFC2782]) for identifying metadata inside DNS names. A system-wide resolver (or DNS middlebox) that cannot handle underscore labels will not work with DNS-SD at all, so it is safe to suppose that such resolvers will not attempt to do special processing on these labels. Therefore, the <Service> portion of the Service Instance Name will not be subject to the profile. By the same token, it should be noted that underscore labels are never subject to IDNA processing (they're formally incompatible), and therefore concerns about IDNA are irrelevant for these labels.

4.3. The <Domain> Portion of the Service Instance Name

The <Domain> portion of the Service Instance Name forms an integral part of the owner name submitted for DNS resolution. A system-wide resolver that is IDNA2008-aware is likely to interpret labels with UTF-8 in the owner name as candidates for IDNA2008 processing. More important, operators of internationalized domain names will frequently publish such names in the DNS as A-labels; certainly, the top-most labels will always be A-labels. Therefore, these labels will need to be subject to the profile. DNS-SD implementations ought to identify the <Domain> portion of the Service Instance Name and treat it subject to IDNA2008 in case the domain is to be queried from the global DNS. In the event that the <Domain> portion of the Service Instance Name fails to resolve, it is acceptable to substitute labels with plain UTF-8, starting at the lowest label in the DNS tree and working toward the root. This approach differs from the rule for resolution published in [RFC6763], because it privileges IDNA2008-compatible labels over UTF-8 labels.

One might argue against this restriction on either of two grounds:

1. It is possible the names may be in the DNS in UTF-8, and RFC 6763 already specifies a fallback strategy of progressively attempting first the UTF-8 label lookup (it might not be a U-label) and then if possible the A-label lookup.

2. Zone administrators that wish to support DNS-SD can publish a UTF-8 version of the zone along side the A-label version of the zone.

The first of these is rejected because it represents a potentially significant increase in DNS lookup traffic for no value. It is possible for a DNS-SD application to identify the <Domain> portion of the Service Instance Name. The standard way to publish IDNs on the Internet uses IDNA. Therefore, additional lookups should not be encouraged. When [RFC6763] was published, the bulk of IDNs were lower in the tree. Now that there are internationalized labels in the root zone, it is desirable to minimize queries to the Internet infrastructure if they are sure to be answered in the negative.

The second reason depends on the idea that it is possible to maintain two names in sync with one another. This is not strictly speaking true, although in this case the domain operator could simply create a DNAME record [RFC6672] from the UTF-8 name to the IDNA2008 zone. This still, however, relies on being able to reach the (UTF-8) name in question, and it is unlikely that the UTF-8 version of the zone will be delegated from anywhere. Moreover, in many organizations the support for DNS-SD and the support for domain name delegations are not performed by the same department, and depending on a co-ordination between the two will make the system more fragile, or slower, or both.

Some resolvers -- particularly those that are used in mixed DNS and non-DNS environments -- may be aware of different operational conventions in different parts of the DNS tree. For example, it may be possible for implementations to use hints about the boundary of an organization's domain name infrastructure, in order to tell (for instance) that example.com. is part of the Example Organization while com. is a large delegation-centric zone on the public Internet. In such cases, the resolution system might reverse its preferences to prefer plain UTF-8 labels when resolving names below the boundary point in the DNS tree. The result would be that any lookup past the boundary point and closer to the root would use LDH-labels first, falling back to UTF-8 only after a failure; but a lookup below the boundary point would use UTF-8 labels first, and try other strategies only in case of negative answers. The mechanism to learn such a boundary is beyond the scope of this document.

5. Acknowledgements

The author gratefully acknowledges the insights of Joe Abley, Stuart Cheshire, Paul Hoffman, Kerry Lynn, and Dave Thaler. Kerry Lynn deserves special gratitude for his energy and persistence in pressing

unanswered questions. Doug Otis sent many comments about visual confusability.

6. IANA Considerations

This memo makes no requests of IANA.

7. Security Considerations

This memo presents some requirements for future development, but does not specify anything. It makes no additional security-specific requirements. Issues arising due to visual confusability of names apply to this case as well as to any other case of internationalized names, but interoperation between different resolution systems and conventions does not alter the severity of those issues.

8. Informative References

- [I-D.ietf-dnsop-dns-terminology]
Hoffman, P., Sullivan, A., and K. Fujiwara, "DNS Terminology", draft-ietf-dnsop-dns-terminology-05 (work in progress), September 2015.
- [RFC0952] Harrenstien, K., Stahl, M., and E. Feinler, "DoD Internet host table specification", RFC 952, October 1985.
- [RFC1034] Mockapetris, P., "Domain names - concepts and facilities", STD 13, RFC 1034, November 1987.
- [RFC1035] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, November 1987.
- [RFC2181] Elz, R. and R. Bush, "Clarifications to the DNS Specification", RFC 2181, July 1997.
- [RFC2782] Gulbrandsen, A., Vixie, P., and L. Esibov, "A DNS RR for specifying the location of services (DNS SRV)", RFC 2782, February 2000.
- [RFC5198] Klensin, J. and M. Padlipsky, "Unicode Format for Network Interchange", RFC 5198, March 2008.
- [RFC5890] Klensin, J., "Internationalized Domain Names for Applications (IDNA): Definitions and Document Framework", RFC 5890, August 2010.
- [RFC5891] Klensin, J., "Internationalized Domain Names in Applications (IDNA): Protocol", RFC 5891, August 2010.

- [RFC5892] Faltstrom, P., "The Unicode Code Points and Internationalized Domain Names for Applications (IDNA)", RFC 5892, August 2010.
- [RFC5893] Alvestrand, H. and C. Karp, "Right-to-Left Scripts for Internationalized Domain Names for Applications (IDNA)", RFC 5893, August 2010.
- [RFC5894] Klensin, J., "Internationalized Domain Names for Applications (IDNA): Background, Explanation, and Rationale", RFC 5894, August 2010.
- [RFC5895] Resnick, P. and P. Hoffman, "Mapping Characters for Internationalized Domain Names in Applications (IDNA) 2008", RFC 5895, September 2010.
- [RFC6055] Thaler, D., Klensin, J., and S. Cheshire, "IAB Thoughts on Encodings for Internationalized Domain Names", RFC 6055, February 2011.
- [RFC6672] Rose, S. and W. Wijngaards, "DNAME Redirection in the DNS", RFC 6672, June 2012.
- [RFC6762] Cheshire, S. and M. Krochmal, "Multicast DNS", RFC 6762, February 2013.
- [RFC6763] Cheshire, S. and M. Krochmal, "DNS-Based Service Discovery", RFC 6763, February 2013.

Appendix A. Change History

Note to RFC Editor: this section should be removed prior to publication.

A.1. draft-ietf-dnssd-mdns-dns-interop-02

- o Altered the title to make it more generic than mDNS.
- o Addressed issues raised by Dave Thaler in review on 2015-07-18.
- o Added a note to Section 7 about visual confusion. I don't know whether this will satisfy Doug Otis but it is the only thing I can see that could possibly be relevant.
- o Added discussion of finding "boundary" in Section 4.3.

A.2. draft-ietf-dnssd-mdns-dns-interop-01

Alter text to make clear that the main issue is the way the public DNS is currently administered, not system resolvers. I suppose this should have been clear before, but I didn't do that. Many thanks to Kerry Lynn for penetrating questions that illuminated what I'd left out.

A.3. draft-ietf-dnssd-mdns-dns-interop-00

1st WG version

Add text to make clear that fallback from A-label lookup to UTF-8 label lookup ok, per WG comments at IETF 91

A.4. draft-sullivan-dnssd-mdns-dns-interop-01

- o Decided which portions would be affected
- o Explained the difference in user interfaces between DNS-SD and usual DNS operation
- o Provided background on why the Domain portion should be treated differently

A.5. draft-sullivan-dnssd-mdns-dns-interop-00

Initial version.

Author's Address

Andrew Sullivan
Dyn
150 Dow St.
Manchester, NH 03101
U.S.A.

Email: asullivan@dyn.com

DNSSD
Internet-Draft
Intended status: Informational
Expires: July 7, 2017

A. Sullivan
Dyn
January 3, 2017

On Interoperation of Labels Among Conventional DNS and Other Resolution
Systems
draft-ietf-dnssd-mdns-dns-interop-04

Abstract

Despite its name, DNS-Based Service Discovery can use naming systems other than the Domain Name System when looking for services. Moreover, when it uses the DNS, DNS-Based Service Discovery uses the full capability of DNS, rather than using a subset of available octets. In order for DNS-SD to be used effectively in environments where multiple different name systems and conventions for their operation are in use, it is important to attend to differences in the underlying technology and operational environment. This memo presents an outline of the requirements for selection of labels for conventional DNS and other resolution systems when they are expected to interoperate in this manner.

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 July 7, 2017.

Copyright Notice

Copyright (c) 2017 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	2
1.1. Conventions and terms used in this document	3
2. Why there could be a problem at all	4
3. Requirements for a profile for label interoperation	5
4. DNS-SD portions	6
4.1. The <Instance> Portion of the Service	
Instance Name	6
4.2. The <Service> Portion of the Service	
Instance Name	7
4.3. The <Domain> Portion of the Service Instance	
Name	7
5. Acknowledgements	9
6. IANA Considerations	9
7. Security Considerations	9
8. Informative References	9
Appendix A. Change History	11
A.1. draft-ietf-dnssd-mdns-dns-interop-04	11
A.2. draft-ietf-dnssd-mdns-dns-interop-03	11
A.3. draft-ietf-dnssd-mdns-dns-interop-02	11
A.4. draft-ietf-dnssd-mdns-dns-interop-01	11
A.5. draft-ietf-dnssd-mdns-dns-interop-00	11
A.6. draft-sullivan-dnssd-mdns-dns-interop-01	12
A.7. draft-sullivan-dnssd-mdns-dns-interop-00	12
Author's Address	12

1. Introduction

DNS-Based Service Discovery (DNS-SD, [RFC6763]) specifies a mechanism for discovering services using queries to the Domain Name System (DNS, [RFC1034], [RFC1035]); and to any other system that uses domain names, such as Multicast DNS (mDNS, [RFC6762]). Many applications that use the DNS follow "Internet hostname" syntax [RFC0952] for labels -- the so-called LDH rule. That convention is the reason behind the development of Internationalized Domain Names for Applications (IDNA2008, [RFC5890], [RFC5891], [RFC5892], [RFC5893], [RFC5894], [RFC5895]). It is worth noting that the LDH rule is a convention, and not a rule of the DNS; this is made entirely plain by [RFC2181], section 11, and discussed further in [RFC6055], section 3.

Nevertheless, there is a widespread belief that in many circumstances domain names cannot be used in the DNS unless they cleave to the LDH rule.

At the same time, mDNS requires that labels be encoded in UTF-8, and permits a range of characters in labels that are not permitted by IDNA2008 or the LDH rule. For example, mDNS encourages the use of spaces and punctuation in mDNS names (see [RFC6763], section 4.1.3). It does not restrict which Unicode code points may be used in those labels, so long as the code points are UTF-8 in Net-Unicode [RFC5198] format.

Users and developers of applications are, of course, frequently unconcerned with (not to say oblivious to) the name-resolution system(s) in service at any given moment, and are inclined simply to use the same domain names in different contexts. As a result, the same domain name might be tried using different name resolution technologies. If a given name will not work across the various environments, then user expectations are likely to be best satisfied when at least some parts of the domain names to be queried are compatible with the rules and conventions for all the relevant technologies. Given the uses of DNS-SD, a choice for such compatibility likely lies with the application designer or service operator.

One approach to interoperability under these circumstances is to use a single operational convention (a "profile") for domain names under the different naming systems. This memo assumes such a use profile, and attempts to outline what is necessary to make it work without specifying any particular technology. It does assume, however, that the global DNS is eventually likely to be implicated. Given the general tendency of all resolution eventually to fall through to the DNS, that assumption does not seem controversial.

It is worth noting that users of DNS-SD do not use the service discovery names in the same way that users of other domain names might. In many cases, domain names can be entered as direct user input. But the service discovery context generally assumes users are picking a service from a list. As a result, the sorts of application considerations that are appropriate to the general-purpose DNS name, and that resulted in the A-label/U-label split (see below) in IDNA2008, are not entirely the right approach for DNS-SD.

1.1. Conventions and terms used in this document

Wherever appropriate, this memo uses the terminology defined in Section 2 of [RFC5890]. In particular, the reader is assumed to be familiar with the terms "U-label", "LDH label", and "A-label" from

that document. Similarly, the reader is assumed to be familiar with the U+NNNN notation for Unicode code points used in [RFC5890] and other documents dealing with Unicode code points. In the interests of brevity and consistency, the definitions are not repeated here.

Sometimes this memo refers to names in the DNS as though the LDH rule and IDNA2008 are strict requirements. They are not. DNS labels are, in principle, just collections of octets, and therefore in principle the LDH rule is not a constraint. In practice, applications sometimes intercept labels that do not conform to the LDH rule and apply IDNA and other transformations.

