Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: January 2, 2015

A. Aggarwal
Qualcomm (QCE)

July 01, 2014

Optimizing DNS-SD query using TXT records
draft-aggarwal-dnssd-optimize-query-00

Abstract

DNS-SD allows a client to find a list of named instances of a service name over a particular transport within a domain of interest using standard DNS queries. As the number of potential responders increases, DNS-SD based discovery doesn't scale well. To mitigate the scaling issues, schemes to narrow down the search context would be needed. The document proposes to include key/value pairs in the form of a DNS TXT record along with the service name in the DNS query to assist with the discovery process. The DNS TXT record can be placed in the additional section of the query without requiring any changes to the structure of DNS messages.

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 January 2, 2015.

Table of Contents

1. Introduction

DNS-SD [RFC6763] in combination with mDNS [RFC6762] provide a discovery framework for service names registered with IANA over a local link. The objective of DNS-SD was to discover service instances that implement a given service. The use of mDNS scales well when the number of service instances that implement a given service are limited in number on the local link. However, when the number of wireless devices (e.g., Wi-Fi) approach hundreds of devices in a typical link, several service instances may respond when a DNS-SD query is issued for a given service name. The number of wireless devices is slated to grow further as more devices (things) are deployed as part of the Internet of Things (IoT) era.

At the same time, the DNS-SD protocol also enables discovery in various operating environments that rely on unicast DNS. Being able to narrow down the search context beyond the service name scope will be even more critical for such DNS-SD based discovery schemes to scale.

This contribution proposes one such solution.

This document proposes no change to the structure of DNS messages, and no new operation codes, response codes, resource record types, or any other new DNS protocol values.

1.1. Sample Use Cases

Some sample use cases that might experience scaling problems are mentioned below:

- A client application is looking to find color printers on the local network

- A lighting application needs to discover lighting fixtures or bulbs from a given manufacturer before establishing a session with each device to control the fixtures

2. Background

There are two potential mechanisms that can help a DNS-SD querier narrow down the answers of interest within the scope of DNS-SD [RFC6763]:

- Placing TXT records in the response: DNS has an efficiency feature whereby a DNS server may place additional records in the additional section of the DNS message. These additional records are records that the client did not explicitly request, but the server has reasonable grounds to expect that the client might request them shortly, so including them can save the client from having to issue additional queries. DNS-SD clarifies that the intention of DNS-SD TXT records is to convey a small amount of useful additional information about a service. Ideally, it should not be necessary for a client to retrieve this additional information before it can usefully establish a connection to the service. For a well-designed application protocol, even if there is no information at all in the TXT record, it should be possible, knowing only the host name, port number, and protocol being used, to communicate with that listening process and then perform version- or feature-negotiation to determine any further options or capabilities of the service instance.

- Using subtype as part of the question: DNS-SD allows a querier to send a subtype along with the service name. It does require that the subtype be 63 octets or fewer. DNS-SD RFC further clarifies that these should be documented in the protocol specification in question and/or in the "notes" field of the registration request sent to IANA.

It can be argued that mechanisms in place to narrow down the search beyond the service name are not very flexible. While nothing prevents an application implementing DNS-SD to eventually find the service instance of interest, it results in unnecessary traffic and delay. The proposal is to enable a richer search query mechanism by

explicitly adding key/value pairs in the query to avoid having to establish sessions with all services that match the service name in the question. Since DNS-SD allows a responder to include TXT records in the additional section with key-value pairs that it thinks the client may request, session establishment with the responder can be avoided if the desired key/value pairs (from the client's perspective) were included in the response.

3. Proposed Changes

DNS-SD as defined in [RFC6763] uses DNS TXT records to store arbitrary key/value pairs conveying additional information about the named service. Each key/value pair is encoded as its own constituent string within the DNS TXT record, in the form "key=value" (without the quotation marks). The proposal is for the client to be able to query for key/value pairs along with the service name. The DNS TXT record in the additional section of the query serves to send this additional information. Since DNS messages are allowed to have an additional section, this proposal doesn't require any changes to the structure of DNS messages.

DNS TXT record is allowed to have multiple key/value pairs. If multiple keys are present in a given TXT record, they are AND'ed and the responder must match all the keys in the TXT record. At the same time, DNS query could include more than one TXT record analogous to multiple TXT records in the response. If multiple TXT records are present in the query, they are logically OR'ed while the keys of each TXT record are AND'ed as stated above.

4. Realization of the proposal

Actual key/value pairs that can be sent are specified within the application protocol specification. Some examples to aid in the understanding of the proposal are mentioned below. They correspond to the use cases introduced earlier e.g.

- A client application looking for color printer can add color=true in the DNS TXT record as part of the additional section of the query

- A lighting application looking to discover bulbs by a certain manufacturer (such as Philips), can add the DNS TXT record in the additional section of the query with manuf=Philips

- The discovery scope can be further constrained by defining additional keys within the service protocol specification. By augmenting the query with additional context, the spurious traffic

and additional delay in finding the service instance of interest is reduced.

5. Deployment Considerations

An important deployment consideration is to analyze the behavior of an existing mDNS responder and unicast DNS to the receipt of DNS-SD query with a service name in the question section and TXT records in the additional section. If mDNS responder doesn't recognize TXT records, no filtering would occur and a response will be sent only if there is a match for the service name.

Regarding the behavior of unicast DNS when the standard query carries TXT records in the additional section, the DNS will respond strictly based on the service name in the question without any filtering based on the TXT records. DNS will issue a negative response unless there is a record matching the question. In summary, unicast DNS will continue to serve DNS queries that include TXT records in the additional section.

6. API Considerations

Several high level operating systems (Android, iOS) provide service discovery APIs. For the proposed enhancement to be realized, service registration should allow for service specific key/value pairs to be registered. This capability should already exist since it is allowed as per the current DNS-SD specification. The additional impact would be for the client application to be able to query for specific key/ value pairs along with the service name over a specific transport.

7. Security Considerations

The additional data (key/value pairs signifying the search context) beyond the service name in the DNS-SD query inherently reveals more information about what the client is searching for. DNS and mDNS today do not provide confidentiality, so observers already have access to potentially sensitive information such as what names one is requesting; addressing this issue is outside the scope of this extension. Even if confidentiality were to be solved, this extension still provides more information to the actual DNS/mDNS responders themselves. A client concerned about such information disclosure can simply choose not to use this extension for such queries, and thus trade off efficiency for privacy.

8. IANA Considerations

This memo includes no request to IANA.

9. Acknowledgements

Thanks to Dave Thaler for helping develop this idea and formalizing as a contribution for DNS-SD enhancements.

10. Normative References

[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.

[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.

Author’s Address

Ashutosh Aggarwal
Qualcomm (QCE)
5775 Morehouse Dr
San Diego , California 92121
USA

Phone: +1 858 658 2229
Email: aggarwal@qce.qualcomm.com

DNSOP Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 7, 2017

R. Bellis
ISC
S. Cheshire
Apple Inc.
J. Dickinson
Sinodun
A. Mankin
T. Pusateri
Unaffiliated
July 6, 2016

DNS Session Signaling
draft-bellis-dnsop-session-signal-00

Abstract

The Extension Mechanisms for DNS (EDNS(0)) [RFC6891] is explicitly defined to only have "per-message" semantics. This document defines a new Session Signaling OpCode used to carry persistent "per-session" type-length-values (TLVs), and defines an initial set of TLVs used to handle feature negotiation and to manage session timeouts and termination.

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 January 7, 2017.

Copyright Notice

Table of Contents

1. Introduction

The Extension Mechanisms for DNS (EDNS(0)) [RFC6891] is explicitly defined to only have "per-message" semantics. This document defines a new Session Signaling OpCode used to carry persistent "per-session" type-length-values (TLVs), and defines an initial set of TLVs used to handle feature negotiation and to manage session timeouts and termination.

A further issue with EDNS(0) is that there is no standard mechanism for a client to be able to tell whether a server has processed or otherwise acted upon the individual options contained with an OPT RR. The Session Signaling Opcode therefore requires an explicit response to each TLV within a request.

The message format (see Section 3.1) does not completely conform to the standard DNS packet format but is designed such that existing DNS

protocol parsers should be able to read the packet header and then simply ignore the extra data that appears thereafter.

2. Terminology

The terms "initiator" and "responder" correspond respectively to the initial sender and subsequent receiver of a Session Signaling TLV, regardless of which was the "client" and "server" in the usual DNS sense. The term "sender" may apply to either an initiator or responder.