The DNS, perhaps unfortunately, has produced its own jargon. Unfamiliar DNS-related terms in this memo should be found in [RFC7719].

The term "owner name" (common to the DNS vernacular; see above) is used here to apply not just to the domain names to be looked up in the DNS, but to any name that might be looked up either in the DNS or using another technology. It therefore includes names that might not actually exist anywhere. In addition, what follows depends on the idea that not every domain name will be looked up in the DNS. For instance, names ending in "local." (in the presentation format) are not ordinarily looked up using DNS, but instead looked up using mDNS.

DNS-SD specifies three portions of the owner name for a DNS-SD resource record. These are the <Instance> portion, the <Service> portion, and the <Domain> portion. The owner name made of these three parts is called the Service Instance Name. It is worth observing that a portion may be more than one label long. See [RFC6763], section 4.1. Further discussion of the parts is found in Section 4.

Throughout this memo, mDNS is used liberally as the alternative resolution mechanism to DNS. This is for convenience rather than rigour: any alternative name resolution to DNS could present the same friction with the prevailing operational conventions of the global DNS. It so happens that mDNS is the overwhelmingly successful alternative as of this writing, so it is used in order to make the issues plainer to the reader. Other alternative resolution mechanisms may in general be read wherever mDNS appears in the text, except where details of the mDNS specification appear.

2. Why there could be a problem at all

One might reasonably wonder why there is a problem to be solved at all. After all, DNS labels permit any octet whatsoever, and anything

that can be useful with DNS-SD cannot use any names that are outside the protocol strictures of the DNS.

The reason for the trouble is twofold. First, and least troublesome, is the possibility of resolvers that are attempting to offer IDNA service system-wide. Given the design of IDNA2008, it is reasonable to suppose that on some systems high-level name resolution libraries will perform the U-label/A-label transformation automatically, saving applications from these details. But system-level services do not always have available to them the resolution context, and may apply the transformation in a way that foils rather than helps the application. Of course, if this were the main problem, it would presumably be self-correcting; for the right answer would be, "Don't use those libraries for DNS-SD," and DNS-SD would not work reliably in cases where such libraries were in use. This would be unfortunate; but given that DNS-SD in Internet contexts is as of this writing not in ubiquitous use, it should not represent a fatal issue.

The greater problem is that the "infrastructure" types of DNS service -- the root zone, the top-level domains, and so on -- have embraced IDNA and refuse registration of raw UTF-8 into their zones. As of this writing there is (perhaps unfortunately) no reliable way to discover where these sorts of DNS services end. Nevertheless, some client programs (notably web browsers) have adopted a number of different policies about how domain names will be looked up and presented to users given the policies of the relevant DNS zone operators. None of these policies permit raw UTF-8. Since it is anticipated that DNS-SD when used with the DNS will be inside domain names beneath those kinds of "infrastructure" domains, the implications of IDNA2008 must be a consideration.

For further exploration of issues relating to encoding of domain names generally, the reader should consult [RFC6055].

3. Requirements for a profile for label interoperation

Any interoperability between DNS (including prevailing operational conventions) and other resolution technologies will require interoperability across the portions of a DNS-SD Service Instance Name that are implicated in regular DNS lookups. Only some portions are implicated. In any case, if a given portion is implicated, the profile will need to apply to all labels in that portion.

In addition, because DNS-SD Service Instance Names can be used in a domain name slot, care must be taken by DNS-SD-aware resolvers to handle the different portions as outlined here, so that DNS-SD portions that do not use IDNA2008 will not be treated as U-labels and will not accidentally undergo IDNA processing.

Because the profile will apply to names that might appear in the public DNS, and because other resolution mechanisms (such as mDNS) could permit labels that IDNA does not, the profile might reduce the labels that could be used with those other resolution mechanisms. One consequence of this is that some recommendations from [RFC6763] will not really be possible to implement using names subject to the profile. In particular, [RFC6763], section 4.1.3 recommends that labels always be stored and communicated as UTF-8, even in the DNS. Because of the way the public DNS is currently operated (see Section 2), the advice to store and transmit labels as UTF-8 in the DNS is likely either to encounter problems, or to result in unnecessary traffic to the public DNS, or both. In particular, many labels in the <Domain> part of a Service Instance Name are unlikely to be found in the UTF-8 form in the public DNS tree for zones that are using IDNA2008. By contrast, for example, mDNS normally uses UTF-8.

U-labels cannot contain upper case letters (see [RFC5894], sections 3.1.3 and 4.2). That restriction extends to ASCII-range upper case letters that work fine in LDH-labels. It may be confusing that the character "A" works in the DNS when none of the characters in the label has a diacritic, but does not work when there is such a diacritic in the label. Labels in mDNS names (or other resolution technologies) may contain upper case characters, so the profile will need either to restrict the use of upper case or come up with a convention for case folding (even in the presence of diacritics) that is reliable and predictable to users.

4. DNS-SD portions

Service Instance Names are made up of three portions.

4.1. The <Instance> Portion of the Service Instance Name

[RFC6763] is clear that the <Instance> portion of the Service Instance Name is intended for presentation to users, and therefore virtually any character is permitted in it. There are two ways that a profile might address this portion.

The first way would be to treat this portion as likely to be intercepted by system-wide IDNA-aware (but otherwise context-unaware) resolvers, or likely subject to strict IDNA conformance requirements for publication in the relevant zone. In this case, the portion would need to be made subject to the profile, thereby curtailing what characters may appear in this portion. This approach permits DNS-SD to use any standard system resolver but presents inconsistencies with the DNS-SD specification and with DNS-SD use that is exclusively mDNS-based. Therefore, this strategy is rejected.

Instead, DNS-SD implementations can intercept the <Instance> portion of a Service Instance Name and ensure that those labels are never handed to IDNA-aware resolvers that might attempt to convert these labels into A-labels. Under this approach, the DNS-SD <Instance> portion works as it always does, but at the cost of using special resolution code built into the DNS-SD system. A practical consequence of this is that zone operators need to be prepared not to apply the LDH rule to all labels, and may need to make special concessions to ensure that the <Instance> portion can contain spaces, upper and lower case, and any UTF-8 code point; or else to prepare a user interface to handle the exceptions that would otherwise be generated. Automatic conversion to A-labels is not acceptable.

It is worth noting that this advice is not actually compatible with advice in [RFC6055], section 4. That section appears to assume that names are not really composed of subsections, but because [RFC6763] specifies portions of names, the advice in this memo is to follow the advice of [RFC6055] according to the portion of the domain name, rather than for the whole domain name. As a practical matter, this likely means special-purpose name resolution software for DNS-SD.

4.2. The <Service> Portion of the Service Instance Name

DNS-SD includes a <Service> component in the Service Instance Name. This component is not really user-facing data, but is instead control data embedded in the Service Instance Name. This component includes so-called "underscore labels", which are labels prepended with U+005F (_). The underscore label convention was established by DNS SRV ([RFC2782]) for identifying metadata inside DNS names. A system-wide resolver (or DNS middlebox) that cannot handle underscore labels will not work with DNS-SD at all, so it is safe to suppose that such resolvers will not attempt to do special processing on these labels. Therefore, the <Service> portion of the Service Instance Name will not be subject to the profile. By the same token, underscore labels are never subject to IDNA processing (they are formally incompatible), and therefore concerns about IDNA are irrelevant for these labels.

4.3. The <Domain> Portion of the Service Instance Name

The <Domain> portion of the Service Instance Name forms an integral part of the owner name submitted for DNS resolution. A system-wide resolver that is IDNA2008-aware is likely to interpret labels with UTF-8 in the owner name as candidates for IDNA2008 processing. More important, operators of internationalized domain names will frequently publish such names in the public DNS as A-labels; certainly, the top-most labels will always be A-labels. Therefore, these labels will need to be subject to the profile. DNS-SD

implementations ought somehow to identify the <Domain> portion of the Service Instance Name and treat it subject to IDNA2008 in case the domain is to be queried from the global DNS. In the event that the <Domain> portion of the Service Instance Name fails to resolve, it is acceptable to substitute labels with plain UTF-8, starting at the lowest label in the DNS tree and working toward the root. This approach differs from the rule for resolution published in [RFC6763], because it privileges IDNA2008-compatible labels over UTF-8 labels. There is more than one way to achieve such a result, but in terms of predictability it is probably best if the lowest-level resolution component is able to learn the correct resolution context, so that it can perform the correct transformations on the various domain portions.

One might argue against the above restriction on either of two grounds:

1. It is possible the names may be in the DNS in UTF-8, and RFC 6763 already specifies a fallback strategy of progressively attempting first the UTF-8 label lookup (it might not be a U-label) and then if possible the A-label lookup.
2. Zone administrators that wish to support DNS-SD can publish a UTF-8 version of the zone along side the A-label version of the zone.

The first of these is rejected because it represents a potentially significant increase in DNS lookup traffic for no value. It is possible for a DNS-SD application to identify the <Domain> portion of the Service Instance Name. The standard way to publish IDNs on the Internet uses IDNA. Therefore, additional lookups should not be encouraged. When [RFC6763] was published, the bulk of IDNs were lower in the tree. Now that there are internationalized labels in the root zone, it is desirable to minimize queries to the Internet infrastructure if they are sure to be answered in the negative.

The second reason depends on the idea that it is possible to maintain two names in sync with one another. This is not strictly speaking true, although in this case the domain operator could simply create a DNAME record [RFC6672] from the UTF-8 name to the IDNA2008 zone. This still, however, relies on being able to reach the (UTF-8) name in question, and it is unlikely that the UTF-8 version of the zone will be delegated from anywhere. Moreover, in many organizations the support for DNS-SD and the support for domain name delegations are not performed by the same department, and depending on a co-ordination between the two will make the system more fragile, or slower, or both.

Some resolvers -- particularly those that are used in mixed DNS and non-DNS environments -- may be aware of different operational conventions in different parts of the DNS tree. For example, it may be possible for implementations to use hints about the boundary of an organization's domain name infrastructure, in order to tell (for instance) that example.com. is part of the Example Organization while com. is a large delegation-centric zone on the public Internet. In such cases, the resolution system might reverse its preferences to prefer plain UTF-8 labels when resolving names below the boundary point in the DNS tree. The result would be that any lookup past the boundary point and closer to the root would use LDH-labels first, falling back to UTF-8 only after a failure; but a lookup below the boundary point would use UTF-8 labels first, and try other strategies only in case of negative answers. The mechanism to learn such a boundary is beyond the scope of this document.

5. Acknowledgements

The author gratefully acknowledges the insights of Joe Abley, Stuart Cheshire, Paul Hoffman, Kerry Lynn, and Dave Thaler. Kerry Lynn deserves special gratitude for his energy and persistence in pressing unanswered questions. Doug Otis sent many comments about visual confusability.

6. IANA Considerations

This memo makes no requests of IANA.

7. Security Considerations

This memo presents some requirements for future development, but does not specify anything. It makes no additional security-specific requirements. Issues arising due to visual confusability of names apply to this case as well as to any other case of internationalized names, but interoperation between different resolution systems and conventions does not alter the severity of those issues.

8. Informative References

- [RFC0952] Harrenstien, K., Stahl, M., and E. Feinler, "DoD Internet host table specification", RFC 952, October 1985.
- [RFC1034] Mockapetris, P., "Domain names - concepts and facilities", STD 13, RFC 1034, November 1987.
- [RFC1035] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, November 1987.

- [RFC2181] Elz, R. and R. Bush, "Clarifications to the DNS Specification", RFC 2181, July 1997.
- [RFC2782] Gulbrandsen, A., Vixie, P., and L. Esibov, "A DNS RR for specifying the location of services (DNS SRV)", RFC 2782, February 2000.
- [RFC5198] Klensin, J. and M. Padlipsky, "Unicode Format for Network Interchange", RFC 5198, March 2008.
- [RFC5890] Klensin, J., "Internationalized Domain Names for Applications (IDNA): Definitions and Document Framework", RFC 5890, August 2010.
- [RFC5891] Klensin, J., "Internationalized Domain Names in Applications (IDNA): Protocol", RFC 5891, August 2010.
- [RFC5892] Faltstrom, P., "The Unicode Code Points and Internationalized Domain Names for Applications (IDNA)", RFC 5892, August 2010.
- [RFC5893] Alvestrand, H. and C. Karp, "Right-to-Left Scripts for Internationalized Domain Names for Applications (IDNA)", RFC 5893, August 2010.
- [RFC5894] Klensin, J., "Internationalized Domain Names for Applications (IDNA): Background, Explanation, and Rationale", RFC 5894, August 2010.
- [RFC5895] Resnick, P. and P. Hoffman, "Mapping Characters for Internationalized Domain Names in Applications (IDNA) 2008", RFC 5895, September 2010.
- [RFC6055] Thaler, D., Klensin, J., and S. Cheshire, "IAB Thoughts on Encodings for Internationalized Domain Names", RFC 6055, February 2011.
- [RFC6672] Rose, S. and W. Wijngaards, "DNAME Redirection in the DNS", RFC 6672, June 2012.
- [RFC6762] Cheshire, S. and M. Krochmal, "Multicast DNS", RFC 6762, February 2013.
- [RFC6763] Cheshire, S. and M. Krochmal, "DNS-Based Service Discovery", RFC 6763, February 2013.