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].

3. Protocol Details

Session Signaling messages MUST only be carried in protocols and in environments that can guarantee that the same two endpoints are in communication for the entire lifetime of the session.

Session Signaling messages relate only to the specific session in which they are being carried. Where a middle box (e.g. a DNS proxy, forwarder, or session multiplexer) is in the path the message MUST NOT be blindly forwarded in either direction by that middle box. This does not preclude the use of these messages in the presence of a NAT box that rewrites Layer 3 or Layer 4 headers but otherwise maintains the effect of a single session.

<< RB: OSI Layer 5 session analog? This is obviously intended for TCP "sessions" which aren’t distinct from Layer 4, but is this also applicable to DNS-o-DTLS, or DNS over UDP with an EDNS cookie - I think probably "yes" for the former, but "no" for the latter. I’m wondering whether "session" is even the right term to be using here >>

3.1. Message Format

A message containing a Session Signaling Opcode does not conform to the usual DNS message format. The 12 octet header format from [RFC1035] is preserved, but the four section count fields (QDCOUNT, ANCOUNT, NSCOUNT and ARCOUNT) MUST all be set to zero.

A list of TLVs are used in place of the usual sections, and MUST appear immediately after the 12 octet header. The total size of the TLVs is calculated from the value of the standard two octet framing word minus the 12 octets of the DNS header.

3.2. Message Handling

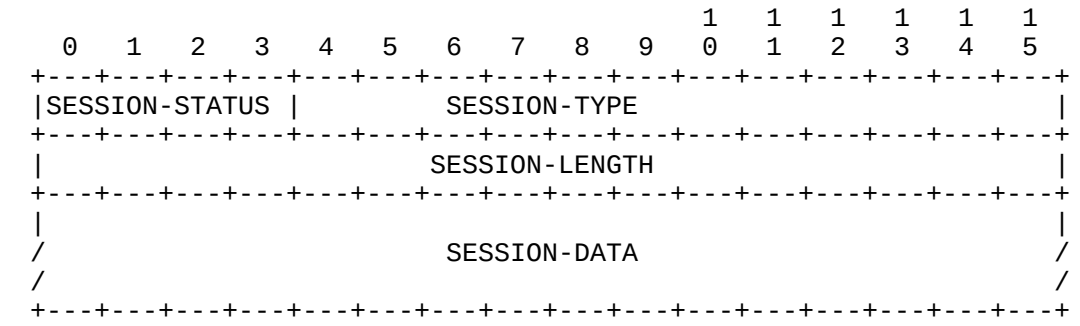
Both clients and servers may unilaterally send Session Signaling messages at any point in the lifetime of a session and are therefore considered to be the initiator with respect to that message. The initiator MUST set the value of the QR bit in the DNS header to zero (0), and the responder MUST set it to one (1).

Every Session Signaling request message MUST elicit a response (which MUST have the same ID in the DNS message header as in the request) and every TLV contained within the request requires a corresponding TLV in the response.

In order to preserve the correct sequence of state, Session Signaling requests MUST NOT be processed out of order. Similarly the TLVs in a message MUST be processed in the order in which they are contained in the message, and the order of the TLVs in the response MUST correspond with the order of the TLVs in the request.

<< RB: should the presence of a SS message create a "sequencing point", such that all pending responses must be answered? >>

3.3. TLV Format



SESSION-STATUS: A 4 bit field used in a response to indicate the success (or otherwise) of an operation, as defined in the DNS Session Signaling Status Codes Registry. It SHOULD contain "NOERROR" (0) in a request message but the responder MUST NOT reject the request if it does not.

SESSION-TYPE: A 12 bit field in network order giving the type of the current Session Signaling TLV per the IANA DNS Session Signaling Type Codes Registry.

SESSION-LENGTH: A 16 bit field in network order giving the size in octets of SESSION-DATA.

SESSION-DATA: Type-code specific. The SESSION-DATA field MUST be NUL padded to an even number of octets such that each Session Signaling TLV is aligned on a two octet boundary relative to the start of the first Session Signaling TLV. Padding octets MUST NOT be included in the calculation of SESSION-LENGTH but MUST be included in the calculation of the overall message length.