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

Appendix A. Change History

Note to RFC Editor: this section should be removed prior to publication.

A.1. draft-ietf-dnssd-mdns-dns-interop-04

- o Unchanged content to reset the I-D repo timer.

A.2. draft-ietf-dnssd-mdns-dns-interop-03

- o Additional alteration of title
- o Attempt to address WGLC comments from Dave Thaler (2016-04-02)

A.3. draft-ietf-dnssd-mdns-dns-interop-02

- o Altered the title to make it more generic than mDNS.
- o Addressed issues raised by Dave Thaler in review on 2015-07-18.
- o Added a note to Section 7 about visual confusion. I don't know whether this will satisfy Doug Otis but it is the only thing I can see that could possibly be relevant.
- o Added discussion of finding "boundary" in Section 4.3.

A.4. draft-ietf-dnssd-mdns-dns-interop-01

Alter text to make clear that the main issue is the way the public DNS is currently administered, not system resolvers. I suppose this should have been clear before, but I didn't do that. Many thanks to Kerry Lynn for penetrating questions that illuminated what I'd left out.

A.5. draft-ietf-dnssd-mdns-dns-interop-00

1st WG version

Add text to make clear that fallback from A-label lookup to UTF-8 label lookup ok, per WG comments at IETF 91

A.6. draft-sullivan-dnssd-mdns-dns-interop-01

- o Decided which portions would be affected
- o Explained the difference in user interfaces between DNS-SD and usual DNS operation
- o Provided background on why the Domain portion should be treated differently

A.7. draft-sullivan-dnssd-mdns-dns-interop-00

Initial version.

Author's Address

Andrew Sullivan
Dyn
150 Dow St.
Manchester, NH 03101
U.S.A.

Email: asullivan@dyn.com

Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: September 22, 2016

T. Pusateri
Seeking affiliation
S. Cheshire
Apple Inc.
March 21, 2016

DNS Push Notifications
draft-ietf-dnssd-push-06

Abstract

The Domain Name System (DNS) was designed to return matching records efficiently for queries for data that is relatively static. When those records change frequently, DNS is still efficient at returning the updated results when polled. But there exists no mechanism for a client to be asynchronously notified when these changes occur. This document defines a mechanism for a client to be notified of such changes to DNS records, called DNS Push Notifications.

Status of This Memo

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 22, 2016.

Copyright Notice

Copyright (c) 2016 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	2
1.1. Requirements Language	3
2. Motivation	3
3. Overview	5
4. Transport	6
4.1. Client-Initiated Termination	7
4.2. Server-Initiated Termination	7
5. State Considerations	9
6. Protocol Operation	10
6.1. Discovery	11
6.2. DNS Push Notification SUBSCRIBE	13
6.3. DNS Push Notification UNSUBSCRIBE	18
6.4. DNS Push Notification Update Messages	19
6.5. DNS RECONFIRM	22
6.6. DNS Push Notification Termination Message	24
7. Security Considerations	26
7.1. Security Services	26
7.2. TLS Name Authentication	27
7.3. TLS Compression	27
7.4. TLS Session Resumption	27
8. IANA Considerations	27
9. Acknowledgements	28
10. References	28
10.1. Normative References	28
10.2. Informative References	30
Authors' Addresses	31

1. Introduction

DNS records may be updated using DNS Update [RFC2136]. Other mechanisms such as a Hybrid Proxy [I-D.ietf-dnssd-hybrid] can also generate changes to a DNS zone. This document specifies a protocol for Unicast DNS clients to subscribe to receive asynchronous notifications of changes to RRsets of interest. It is immediately relevant in the case of DNS Service Discovery [RFC6763] but is not limited to that use case, and provides a general DNS mechanism for DNS record change notifications. Familiarity with the DNS protocol and DNS packet formats is assumed [RFC1034] [RFC1035] [RFC6195].

1.1. Requirements Language

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

2. Motivation

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

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

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

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

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

3. Overview

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

This specification adopts a simplified subset of these existing syntax and semantics, and uses them for DNS Push Notification messages going in the opposite direction, from server to client, to communicate changes to a zone. The client subscribes for Push Notifications by connecting to the server and sending DNS message(s) indicating the RRSet(s) of interest. When the client loses interest in updates to these records, it unsubscribes.

The DNS Push Notification server for a zone is any server capable of generating the correct change notifications for a name. It may be a master, slave, or stealth name server [RFC1996]. Consequently, the "_dns-push-tls._tcp.<zone>" SRV record for a zone MAY reference the same target host and port as that zone's "_dns-update-tls._tcp.<zone>" SRV record. When the same target host and port is offered for both DNS Updates and DNS Push Notifications, a client MAY use a single TCP connection to that server for both DNS Updates and DNS Push Notification Queries.

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

DNS Updates and DNS Push Notifications may be handled on different ports on the same target host, in which case they are not considered to be the "same server" for the purposes of this specification, and communications with these two ports are handled independently.

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

DNS Push Notification clients are NOT required to implement DNS Update Prerequisite processing. Prerequisites are used to perform tentative atomic test-and-set type operations when a client updates records on a server, and that concept has no applicability when it comes to an authoritative server informing a client of changes to DNS records.

This DNS Push Notification specification includes support for DNS classes, for completeness. However, in practice, it is anticipated that for the foreseeable future the only DNS class in use will be DNS

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

4. Transport

Implementations of DNS Update [RFC2136] MAY use either User Datagram Protocol (UDP) [RFC0768] or Transmission Control Protocol (TCP) [RFC0793] as the transport protocol, in keeping with the historical precedent that DNS queries must first be sent over UDP [RFC1123]. This requirement to use UDP has subsequently been relaxed [RFC5966][I-D.ietf-dnsop-5966bis].

In keeping with the more recent precedent, DNS Push Notification is defined only for TCP. DNS Push Notification clients MUST use TLS over TCP.

Connection setup over TCP ensures return reachability and alleviates concerns of state overload at the server through anonymous subscriptions. All subscribers are guaranteed to be reachable by the server by virtue of the TCP three-way handshake. Because TCP SYN flooding attacks are possible with any protocol over TCP, implementers are encouraged to use industry best practices to guard against such attacks [IPJ.9-4-TCP SYN] [RFC4953].

Transport Layer Security (TLS) [RFC5246] is well understood and deployed across many protocols running over TCP. It is designed to prevent eavesdropping, tampering, or message forgery. TLS is REQUIRED for every connection between a client subscriber and server in this protocol specification. Additional security measures such as client authentication during TLS negotiation MAY also be employed to increase the trust relationship between client and server. Additional authentication of the SRV target using DNSSEC verification and DANE TLSA records [RFC7673] is strongly encouraged. See below in Section 7.2 for details.

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

4.1. Client-Initiated Termination

An individual subscription is terminated by sending an UNSUBSCRIBE message for that specific subscription, or all subscriptions can be cancelled at once by the client closing the connection with a TCP RST. When a client terminates an individual subscription (via UNSUBSCRIBE) or all subscriptions on that connection (by closing the connection) it is signalling to the server that it is longer interested in receiving those particular updates. It is informing the server that the server may release any state information it has been keeping with regards to these particular subscriptions.

After terminating its last subscription on a connection via UNSUBSCRIBE, a client MAY close the connection immediately with a TCP FIN, or it may keep it open if it anticipates performing further operations on that connection in the future. If a client wishes to keep an idle connection open, it MUST meet its keepalive obligations [I-D.ietf-dnsop-edns-tcp-keepalive] or the server is entitled to close the connection (see below).

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

4.2. Server-Initiated Termination

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

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

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

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

If a client disconnects from the network abruptly, without closing its connection, the server learns of this after failing to receive further traffic from that client. If no requests, responses, update messages or keepalive traffic occurs on a connection for 1.5 times the idle timeout, then this indicates that the client is probably no longer on the network, and the server **SHOULD** abort the connection with a TCP RST.

[We need to discuss the nature of "the required keepalives". Are they TCP-layer keepalives? DNS-layer keepalives? There is currently no DNS-layer keepalive or 'no-op' operation defined. What would that operation be? A DNS QUERY containing zero questions? A DNS SUBSCRIBE containing zero questions? An "empty" DNS message over the TCP connection (just a pair of zero bytes, signifying a zero-length message)? One benefit of TCP-layer keepalives is that they transmit fewer bytes, and involve less software overhead for processing those bytes. Another benefit is that it is more feasible to implement these in networking offload hardware, which can allow devices to meet their TCP keepalive obligations while sleeping. This is particularly important for battery-powered devices like mobile phones and tablets. On the other hand, using TCP-layer keepalives requires an API for a client to tell the networking stack at what frequency to perform TCP-layer keepalives, and an API for a server to request the networking stack to inform it when TCP-layer keepalives are not received by the required deadline. TCP-layer keepalives also only verify liveness of the remote networking stack, whereas DNS-layer keepalives provide higher assurance of liveness of the remote server application software -- though this a limited benefit, since there is no reason to expect that DNS Push Notification server software will routinely become wedged and unresponsive.]

After sending an error response to a client, the server **MAY** close the connection with a TCP FIN.

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

Apart from the cases described above, a server **MUST NOT** close a connection with a DNS Push Notification client, except in

extraordinary error conditions. Closing the connection is the client's responsibility, to be done at the client's discretion, when it so chooses. A DNS Push Notification server only closes a DNS Push Notification connection under exceptional circumstances, such as when the server application software or underlying operating system is restarting, the server application terminated unexpectedly (perhaps due to a bug that makes it crash), or the server is undergoing maintenance procedures. When possible, a DNS Push Notification server SHOULD send a Termination Message (Section 6.6) informing the client of the reason for the connection being closed.

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

5. State Considerations

Each DNS Push Notification server is capable of handling some finite number of Push Notification subscriptions. This number will vary from server to server and is based on physical machine characteristics, network bandwidth, and operating system resource allocation. After a client establishes a connection to a DNS server, each record subscription is individually accepted or rejected. Servers may employ various techniques to limit subscriptions to a manageable level. Correspondingly, the client is free to establish simultaneous connections to alternate DNS servers that support DNS Push Notifications for the zone and distribute record subscriptions at its discretion. In this way, both clients and servers can react to resource constraints. Token bucket rate limiting schemes are also effective in providing fairness by a server across numerous client requests.

6. Protocol Operation

A DNS Push Notification exchange begins with the client discovering the appropriate server, and then making a TLS/TCP connection to it. The client may then add and remove Push Notification subscriptions over this connection. In accordance with the current set of active subscriptions the server sends relevant asynchronous Push Notifications to the client. Note that a client **MUST** be prepared to receive (and silently ignore) Push Notifications for subscriptions it has previously removed, since there is no way to prevent the situation where a Push Notification is in flight from server to client while the client's UNSUBSCRIBE message cancelling that subscription is simultaneously in flight from client to server.

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

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

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

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

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

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

6.1. Discovery

The first step in DNS Push Notification subscription is to discover an appropriate DNS server that supports DNS Push Notifications for the desired zone. The client **MUST** also determine which TCP port on the server is listening for connections, which need not be (and often is not) the typical TCP port 53 used for conventional DNS.

1. The client begins the discovery by sending a DNS query to the local resolver with record type SOA [RFC1035] for the name of the record it wishes to subscribe.
2. If the SOA record exists, it **MUST** be returned in the Answer Section of the reply. If not, the local resolver **SHOULD** include the SOA record for the zone of the requested name in the Authority Section.
3. If no SOA record is returned, the client then strips off the leading label from the requested name. If the resulting name has at least one label in it, the client sends a new SOA query and processing continues at step 2 above. If the resulting name is empty (the root label) then this is a network configuration error and the client gives up. The client **MAY** retry the operation at a later time.
4. Once the SOA is known, the client sends a DNS query with type SRV [RFC2782] for the record name "_dns-push-tls._tcp.<zone>", where <zone> is the owner name of the discovered SOA record.
5. If the zone in question does not offer DNS Push Notifications then SRV record **MUST NOT** exist and the SRV query will return a negative answer.

6. If the zone in question is set up to offer DNS Push Notifications then this SRV record MUST exist. The SRV "target" contains the name of the server providing DNS Push Notifications for the zone. The port number on which to contact the server is in the SRV record "port" field. The address(es) of the target host MAY be included in the Additional Section, however, the address records SHOULD be authenticated before use as described below in Section 7.2 [RFC7673].
7. More than one SRV record may be returned. In this case, the "priority" and "weight" values in the returned SRV records are used to determine the order in which to contact the servers for subscription requests. As described in the SRV specification [RFC2782], the server with the lowest "priority" is first contacted. If more than one server has the same "priority", the "weight" indicates the weighted probability that the client should contact that server. Higher weights have higher probabilities of being selected. If a server is not reachable or is not willing to accept a subscription request, then a subsequent server is to be contacted.

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

6.2. DNS Push Notification SUBSCRIBE

A DNS Push Notification client indicates its desire to receive DNS Push Notifications for a given domain name by sending a SUBSCRIBE request over the established TCP connection to the server. A SUBSCRIBE request is formatted identically to a conventional DNS QUERY request [RFC1035], except that the opcode is SUBSCRIBE (6) instead of QUERY (0). If neither QTYPE nor QCLASS are ANY (255) then this is a specific subscription to changes for the given name, type and class. If one or both of QTYPE or QCLASS are ANY (255) then this subscription matches any type and/or any class, as appropriate.

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

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

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

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

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

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

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

SUBSCRIBE requests on a given connection **MUST** be unique. A client **MUST NOT** send a SUBSCRIBE message that duplicates the name, type and class of an existing active subscription on that TLS/TCP connection. For the purpose of this matching, the established DNS case-insensitivity for US-ASCII letters applies (e.g., "foo.com" and "Foo.com" are the same). If a server receives such a duplicate SUBSCRIBE message this is an error and the server **MUST** immediately close the TCP connection.

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

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

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

Since all SUBSCRIBE operations are implicitly long-lived operations, the server MUST interpret a SUBSCRIBE request as if it contained an EDNS(0) TCP Keepalive option [I-D.ietf-dnsop-edns-tcp-keepalive]. A client MUST NOT include an actual EDNS(0) TCP Keepalive option in the request, since it is automatic, and implied by the semantics of SUBSCRIBE. If a server receives a SUBSCRIBE request that does contain an actual EDNS(0) TCP Keepalive option this is an error and the server MUST immediately close the TCP connection.

A SUBSCRIBE operation MAY include an explicit EDNS(0) [RFC6891] OPT record where necessary to carry additional information.

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

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

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

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

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

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

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

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

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

Mnemonic	Value	Description
NOERROR	0	SUBSCRIBE successful.
FORMERR	1	Server failed to process request due to a malformed request.
SERVFAIL	2	Server failed to process request due to resource exhaustion.
NXDOMAIN	3	NOT APPLICABLE. DNS Push Notification MUST NOT return NXDOMAIN errors in response to SUBSCRIBE requests.
NOTIMP	4	Server does not implement DNS Push Notifications.
REFUSED	5	Server refuses to process request for policy or security reasons.
NOTAUTH	9	Server is not authoritative for the requested name.

SUBSCRIBE Response codes

This document specifies only these RCODE values for SUBSCRIBE Responses. Servers sending SUBSCRIBE Responses SHOULD use one of these values. However, future circumstances may create situations where other RCODE values are appropriate in SUBSCRIBE Responses, so clients MUST be prepared to accept SUBSCRIBE Responses with any RCODE value.

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

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

In an error response, with nonzero RCODE, the server MUST contain an EDNS(0) TCP Keepalive option specifying the delay before the client tries again:

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

For RCODE = 2 (SERVFAIL), which occurs due to resource exhaustion, the delay should be chosen according to the level of server overload and the anticipated duration of that overload. By default, a value of one minute is RECOMMENDED.

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

For RCODE = 5 (REFUSED), which occurs on a server that implements DNS Push Notifications, but is currently configured to disallow DNS Push Notifications, the retry delay may be any value selected by the implementer and/or configured by the operator. This is a misconfiguration, since this server is listed in a "_dns-push-tls._tcp.<zone>" SRV record, but the server itself is not currently configured to support DNS Push Notifications. Since it is possible that the misconfiguration may be repaired at any time, the retry delay should not be set too high. By default, a value of 5 minutes is RECOMMENDED.

For RCODE = 9 (NOTAUTH), which occurs on a server that implements DNS Push Notifications, but is not configured to be authoritative for the requested name, the retry delay may be any value selected by the implementer and/or configured by the operator. This is a misconfiguration, since this server is listed in a "_dns-push-tls._tcp.<zone>" SRV record, but the server itself is not currently configured to support DNS Push Notifications for that zone. Since it is possible that the misconfiguration may be repaired at any time, the retry delay should not be set too high. By default, a value of 5 minutes is RECOMMENDED.

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

After sending an error response the server MAY close the TCP connection with a FIN, or MAY allow it to remain open. Clients MUST correctly handle both cases.

6.3. DNS Push Notification UNSUBSCRIBE

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

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

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

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

6.4. DNS Push Notification Update Messages

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

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

The following figure shows the existing DNS Update header format:

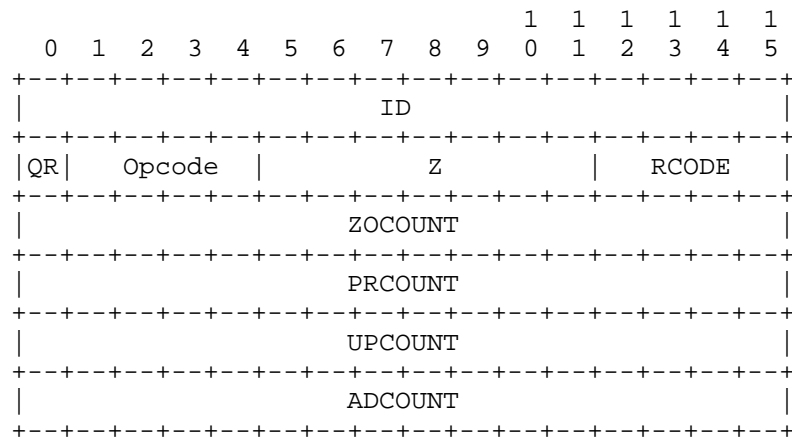


Figure 1

For DNS Push Notifications the following rules apply:

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

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

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

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

UPCOUNT specifies the number of records in the Update Section.

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

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

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

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

Delete an individual RR from a name:
TTL=0, CLASS=NONE;
TYPE, RDLLENGTH and RDATA specifies the RR being deleted.

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

When processing the records received in a Push Notification Update Message, the receiving client MUST validate that the records being added or deleted correspond with at least one currently active subscription on that connection. Specifically, the record name MUST match the name given in the SUBSCRIBE request, subject to the usual established DNS case-insensitivity for US-ASCII letters. If the QTYPE in the SUBSCRIBE request was not ANY (255) then the TYPE of the record must match the QTYPE given in the SUBSCRIBE request. If the QCLASS in the SUBSCRIBE request was not ANY (255) then the CLASS of the record must match the QCLASS given in the SUBSCRIBE request. If a matching active subscription on that connection is not found, then that individual record addition/deletion is silently ignored. Processing of other additions and deletions in this message is not affected. The TCP connection is not closed. This is to allow for the race condition where a client sends an outbound UNSUBSCRIBE while inbound Push Notification Updates for that subscription from the server are still in flight.

In the case where a single change affects more than one active subscription, only one update is sent. For example, an update adding a given record may match both a SUBSCRIBE request with the same QTYPE and a different SUBSCRIBE request with QTYPE=ANY. It is not the case that two updates are sent because the new record matches two active subscriptions.

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

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

Push Notification Update Messages MAY contain an EDNS(0) TCP Keepalive option [I-D.ietf-dnsop-edns-tcp-keepalive] if the idle timeout has changed since the last time the server sent an EDNS(0) TCP Keepalive option on this connection.

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

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

The TTL of an added record is stored by the client and decremented as time passes, with the caveat that for as long as a relevant subscription is active, the TTL does not decrement below 1 second. For as long as a relevant subscription remains active, the client SHOULD assume that when a record goes away the server will notify it of that fact. Consequently, a client does not have to poll to verify that the record is still there. Once a subscription is cancelled (individually, or as a result of the TCP connection being closed) record aging resumes and records are removed from the local cache when their TTL reaches zero.

6.5. DNS RECONFIRM

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

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

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

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

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

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

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

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

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

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

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

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

6.6. DNS Push Notification Termination Message

If a server is low on resources it MAY simply terminate a client connection with a TCP RST. However, the likely behaviour of the client may be simply to reconnect immediately, putting more burden on the server. Therefore, a server SHOULD instead choose to shed client load by (a) sending a DNS Push Notification Termination Message and then (b) immediately closing the client connection with a TCP FIN instead of RST, thereby facilitating reliable delivery of the Termination Message.

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

The following figure shows the existing DNS Update header format:

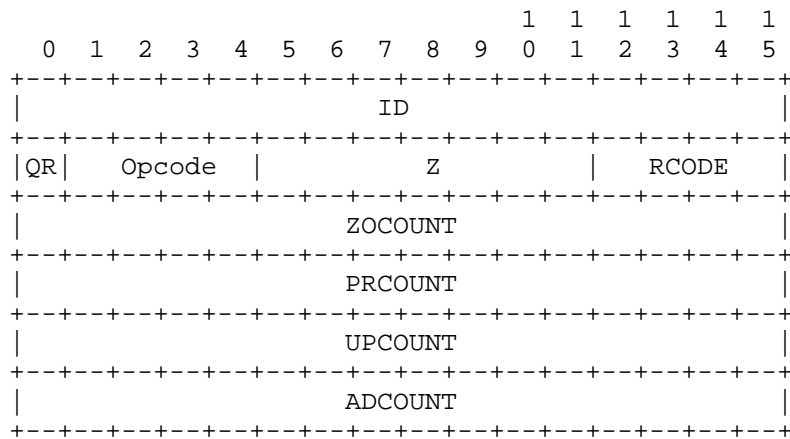


Figure 2

For Termination Messages the following rules apply:

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

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

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

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

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

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

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

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

Termination Response codes

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

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

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

In the case where a server is terminating a large number of connections at once (e.g., if the system is restarting) and the

server doesn't want to be inundated with a flood of simultaneous retries, it SHOULD send different EDNS(0) TCP Keepalive values to each client. These adjustments MAY be selected randomly, pseudorandomly, or deterministically (e.g., incrementing the time value by one for each successive client, yielding a post-restart reconnection rate of ten clients per second).

7. Security Considerations

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

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

Deployment recommendations on the appropriate key lengths and cypher suites are beyond the scope of this document. Please refer to TLS Recommendations [RFC7525] for the best current practices. Keep in mind that best practices only exist for a snapshot in time and recommendations will continue to change. Updated versions or errata may exist for these recommendations.

7.2. TLS Name Authentication

As described in Section 6.1, the client discovers the DNS Push Notification server using an SRV lookup for the record name "_dns-push-tls._tcp.<zone>". The server connection endpoint SHOULD then be authenticated using DANE TLSA records for the associated SRV record. This associates the target's name and port number with a trusted TLS certificate [RFC7673]. This procedure uses the TLS 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.

7.3. TLS Compression

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

7.4. TLS Session Resumption

TLS Session Resumption is permissible on DNS Push Notification servers. The server may keep TLS state with Session IDs [RFC5246] or operate in stateless mode by sending a Session Ticket [RFC5077] to the client for it to store. However, once the connection is closed, any existing subscriptions will be dropped. When the TLS session is resumed, the DNS Push Notification server will not have any subscription state and will proceed as with any other new connection. Use of TLS Session Resumption allows a new TLS connection to be set up more quickly, but the client will still have to recreate any desired subscriptions.

8. IANA Considerations

This document defines the service name: "_dns-push-tls._tcp". It is only applicable for the TCP protocol. This name is to be published in the IANA Service Name Registry.

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

9. Acknowledgements

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

This draft has been improved due to comments from Ran Atkinson, Tim Chown, Mark Delany, Ralph Droms, Bernie Holz, Jan Komissar, Manju Shankar Rao, Markus Stenberg, and Dave Thaler.

10. References

10.1. Normative References

[I-D.ietf-dnsop-5966bis]

Dickinson, J., Dickinson, S., Bellis, R., Mankin, A., and D. Wessels, "DNS Transport over TCP - Implementation Requirements", draft-ietf-dnsop-5966bis-06 (work in progress), January 2016.

[I-D.ietf-dnsop-edns-tcp-keepalive]

Wouters, P., Abley, J., Dickinson, S., and R. Bellis, "The edns-tcp-keepalive EDNS0 Option", draft-ietf-dnsop-edns-tcp-keepalive-06 (work in progress), February 2016.

[I-D.ietf-tls-tls13]

Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", draft-ietf-tls-tls13-11 (work in progress), December 2015.