<< RB: the padding is specified such that client code can read the type and length fields directly from an aligned uint16_t array (with byte swapping) >>

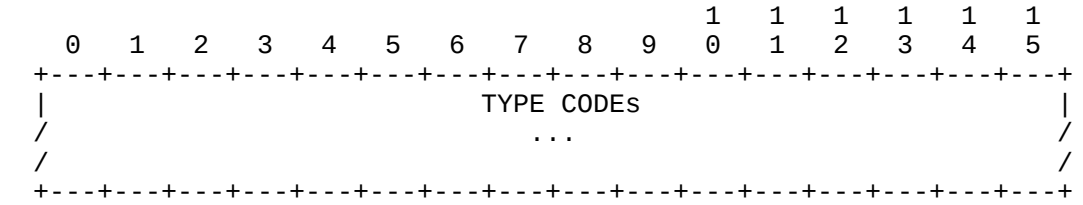
4. Mandatory TLVs

4.1. Feature Negotiation

4.1.1. TypeCode Support

The TypeCode Support TLV (1) is used to allow a client and server to exchange information about which Session Signaling Type Codes they support.

The SESSION-DATA contains a list of the Session Signaling Type Codes supported by the sender.



TYPE CODEs: A list of 16 bit words in network order comprising the complete list of Session Signaling Type Codes supported by the sender. Since a Session Signaling Type Code is in reality only a 12 bit value, the four most significant bits of each word MUST be zero. The number of TYPE CODEs can be calculated from the total length of the TLV.

An initiator MAY send its own list of supported Session Signaling Type Codes in a TypeCode Support TLV, and if sent they MUST be complete. Otherwise the SESSION-DATA MUST be empty. In either case the responder MUST respond with its complete list of supported Type Codes.

4.2. Layer 4 Connection Management TLVs

4.2.1. Terminate

The Terminate TLV (64) MAY be sent by a server to request that the client terminate the session, and when sent MUST be the only TLV present. It MUST NOT be requested by a client.

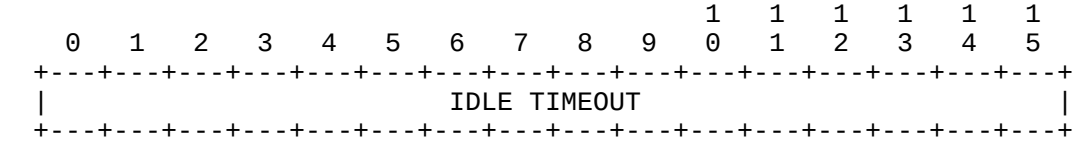
The client SHOULD terminate the session as soon as possible, but MAY wait for any inflight queries to be answered. It MUST NOT initiate any new queries over the existing session, nor send any further TLVs other than its response to the Terminate request.

<< RB: dns-sd push has a "reconnect delay" option but I think it's of questionable value since in an anycast or load-balancing architecture there's no way for the client to know which instance sent the option nor control which server instance the next connection will go to. This would IMHO be better controlled directly at the TCP layer. >>

4.2.2. Idle Timeout

The Idle Timeout TLV (65) has similar semantics to the EDNS TCP Keepalive Option [RFC7828]. It is used by a server to tell the client how long it may leave the current session idle for.

The SESSION-DATA is as follows:



IDLE TIMEOUT: the idle timeout for the current session, specified as a 16 bit word in network order in units of 100 milliseconds.

It is NOT an error for this TLV and the similar EDNS option to appear within the same session. The client SHOULD pay attention to the most recently received value, regardless of which method was used to send it.

The client SHOULD terminate the current session if it remains idle for longer than the specified timeout (and MAY of course terminate the session earlier). The server MAY unilaterally terminate the connection at any time, but SHOULD allow the client to keep the connection open if further messages are received before the idle timeout expires.

<< RB: this assumes that the EDNS OPT RR is added at the final stage of message processing, and therefore not affected by out-of-order processing - c.f. comment above about sequencing points >>

5. IANA Considerations

5.1. DNS Session Signaling OpCode Registration

IANA are directed to assign the value TBD for the Session Signaling OpCode in the DNS OpCodes Registry.