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

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

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

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

- [RFC1123] Braden, R., Ed., "Requirements for Internet Hosts - Application and Support", STD 3, RFC 1123, DOI 10.17487/RFC1123, October 1989, <<http://www.rfc-editor.org/info/rfc1123>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC2136] Vixie, P., Ed., Thomson, S., Rekhter, Y., and J. Bound, "Dynamic Updates in the Domain Name System (DNS UPDATE)", RFC 2136, DOI 10.17487/RFC2136, April 1997, <<http://www.rfc-editor.org/info/rfc2136>>.
- [RFC2782] Gulbrandsen, A., Vixie, P., and L. Esibov, "A DNS RR for specifying the location of services (DNS SRV)", RFC 2782, DOI 10.17487/RFC2782, February 2000, <<http://www.rfc-editor.org/info/rfc2782>>.
- [RFC4953] Touch, J., "Defending TCP Against Spoofing Attacks", RFC 4953, DOI 10.17487/RFC4953, July 2007, <<http://www.rfc-editor.org/info/rfc4953>>.
- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <<http://www.rfc-editor.org/info/rfc5246>>.
- [RFC5966] Bellis, R., "DNS Transport over TCP - Implementation Requirements", RFC 5966, DOI 10.17487/RFC5966, August 2010, <<http://www.rfc-editor.org/info/rfc5966>>.
- [RFC6066] Eastlake 3rd, D., "Transport Layer Security (TLS) Extensions: Extension Definitions", RFC 6066, DOI 10.17487/RFC6066, January 2011, <<http://www.rfc-editor.org/info/rfc6066>>.
- [RFC6195] Eastlake 3rd, D., "Domain Name System (DNS) IANA Considerations", RFC 6195, DOI 10.17487/RFC6195, March 2011, <<http://www.rfc-editor.org/info/rfc6195>>.
- [RFC6891] Damas, J., Graff, M., and P. Vixie, "Extension Mechanisms for DNS (EDNS(0))", STD 75, RFC 6891, DOI 10.17487/RFC6891, April 2013, <<http://www.rfc-editor.org/info/rfc6891>>.

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

10.2. Informative References

- [I-D.ietf-dnssd-hybrid]
Cheshire, S., "Hybrid Unicast/Multicast DNS-Based Service Discovery", draft-ietf-dnssd-hybrid-03 (work in progress), November 2015.
- [I-D.sekar-dns-llq]
Sekar, K., "DNS Long-Lived Queries", draft-sekar-dns-llq-01 (work in progress), August 2006.
- [IPJ.9-4-TCPSYN]
Eddy, W., "Defenses Against TCP SYN Flooding Attacks", The Internet Protocol Journal, Cisco Systems, Volume 9, Number 4, December 2006.
- [RFC1996] Vixie, P., "A Mechanism for Prompt Notification of Zone Changes (DNS NOTIFY)", RFC 1996, DOI 10.17487/RFC1996, August 1996, <<http://www.rfc-editor.org/info/rfc1996>>.
- [RFC4287] Nottingham, M., Ed. and R. Sayre, Ed., "The Atom Syndication Format", RFC 4287, DOI 10.17487/RFC4287, December 2005, <<http://www.rfc-editor.org/info/rfc4287>>.
- [RFC5077] Salowey, J., Zhou, H., Eronen, P., and H. Tschofenig, "Transport Layer Security (TLS) Session Resumption without Server-Side State", RFC 5077, DOI 10.17487/RFC5077, January 2008, <<http://www.rfc-editor.org/info/rfc5077>>.
- [RFC6762] Cheshire, S. and M. Krochmal, "Multicast DNS", RFC 6762, DOI 10.17487/RFC6762, February 2013, <<http://www.rfc-editor.org/info/rfc6762>>.
- [RFC6763] Cheshire, S. and M. Krochmal, "DNS-Based Service Discovery", RFC 6763, DOI 10.17487/RFC6763, February 2013, <<http://www.rfc-editor.org/info/rfc6763>>.
- [RFC7525] Sheffer, Y., Holz, R., and P. Saint-Andre, "Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", BCP 195, RFC 7525, DOI 10.17487/RFC7525, May 2015, <<http://www.rfc-editor.org/info/rfc7525>>.

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

Authors' Addresses

Tom Pusateri
Seeking affiliation
Hilton Head Island, SC
USA

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

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

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

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.

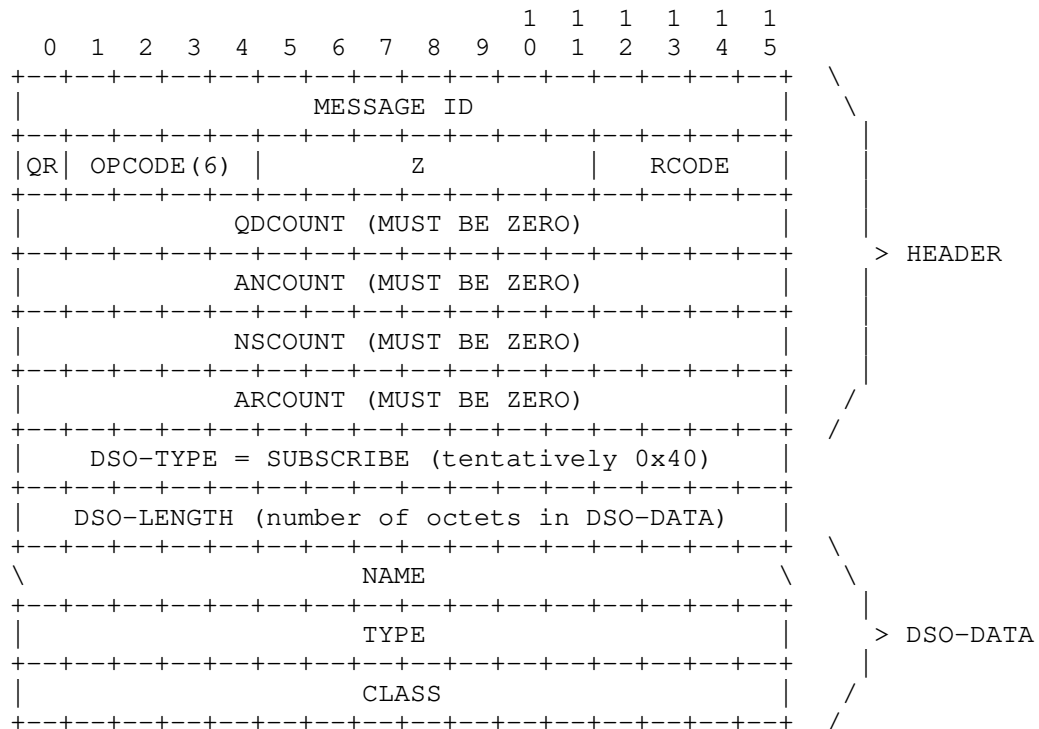


Figure 1: SUBSCRIBE Request

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.

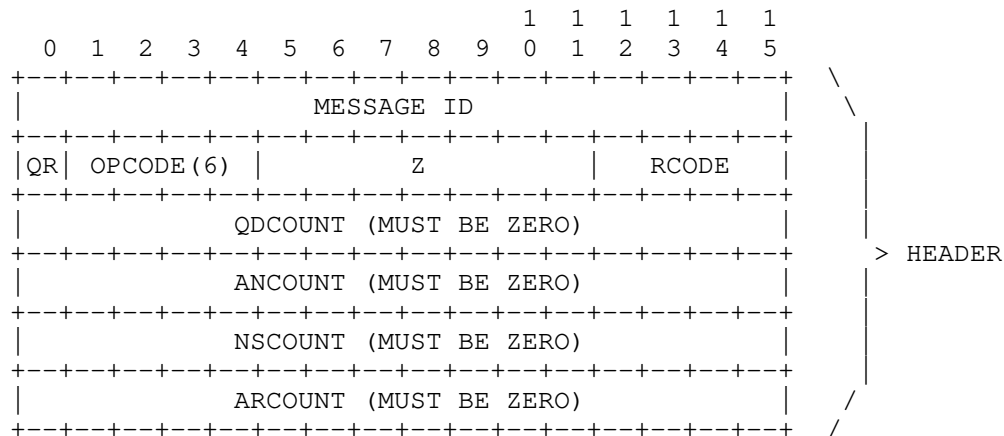


Figure 2: SUBSCRIBE Response

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 0xFFFFFFF0, 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 = 0xFFFFFFF0 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.

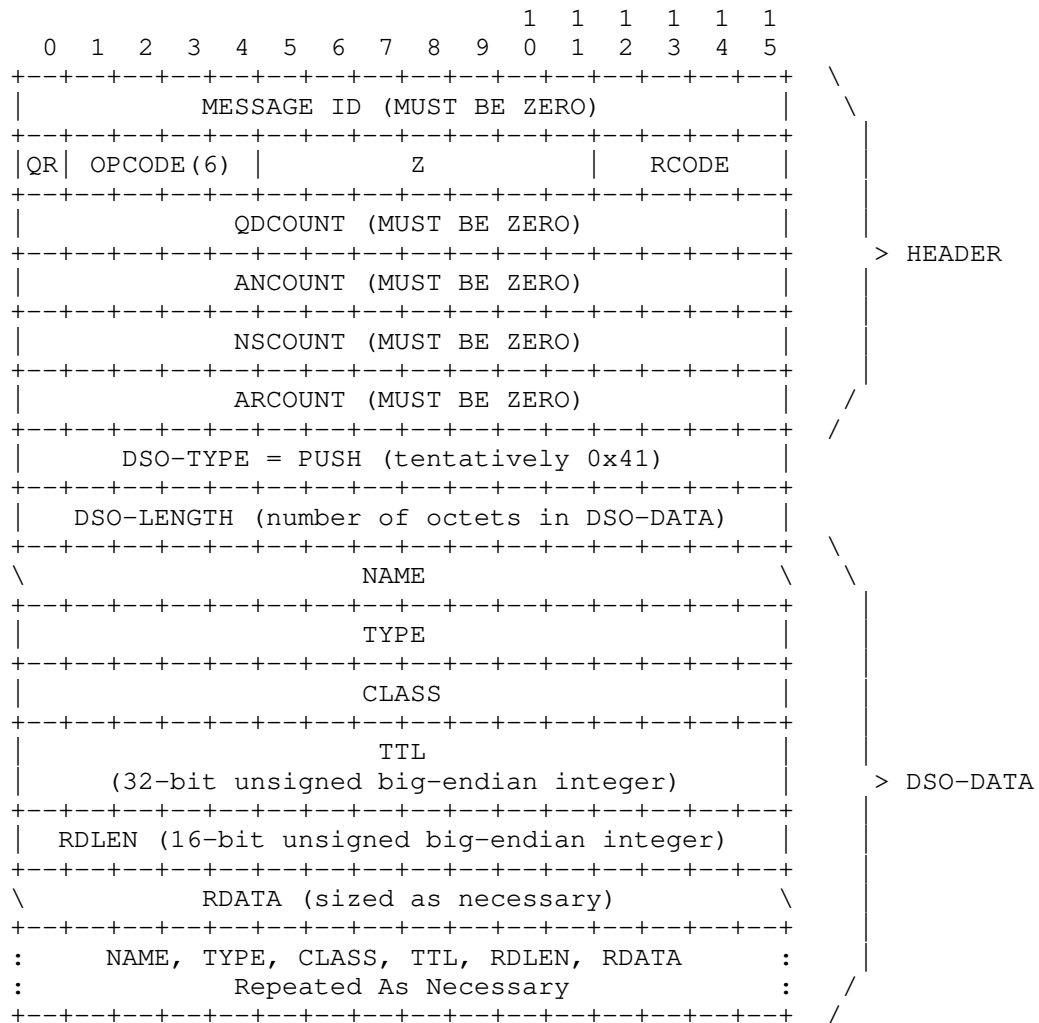


Figure 3: PUSH Message

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.

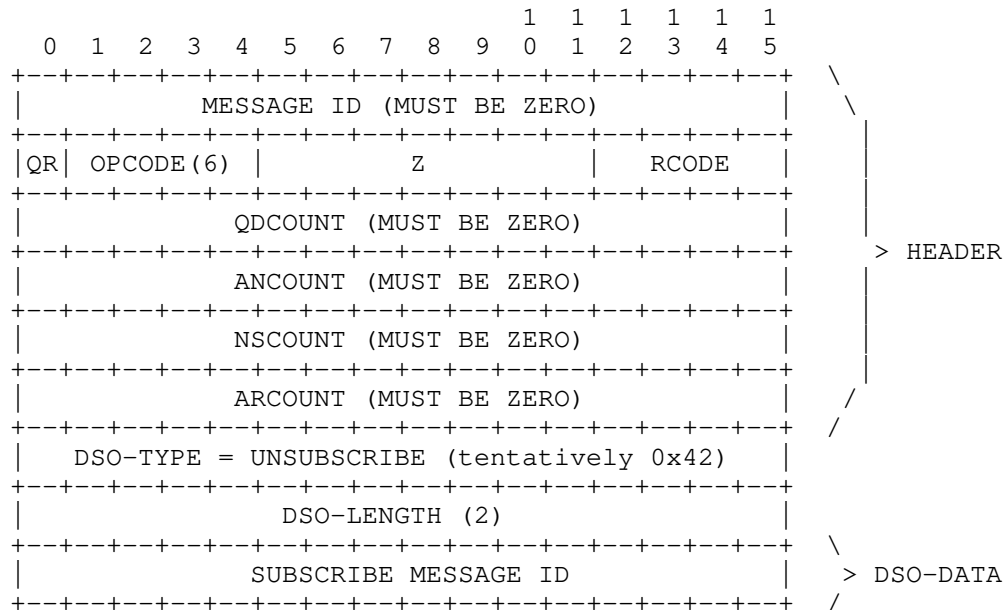


Figure 4: UNSUBSCRIBE Message

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.

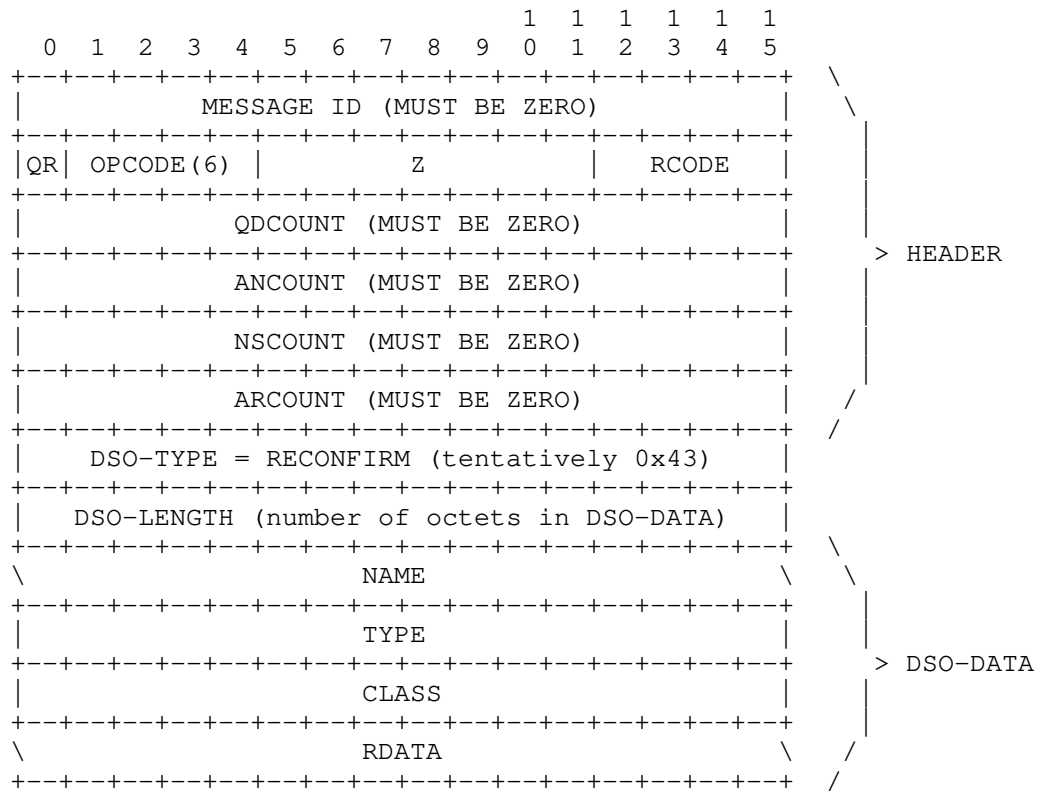


Figure 5: RECONFIRM Message

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>.
- [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>.
- [SYN] Eddy, W., "Defenses Against TCP SYN Flooding Attacks", The Internet Protocol Journal, Cisco Systems, Volume 9, Number 4, December 2006.
- [XEP0060] Millard, P., Saint-Andre, P., and R. Meijer, "Publish-Subscribe", XSF XEP 0060, July 2010.

Authors' Addresses

Tom Pusateri
Unaffiliated
Raleigh, NC 27608
USA

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

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

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

dnssd
Internet-Draft
Intended status: Informational
Expires: September 18, 2016

D. Otis
Trend Micro
H. Rafiee
Rozanak.com
March 17, 2016

Scalable DNS-SD (SSD) Threats
draft-otis-dnssd-scalable-dns-sd-threats-03

Abstract

mDNS combined with Service Discovery (DNS-SD) extends network resource distribution beyond the reach of multicast normally limited by the MAC Bridge. Since related resources are often not authenticated, either local resources are inherently trustworthy or are subsequently verified by associated services. Resource distribution becomes complex when a hybrid scheme combines adjacent network resources into a common unicast DNS-SD structure. This document explores related security considerations.

Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

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 18, 2016.

Copyright Notice

Copyright (c) 2016 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
1.1. Terminology and Abbreviations	4
2. Scalable DNS-SD (SSD) Realm and Global Namespace	4
2.1. Realm and Global Names	4
2.2. Exfiltration and Poisoning	9
2.3. Amplification Concerns	10
3. Protection of SSD related interchange	12
3.1. Link-Local	12
3.2. Authorization Issues	12
3.3. Authentication Issues	12
3.4. Privacy Considerations	12
4. IANA Considerations	13
5. Acknowledgements	13
6. References	13
6.1. Normative References	13
6.2. References - Informative	15
Appendix A. mDNS Example of Device Resolution Information	19
Appendix B. Uncontrolled Access Example	20
Authors' Addresses	21

1. Introduction

As described by [IEEE.802-1D.2004], MAC entities normally make services known via multicast announcements that do not extend beyond the Bridge as a basis for networking and layer 3 protocols. mDNS [RFC6762] allows non-centralized resource collection that can be structured as defined in DNS-SD [RFC6763]. This structure, when used in conjunction with DNS [RFC1035], provides an alternative to multicast announcement to deal with wireless links that are orders of magnitude less reliable than their wired counterparts. To improve transmission reliability, [IEEE.802-11.2012] requires positive acknowledgement of unicast frames but does not support positive acknowledgement of multicast frames. In [IEEE.802-11.2012] wireless networks, multicast frames are transmitted at a lower data rate supported by all receivers. Multicast on wireless networks may thereby lower overall network throughput. Some network administrators block some multicast traffic or convert it to a series of link-layer unicast frames. Other types of wireless networks may impose more demanding limitations as described by [RFC4944]. As a result, it is common to observe much higher loss of multicast frames on wireless compared against wired network technologies.

A namespace structured from adjacent networks using proxy-ed mDNS resources lacks a means to quickly resolve unicast name collision. Although an expensive promiscuous mode of unicast operation at multicast destinations might replicate mDNS features within a unicast environment, not well covered in [RFC4903] are issues related to wireless upstream clients unable to operate in promiscuous mode, indeterminate latency, and PPP links requiring a NAT or IPv4 ARP proxy. As such, a non-hybrid multicast/unicast scheme would be problematic.

Scalable DNS-SD (SSD) proposes to automatically gather autonomously named mDNS [RFC6762] resources of adjacent networks within separate namespace zones or realms as defined by [RFC7368]. Realms are often contained in separate subdomains that correspond with a link-local namespace. Making routable resources visible and accessible from other networks via unicast DNS [RFC1035] structured per DNS-SD [RFC6763] mitigates the level of multicast mDNS traffic in larger networks. Reliance on DNS [RFC1035] might leverage multi-network configurations that use mDNS [RFC6762] that proxy mDNS resources into DNS-SD using [I-D.ietf-dnssd-hybrid].

1.1. Terminology and Abbreviations

o Border: A point, typically resident on a router, between two networks at which filtering and forwarding policies for different types of traffic may be applied.

o ISP: Internet Service Provider. An entity that provides access to the Internet. In this document, a service provider specifically offers Internet access using IPv6 and may also offer IPv4 Internet access. The service provider can provide such access over a variety of different transport methods such as DSL, cable, wireless, and others.

o Realm: A network delimited by a defined border. i.e. a guest network within a homenet may form a realm.

o ULA: IPv6 Unique Local Address [RFC4193].

o Global Namespace: A globally unique name resolved within global DNS namespace bootstrapped from a root zone.

o Realm Namespace: A realm specific namespace that may not be resolved within the global DNS namespace, perhaps due to Special Use domain designations.

o Local Namespace: A namespace accessible for link-local resolution that may be referenced from an Ambiguous Local Qualified Domain Name (ALQDN) representing a network segment or broadcast domain.

2. Scalable DNS-SD (SSD) Realm and Global Namespace

2.1. Realm and Global Names

Conflicts between a local realm and global DNS [RFC1035] namespaces may occur. Without adequate feedback and latency constraints, a client may be unable to determine desired service targets. Target assessment may impair network stability when a cache policy renames resources propagated into different realms. Determining actual conflicts might depend on inherent identifiers such as MAC addresses or device specific GUIDs, otherwise conflict resolution may become increasingly byzantine.

2.1.1.1. SSD Structures

SSD locates SRV and TXT RRsets resources in the forms:

```
_<sn>._<Proto>.<SrvDOM>.<ParentDOM>.  
  
<Instance>._<sn>._<Proto>.<SrvDOM>.<ParentDOM>.  
  
<sub>._sub._<sn>._<Proto>.<SrvDOM>.<ParentDOM>.
```

For DNS-SD, Proto="_udp" represents all non-TCP transports otherwise it is "_tcp".

_<sn> = IANA Registered Service Name

To facilitate browsing, DNS-SD also supports a DNS meta-query of PTR RRsets at "_services._dns-sd._udp.<Domain>" which yields service names which may vary by host along with a domain name. Only the first two labels in the PTR rdata are relevant in the construction of subsequent Service Instance Enumeration PTR queries to further discover specific service types.

[I-D.ietf-dnssd-hybrid] conveyance extends '.local.' TLD namespace into '.home.' or an Ambiguous Local Qualified Domain Name (ALQDN) space, such as '.sitelocal.' as described in Section 3.7.4 of [RFC7368] where DNS [RFC1035] can be facilitated using split horizon methods described by [RFC6950] or similar schemes described by [RFC6281]. It seems there is some controversy regarding the designation of .home as a special use domain defined by [I-D.cheshire-homenet-dot-home] which does not follow conventions defined by [I-D.ietf-dnsop-alt-tld]. Either scheme avoids security issues related with global DNS publishing DNS-SD, even when configured in a split-horizon mode of operation. Unfortunately, the latter can not ensure the single de-facto .home label is formally excluded from the global DNS when designating a namespace that does not use multicast DNS. Any scheme supporting a sitelocal namespace must ensure queries are not forwarded to the Internet or to global DNS servers.

[I-D.ietf-dnssd-hybrid] suggests a split of traditional namespace that is restricted to letters, digits and hyphens and resolves only address resources, from the rich text namespace resolving PTR, SRV and TXT that facilitate service browsing. These resources are further bifurcated into separate link related namespace resources.

2.1.2. Scope of Discovery

As mDNS [RFC6762] is currently restricted to a single link, the scope of the advertisement is limited, by design, to the shared link between client and the device offering a service. When scaling for multi-links, the owner of the advertised service may propagate to a larger set of links or a larger realm than expected, which may result in unauthorized clients (from the perspective of the owner) connecting to the advertised service. It also discloses information (about the host and service) to a larger set of potential attackers.

If the scope of the discovery is not properly constrained, then information leaks may happen beyond the appropriate network and expose the network to various forms of attack. As such, services normally limited to local link should be assigned a separate subdomain normally not accessible from the Internet.

To reduce the amount of multicast traffic, widely distributing mDNS resources using unicast DNS-SD may scale better, but exposure of mDNS [RFC6762] derived resources to the Internet along with possibly sensitive details has proven problematic as noted by [CERTvu550620]. Protocol vulnerabilities can be found in reports published by a large number of vendors, Computer Emergency Response Teams (CERT), and Computer Security Incident Response Teams (CSIRT). With this diversity of sources, specific concerns may not be captured by Request for Comments (RFC) publications of the Internet Engineering Task Force (IETF).

Services might be sought outside the ".local." domain when applications obtain domain search lists provided by DHCP ([RFC2131] and [RFC3315] for IPv4 and IPv6 respectively or RA DNSSL [RFC6106] also for IPv6. Unfortunately, DHCPv6 does not support ULA assignment which instead requires some sort of NAT found with VPNs. Domains also need to be published in DNS [RFC1035] as A-Labels [RFC3492] because IDNA2008 compliance depends on A-label enforcement by registrars.

The SRV scheme used by mDNS [RFC6762] has also been widely adopted in the Windows OS since it offered a functional replacement for Windows Internet Name Service (WINS) as their initial attempt lacked sufficient name hierarchy. Such common use may represent security considerations whenever these records might become automatically published.

2.1.2.1. Visual Spoofing

Visual selection of autonomously named resources becomes especially salient when names are not ensured to be uniquely represented. mDNS

[RFC6762] only requires compliance with [RFC5198] rather than IDNA2008 [RFC5895]. This less restrictive use of namespace may impair the defense of critical services from look-alike attack. mDNS [RFC6762] does not ensure instances are visually unique and allows spaces and punctuation not permitted by IDNA2008.

To better ensure local namespace can be recognized, alternative zones might replace ASCII punctuation and spaces in SrvDOM labels with the '_' character except when located as the leftmost character. Such a convention should reduce visual confusion and handling issues related to end of string parsing, since labels in DNS [RFC1035] normally do not contain spaces or punctuation. Nevertheless, DNS [RFC1035] is able to handle such labels within sub-domains of registered domains.

2.1.3. Restricted Distribution of Sitelocal Addresses

ULA or [RFC1918] addresses allow safer automatic publication in DNS since these addresses are unlikely to be routed beyond the site. These addresses also provide a simple scheme to ascertain which addresses should be blocked at a network boundary. The use of other addresses MUST require specific administrative confirmations. It should be noted in the Addendum example, the Brother printer published a Globally routable address.

When doing so, address translation or overlays using Unique Local Addresses, ULAs [RFC4193] can offer a significant level of protection since typical link-local addresses are not usable from other networks. Although ULAs are to be treated as being globally routable, both ULA or [RFC1918] addresses typically indicate site local. Section 3.2 of [RFC4193] are locally defined and handled as Global addresses although not intended to be routed beyond the site or to those not having explicit routing provisions.

Section 4.1 of [RFC4193] indicates the default behavior of exterior routing protocol sessions between administrative routing regions must be to ignore receipt of and not advertise prefixes in the FC00::/7 block. A network operator may specifically configure prefixes longer than FC00::/7 for inter-site communication. Specifically, these prefixes are not designed to aggregate. Routers by default do not block ULA prefixes which makes it important to confirm how ULA traffic is handled by the access provider.

ULA or [RFC1918] addresses are not normally routed over the Internet where their use provides a degree of isolation. For either home or enterprise networks, ULAs as an overlay network avoids dynamic network address translation tables and permits local routing that is isolated from direct Internet access. ULAs also permit local communications to remain unaffected by Internet related link failures

or scope limitations imposed by use of multicast protocols.

ULAs avoid a need to renumber internal-only private nodes when changing ISPs, or when ISPs restructure their address allocations. In these situations, use of ULA offers an effective tool for protecting internal-only nodes. As such, more than just the security considerations discussed in mDNS [RFC6762] and DNS-SD [RFC6763] are needed. For example, DNS-SD [RFC6763] states the following: "Since DNS-SD is just a specification for how to name and use records in the existing DNS, it has no specific additional security requirements over and above those that already apply to DNS queries and DNS updates." This simply overlooks that many devices are not automatically published in DNS nor can it be assumed they are able to handle the access that DNS might permit.

Current BTMM [RFC6281] only publishes ULAs of hosts in DNS able to authenticate when setting up an overlay network. Remaining devices, such as printers, are accessed as services offered by authenticating hosts. DNS resources should never be considered to offer privacy even in split-horizon configurations. DNS is unable to authenticate incoming queries nor can it offer application layer protection. Since many prefixes are expected to be in use within environments served by [I-D.ietf-dnssd-hybrid], errors related to network boundary detections becomes critical. As such, DNS SHOULD NOT publish addresses of devices unable to authenticate sessions traversing the Internet.

2.1.4. Avoid publishing DNS-SD resources in global DNS for home networks.

Even when DNSSEC uses NSEC or NSEC3, DNS-SD nevertheless exposes all listed services. Since most home systems use devices having limited resources and lack many security mechanisms normally used to ensure secure operation, these services should remain obscured from the Internet. Access to DNS-SD resources should be predicated on access granted using secure methods such as via VPN or IPSec exchanges.

2.1.5. Confirming Valid Resources

[RFC6950] Source Address Validation Improvement (SAVI) for DHCP as specified by [RFC7513] may help administrators qualify resources published in DNS. DNS-SD [RFC6763] recommends additional DNS records such as associated PTR and TXT SHOULD be generated to improve network efficiency for both unicast and multicast DNS-SD responses. This behavior further increases some risks related to query/response ratios and the likelihood of exposure of security sensitive information.

This new routable namespace also lacks the benefit of registrar involvement and may not afford an administrator an ability to mitigate nefarious activity, such as spoofing and phishing, without requisite controls having been first carefully established. When a device has access to different realms on multiple interfaces, it is not even clear how simple conflict resolution avoids threatening network stability while resolving names conveyed over disparate technologies.

2.1.6. Selective Forwarding based on IGMP or MLD snooping

Internet Group Management Protocol (IGMP) [RFC3376] supports multicast on IPv4 networks. Multicast Listener Discovery (MLD) [RFC3810] supports multicast management on IPv6 networks using ICMPv6 messaging in contrast to IGMP's bare IP encapsulation. This management allows routers to announce their multicast membership to neighboring routers. To optimize which LANs receive forwarded multicast frames, IGMP or MLD snooping can be used to determine the presence of listeners as a means to permit selective forwarding of multicast frames as well.

2.1.7. VLAN

Use of VLAN such as [RFC5517] can selectively extend multicast forwarding beyond Bridge limitations. While not a general solution, use of VLAN can both isolate and unite specific networks.

2.1.8. DHCP

IP address assignment and host registration might use a single or forwarded DHCP [RFC2131] or [RFC3315] server for IPv4 and IPv6 respectively that responds to interconnected networks as a means to register hosts and addresses. DHCP does not ensure against name or address conflict nor is it intended to configure routers.

2.2. Exfiltration and Poisoning

IP addresses made visible by DNSSEC [RFC4033] or DNS [RFC1035] that conform with DNS-SD [RFC6763] might be used, but the automated population of information into DNS [RFC1035] should be limited to administrative systems.

Automated conversion of mDNS [RFC6762] into unicast DNS [RFC1035] can be problematic from a security standpoint as can widespread propagation of multicast frames. mDNS [RFC6762] only requires compliance with [RFC5198] rather than IDNA2008 [RFC5895]. This means mDNS [RFC6762] will not ensure instances are visually unique and may contain spaces and punctuation not permitted by IDNA2008. As such,

this might cause users into becoming misled about the associated service.

SSD MUST include requisite filtering necessary to prevent data ex-filtration or the interception of sensitive services. Any exchanged data must first ensure locality, limit the resources gathered, resolved, and propagated to just those elements that can be effectively administrated. It is critical to ensure normal network protection is not lost for hosts that depend on link-local addressing and exclusion of routable traffic. A printer would be one such example of a host that can not be upgraded.

2.3. Amplification Concerns

It is unknown whether sufficient filtering of mDNS [RFC6762] to expose just those services likely needed will provide sufficient network protection. The extent of using IGMP or MLD for selective forwarding to mitigate otherwise spurious traffic is unknown.

Instance names and <SrvDOM> intended to correspond with link-local domains may use Unicode for Network Interchange [RFC5198] encoding but excludes ASCII control characters while also allowing escaped periods "\." and other punctuation and spaces.

For DNS-SD, Proto="_udp" represents all non-TCP transports otherwise it is "_tcp".

_<sn> = IANA Registered Service Name

Optional service browsing and various RRsets could result in large responses limited only by an MTU that may become fairly large in various HomeNet networking protocols.

Increased reliance on Resource Record Sets (RRsets) for discovery increases DDoS amplification concerns when overall RRset size is overlooked. The extent of this amplification had been constrained by the minimum MTU first established by [RFC0791] and noted by [RFC1191] of 576 bytes which accommodates 512 byte UDP DNS messages. Most Internet links are now able to handle much larger MTUs. Per [RFC2460], the minimum 1280 byte MTU is specified for IPv6.

To ensure minimal latency, DNS queries are first made using UDP. When a response becomes truncated, TCP is then normally attempted. Reliance on UDP has been relaxed by [RFC5966]. The size of a PTR RRset can be fairly large and result in UDP amplification issues when carried within a large minimum MTU. The potential query/response ratio may have a large impact on ISPs and in turn impact a large number of users.

At each of the DNS-SD SRV and TXT Resource Record Sets locations that offer instance and service enumerations, administration of the resulting RRsets must ensure these resources are suitable for distribution and the DNS-SD query to response ratio is suitable for Internet access.

DNS-SD [RFC6763] should not be viewed as a catalog structure of desired services suitable for Internet use. [I-D.ietf-dnssd-hybrid] is to be used to bridge adjacent networks but this risks conveying resources of hosts unable to safely facilitate Internet access. Since [I-D.ietf-dnssd-hybrid] should opt for the most conservative address mode when selecting addresses to be distributed, ULAs or [RFC1918] address should represent a default option rather than selecting GUAs.

Browsing change notification facilitated with [I-D.ietf-dnssd-push] uses the message structure defined by [RFC2136] but is based on TCP. TCP eliminates spoofed source query attacks and congestion issues. If neither QTYPE nor QCLASS are ANY (255) then this is a specific subscription to changes for the given name. When QTYPE or QCLASS are ANY (255) then this becomes a wildcard subscription to changes of the given name for any type and/or class, as appropriate.

Browsing resource synchronization should use [I-D.ietf-dnssd-push] instead of depending on expanded RRsets or UDP transactions. Directly using DNS when overloaded would be much slower. This is because DNS [RFC1035] recommends 5 second timeouts with a doubling on two subsequent retries for a total of 35 seconds.

2.3.1. Resource Exhaustion Threats

DNS is currently vulnerable whenever responses are much larger than associated queries which could occur when browsing a domain offering services from a large number of hosts. To mitigate specific problematic query sources, an experimental mode of DNS operation is described in a technical note: DNS Response Rate Limiting [ISC-TN-2012-1-Draft1]. Additional information is available at [RedBarn].

Another experiment is [I-D.ietf-dnsop-cookies] which reduces reliance on DNS Response Rate Limiting and minimizes resources needed to handle random initial exchanges in a manner as described by [RFC6013] for forged sources of initial TCP <Syn> where servers keep client state within encrypted cookies.

3. Protection of SSD related interchange

SSD protocols may require additional steps to ensure against the poisoning of resource collection where close attention should be given to the scope of a ULA or [RFC1918] where the related resources are not to be directly exchanged with the Internet.

3.1. Link-Local

[RFC3927] provides an overview of IPv4 address complexities related to dealing with multiple segments and interfaces. IPv6 introduces new paradigms in respect to interface address assignments which offer scoping as explained in [RFC4291].

3.2. Authorization Issues

DNSSEC [RFC4033] can assert the validity but not the veracity of records in a zone file. The trust model of the global DNS [RFC1035] relies on the fact that human administrators either a) manually enter resource records into a zone file, or b) configure the DNS [RFC1035] server to authenticate a trusted device (e.g., a DHCP server) that can automatically maintain such records.

An imposter may register on the local link and appear as a legitimate service. Such "rogue" services may then be automatically registered in wide area DNS-SD [RFC6763].

3.3. Authentication Issues

Up to now, the "plug-and-play" nature of mDNS [RFC6762] devices have relied only on physical connectivity to the local network. If a device is visible via mDNS [RFC6762], it had been assumed to be trusted. When multiple networks are involved, verifying a host is local using mDNS [RFC6762] is no longer possible so other verification schemes will be needed.

3.4. Privacy Considerations

Mobile devices such as smart phones that can expose the location of their owners by registering services in arbitrary zones pose a risk to privacy. Such devices must not register their services in arbitrary zones without the approval of their operators. However, it should be possible to configure one or more "safe" zones, e.g., based on subnet prefix, in which mobile devices may automatically register their services.

As noted in [CERTvu550620] private security information is leaked in many cases. This includes hostnames and MACs, networking details,

service related details such as those for Printers and NAS devices. Many consumer printers can not authenticate users or block addresses when connected with IPv6. Once this information is leaked, malefactors are thereby given unlimited access.

4. IANA Considerations

This document requires no IANA consideration.

5. Acknowledgements

The authors wish to acknowledge valuable contributions from the following: Dave Rand, John C. Klensin, Dan York, Harald Albrecht, and Paul Vixie

6. References

6.1. Normative References

[I-D.cheshire-homenet-dot-home]

Cheshire, S., "Special Use Top Level Domain 'home'", draft-cheshire-homenet-dot-home-02 (work in progress), November 2015.

[I-D.ietf-dnsop-alt-tld]

Kumari, W. and A. Sullivan, "The ALT Special Use Top Level Domain", draft-ietf-dnsop-alt-tld-03 (work in progress), September 2015.

[I-D.ietf-dnsop-cookies]

Eastlake, D. and M. Andrews, "Domain Name System (DNS) Cookies", draft-ietf-dnsop-cookies-09 (work in progress), January 2016.

[I-D.ietf-dnssd-hybrid]

Cheshire, S., "Hybrid Unicast/Multicast DNS-Based Service Discovery", draft-ietf-dnssd-hybrid-03 (work in progress), February 2016.

- [I-D.ietf-dnssd-push] Pusateri, T. and S. Cheshire, "DNS Push Notifications", draft-ietf-dnssd-push-05 (work in progress), January 2016.
- [RFC1035] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, DOI 10.17487/RFC1035, November 1987, <<http://www.rfc-editor.org/info/rfc1035>>.
- [RFC1918] Rekhter, Y., Moskowitz, B., Karrenberg, D., de Groot, G., and E. Lear, "Address Allocation for Private Internets", BCP 5, RFC 1918, DOI 10.17487/RFC1918, February 1996, <<http://www.rfc-editor.org/info/rfc1918>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC2460] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", RFC 2460, DOI 10.17487/RFC2460, December 1998, <<http://www.rfc-editor.org/info/rfc2460>>.
- [RFC2782] Gulbrandsen, A., Vixie, P., and L. Esibov, "A DNS RR for specifying the location of services (DNS SRV)", RFC 2782, DOI 10.17487/RFC2782, February 2000, <<http://www.rfc-editor.org/info/rfc2782>>.
- [RFC3492] Costello, A., "Punycode: A Bootstring encoding of Unicode for Internationalized Domain Names in Applications (IDNA)", RFC 3492, DOI 10.17487/RFC3492, March 2003, <<http://www.rfc-editor.org/info/rfc3492>>.
- [RFC3587] Hinden, R., Deering, S., and E. Nordmark, "IPv6 Global Unicast Address Format", RFC 3587, DOI 10.17487/RFC3587, August 2003, <<http://www.rfc-editor.org/info/rfc3587>>.
- [RFC4033] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "DNS Security Introduction and Requirements", RFC 4033, DOI 10.17487/RFC4033, March 2005, <<http://www.rfc-editor.org/info/rfc4033>>.
- [RFC4193] Hinden, R. and B. Haberman, "Unique Local IPv6 Unicast Addresses", RFC 4193, DOI 10.17487/RFC4193, October 2005, <<http://www.rfc-editor.org/info/rfc4193>>.

- [RFC4291] Hinden, R. and S. Deering, "IP Version 6 Addressing Architecture", RFC 4291, DOI 10.17487/RFC4291, February 2006, <<http://www.rfc-editor.org/info/rfc4291>>.
- [RFC5198] Klensin, J. and M. Padlipsky, "Unicode Format for Network Interchange", RFC 5198, DOI 10.17487/RFC5198, March 2008, <<http://www.rfc-editor.org/info/rfc5198>>.
- [RFC5895] Resnick, P. and P. Hoffman, "Mapping Characters for Internationalized Domain Names in Applications (IDNA) 2008", RFC 5895, DOI 10.17487/RFC5895, September 2010, <<http://www.rfc-editor.org/info/rfc5895>>.
- [RFC5966] Bellis, R., "DNS Transport over TCP - Implementation Requirements", RFC 5966, DOI 10.17487/RFC5966, August 2010, <<http://www.rfc-editor.org/info/rfc5966>>.
- [RFC6106] Jeong, J., Park, S., Beloeil, L., and S. Madanapalli, "IPv6 Router Advertisement Options for DNS Configuration", RFC 6106, DOI 10.17487/RFC6106, November 2010, <<http://www.rfc-editor.org/info/rfc6106>>.
- [RFC6762] Cheshire, S. and M. Krochmal, "Multicast DNS", RFC 6762, DOI 10.17487/RFC6762, February 2013, <<http://www.rfc-editor.org/info/rfc6762>>.
- [RFC6763] Cheshire, S. and M. Krochmal, "DNS-Based Service Discovery", RFC 6763, DOI 10.17487/RFC6763, February 2013, <<http://www.rfc-editor.org/info/rfc6763>>.

6.2. References - Informative

- [CERTvu550620] Seaman, C., "CERT Vulnerability Note VU#550620", March 2015, <<https://www.kb.cert.org/vuls/id/550620>>.
- [IEEE.802-11.2012] "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 Standard 802.11, February 2012, <<http://standards.ieee.org/getieee802/download/802.11-2012.pdf>>.
- [IEEE.802-1D.2004] Institute of Electrical and Electronics Engineers, "Information technology - Telecommunications and

information exchange between systems - Local area networks - Media access control (MAC) bridges", IEEE Standard 802.1D, February 2004, <<http://standards.ieee.org/getieee802/download/802.1D-2004.pdf>>.

[IEEE.802-3.2012]

"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 Standard 802.3, August 2012, <http://standards.ieee.org/getieee802/download/802.3-2012_section1.pdf>.

[ISC-TN-2012-1-Draft1]

Vixie, P. and Rhyolite, "DNS Response Rate Limiting (DNS RRL)", April 2012, <<http://ss.vix.su/~vixie/isc-tn-2012-1.txt>>.

[RFC0791] Postel, J., "Internet Protocol", STD 5, RFC 791, DOI 10.17487/RFC0791, September 1981, <<http://www.rfc-editor.org/info/rfc791>>.

[RFC1112] Deering, S., "Host extensions for IP multicasting", STD 5, RFC 1112, DOI 10.17487/RFC1112, August 1989, <<http://www.rfc-editor.org/info/rfc1112>>.

[RFC1191] Mogul, J. and S. Deering, "Path MTU discovery", RFC 1191, DOI 10.17487/RFC1191, November 1990, <<http://www.rfc-editor.org/info/rfc1191>>.

[RFC2131] Droms, R., "Dynamic Host Configuration Protocol", RFC 2131, DOI 10.17487/RFC2131, March 1997, <<http://www.rfc-editor.org/info/rfc2131>>.

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

[RFC3007] Wellington, B., "Secure Domain Name System (DNS) Dynamic Update", RFC 3007, DOI 10.17487/RFC3007, November 2000, <<http://www.rfc-editor.org/info/rfc3007>>.

[RFC3315] Droms, R., Ed., Bound, J., Volz, B., Lemon, T., Perkins, C., and M. Carney, "Dynamic Host Configuration Protocol for IPv6 (DHCPv6)", RFC 3315, DOI 10.17487/RFC3315,

July 2003, <<http://www.rfc-editor.org/info/rfc3315>>.

- [RFC3376] Cain, B., Deering, S., Kouvelas, I., Fenner, B., and A. Thyagarajan, "Internet Group Management Protocol, Version 3", RFC 3376, DOI 10.17487/RFC3376, October 2002, <<http://www.rfc-editor.org/info/rfc3376>>.
- [RFC3810] Vida, R., Ed. and L. Costa, Ed., "Multicast Listener Discovery Version 2 (MLDv2) for IPv6", RFC 3810, DOI 10.17487/RFC3810, June 2004, <<http://www.rfc-editor.org/info/rfc3810>>.
- [RFC3927] Cheshire, S., Aboba, B., and E. Guttman, "Dynamic Configuration of IPv4 Link-Local Addresses", RFC 3927, DOI 10.17487/RFC3927, May 2005, <<http://www.rfc-editor.org/info/rfc3927>>.
- [RFC4043] Pinkas, D. and T. Gindin, "Internet X.509 Public Key Infrastructure Permanent Identifier", RFC 4043, DOI 10.17487/RFC4043, May 2005, <<http://www.rfc-editor.org/info/rfc4043>>.
- [RFC4510] Zeilenga, K., Ed., "Lightweight Directory Access Protocol (LDAP): Technical Specification Road Map", RFC 4510, DOI 10.17487/RFC4510, June 2006, <<http://www.rfc-editor.org/info/rfc4510>>.
- [RFC4541] Christensen, M., Kimball, K., and F. Solensky, "Considerations for Internet Group Management Protocol (IGMP) and Multicast Listener Discovery (MLD) Snooping Switches", RFC 4541, DOI 10.17487/RFC4541, May 2006, <<http://www.rfc-editor.org/info/rfc4541>>.
- [RFC4903] Thaler, D., "Multi-Link Subnet Issues", RFC 4903, DOI 10.17487/RFC4903, June 2007, <<http://www.rfc-editor.org/info/rfc4903>>.
- [RFC4944] Montenegro, G., Kushalnagar, N., Hui, J., and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks", RFC 4944, DOI 10.17487/RFC4944, September 2007, <<http://www.rfc-editor.org/info/rfc4944>>.
- [RFC5517] HomChaudhuri, S. and M. Foschiano, "Cisco Systems' Private VLANs: Scalable Security in a Multi-Client Environment", RFC 5517, DOI 10.17487/RFC5517, February 2010, <<http://www.rfc-editor.org/info/rfc5517>>.
- [RFC6013] Simpson, W., "TCP Cookie Transactions (TCPCT)", RFC 6013,

- DOI 10.17487/RFC6013, January 2011,
<<http://www.rfc-editor.org/info/rfc6013>>.
- [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,
<<http://www.rfc-editor.org/info/rfc6281>>.
- [RFC6895] Eastlake 3rd, D., "Domain Name System (DNS) IANA
Considerations", BCP 42, RFC 6895, DOI 10.17487/RFC6895,
April 2013, <<http://www.rfc-editor.org/info/rfc6895>>.
- [RFC6950] Peterson, J., Kolkman, O., Tschafenig, H., and B. Aboba,
"Architectural Considerations on Application Features in
the DNS", RFC 6950, DOI 10.17487/RFC6950, October 2013,
<<http://www.rfc-editor.org/info/rfc6950>>.
- [RFC7217] Gont, F., "A Method for Generating Semantically Opaque
Interface Identifiers with IPv6 Stateless Address
Autoconfiguration (SLAAC)", RFC 7217, DOI 10.17487/
RFC7217, April 2014,
<<http://www.rfc-editor.org/info/rfc7217>>.
- [RFC7368] Chown, T., Ed., Arkko, J., Brandt, A., Troan, O., and J.
Weil, "IPv6 Home Networking Architecture Principles",
RFC 7368, DOI 10.17487/RFC7368, October 2014,
<<http://www.rfc-editor.org/info/rfc7368>>.
- [RFC7513] Bi, J., Wu, J., Yao, G., and F. Baker, "Source Address
Validation Improvement (SAVI) Solution for DHCP",
RFC 7513, DOI 10.17487/RFC7513, May 2015,
<<http://www.rfc-editor.org/info/rfc7513>>.
- [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,
<<http://www.rfc-editor.org/info/rfc7558>>.
- [RedBarn] Vixie, P. and Rhyolite, "Response Rate Limiting in the
Domain Name System (DNS RRL)", June 2012,
<<http://www.redbarn.org/dns/ratelimits>>.

Appendix A. mDNS Example of Device Resolution Information

```
dns-sd -L "Brother MFC-9560CDW" _printer._tcp local
Lookup Brother MFC-9560CDW._printer._tcp.local
```

```
16:00:26.965 Brother\032MFC-9560CDW._printer._tcp.local.
can be reached at BRN30066C239958.local.:515
(interface 4) Flags: 2 txtvers=1 qtotal=1
pdl=application/vnd.hp-PCL,application/vnd.brother-hbp
rp=duerqxesz5090 ty=Brother\ MFC-9560CDW\
product=\\(Brother\ MFC-9560CDW\
adminurl=http://BRN30066C239958.local./
priority=75 usb_MFG=Brother usb_MDL=MFC-9560CDW
Color=T Copies=T Duplex=F PaperCustom=T Binary=T Transparent=T TBCP=F
```

Timestamp	A/R	Flg	if	Hostname	Address	TTL
16:14:34.855	Add	3	4	BRN30066C239958.local.	192.168.99.99	245
16:14:34.856	Add	2	4	BRN30066C239958.local.	2699:9999:7300:1510:3205:5CFF:FE23:9958%<0>245	

```
dns-sd -L "Canon MX920 series" _printer._tcp local.
Lookup Canon MX920 series._printer._tcp.local.
```

```
16:47:09.676 Canon\032MX920\032series._printer._tcp.local.
can be reached at 9299990000.local.:515 (interface 4) Flags: 2
txtvers=1 rp=auto note= qtotal=1 priority=60 ty=Canon\ MX920
\ series product=\\(Canon\ MX920\ series\
pdl=application/octet-stream adminurl=http://929999000000.local.
usb_MFG=Canon usb_MDL=MX920\ series
usb_CMD= UUID=00000000-0000-1000-8000-F48139999999
Color=T Duplex=T Scan=T Fax=F mac=F4:81:39:99:99:99
```

```
dns-sd -G v4v6 "9299999000000.local."
Timestamp A/R Flg if Hostname Address TTL
17:07:12.460 Add 3 4 929999000000.local.
FE80:0000:0000:0000:F681:39FF:FE92:9999%en0 65
17:07:12.461 Add 2 4 929999000000.local.
192.168.99.108 65
```

Appendix B. Uncontrolled Access Example

The risk is that adequate IPv6 filtering is simply not available on either current printers, scanners, cameras and other devices never intended to be used directly on the Internet.

For example, in the case of a printer:

ftp [DNS entry]

Trying 2699:9999:7300:1510:3205:5cff:fe23:9958...

Connected to [DNS entry]

220 FTP print service:V-1.13/Use the network password for the ID if updating.

Name (BRN30066C239958.local.:dlr): ftp

230 User ftp logged in.

ftp> ls

229 Entering Extended Passive Mode (|||62468|)

150 Transfer Start

total 1

-r--r--r--	1 root	printer	4096 Sep 28	2001 CFG-PAGE.TXT
------------	--------	---------	-------------	-------------------

-----	1 root	printer	0 Sep 28	2001 Toner-Low-----
-------	--------	---------	----------	---------------------

226 Data Transfer OK.

ftp>

From here, I can print a file with no further authentication. But the printer also now appears on the Internet with TCP ports 21,23,25,80,515,631 and 9100 active. I can scan a document that was left in the flatbed. I can send a fax. Or I can print many copies of black pages if I want to do a physical DOS. And, thanks to the globally routable address present, I can reach this from anywhere in the world.

Authors' Addresses

Douglas Otis
Trend Micro
10101 N. De Anza Blvd
Cupertino, CA 95014
USA

Phone: +1.408.257-1500
Email: doug_otis@trendmicro.com

Hosnieh Rafiee
Rozanak.com
Munich
Germany

Phone: +49 (0)176 57587575
Email: ietf@rozanak.com