5.2. DNS Session Signaling Status Codes Registry

IANA are directed to create the DNS Session Signaling Status Codes Registry, with initial values as follows:

Code	Mnemonic	Description	Reference
0	NOERROR	TLV processed successfully	RFC-TBD1
4	NOTIMP	TLV not implemented	RFC-TBD1
5	REFUSED	TLV declined for policy reasons	RFC-TBD1

Registration of additional Session Signaling Status Codes requires Standards Action.

5.3. DNS Session Signaling Type Codes Registry

IANA are directed to create the DNS Session Signaling Type Codes Registry, with initial values as follows:

Type	Name	Status	Reference
0	Reserved		RFC-TBD1
1	TypeCode Support	Standard	RFC-TBD1
2 - 63	Unassigned, reserved for feature negotiation TLVs		
64	Terminate	Standard	RFC-TBD1
65	Idle Timeout	Standard	RFC-TBD1
66 - 127	Unassigned, reserved for session management TLVs		
127 - 3965	Unassigned		
3968 - 4031	Reserved for local / experimental use		
4032 - 4095	Reserved for future expansion		

Registration of additional Session Signaling Type Codes requires Expert Review. << RB: definition of process required? >>

6. Security Considerations

The authors are not aware of any specific security considerations introduced by this specification at this time.

7. Acknowledgements

TBW

8. Normative References

[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>.

[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>.

[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>.

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

Authors' Addresses

Ray Bellis
Internet Systems Consortium, Inc.
950 Charter Street
Redwood City CA 94063
USA

Phone: +1 650 423 1200
Email: ray@isc.org

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

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

John Dickinson
Sinodun Internet Technologies
Magadalen Centre
Oxford Science Park
Oxford OX4 4GA
United Kingdom

Allison Mankin
Unaffiliated

Email: allison.mankin@gmail.com

Tom Pusateri
Unaffiliated

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

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: December 12, 2016

C. Huitema
Microsoft
D. Kaiser
University of Konstanz

June 10, 2016

Privacy Extensions for DNS-SD
draft-huitema-dnssd-privacy-01.txt

Abstract

DNS-SD allows discovery of services published in DNS or MDNS. The publication normally discloses 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 to solve this problem by a two-stage approach. In the first stage, hosts discover Private Discovery Service Instances via DNS-SD using special formats to protect their privacy. These service instances correspond to Private Discovery Servers running on peers. In the second stage, hosts directly query these Private Discovery Servers via DNS-SD over TLS. A pairwise shared secret necessary to establish these connections is only known to hosts authorized by a pairing system.

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 December 12, 2016.

Table of Contents

1. Introduction

DNS-SD [RFC6763] enables distribution and discovery in local networks without configuration. It is very convenient for users, but it requires the public exposure of the offering and requesting identities along with information about the offered and requested services. Some of the information published by the announcements can be very revealing. These privacy issues and potential solutions are discussed in [KW14a] and [KW14b].

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 traveler wanting to upload pictures from a phone to a laptop when connected to the Wi-Fi network of an Internet cafe, or two travelers 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 analyzing 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 4, and describe the proposed solution in Section 5.

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]. Available services are described using three types of records:

PTR Record: Associates a service type in the domain with an "instance" name of this service type.

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 Service Instance Names

In the first phase of discovery, the client obtains all the PTR records associated with a service type in a given naming domain. Each PTR record contains a Service Instance Name defined in Section 4 of [RFC6763]:

Service Instance Name = <Instance> . <Service> . <Domain>

The <Instance> portion of the Service Instance Name is meant to convey enough information for users of discovery clients to easily select the desired service instance. Nodes that use DNS-SD over mDNS [RFC6762] in a mobile environment will rely on the specificity of the instance name to identify the desired service instance. In our example of users wanting to upload pictures to a laptop in an Internet Cafe, the list of available service instances may look like:

```
Alice’s Images          . _imageStore._tcp . local
Alice’s Mobile Phone    . _presence._tcp   . local
Alice’s Notebook        . _presence._tcp   . local
Bob’s Notebook          . _presence._tcp   . local
Carol’s Notebook        . _presence._tcp   . local
```

Alice will see the list on her phone and understand intuitively that she should pick the first item. The discovery will "just work".

However, DNS-SD/mDNS will reveal to anybody that Alice is currently visiting the Internet Cafe. It further discloses the fact that she uses two devices, shares an image store, and uses a chat application supporting the _presence protocol on both of her devices. She might currently chat with Bob or Carol, as they are also using a _presence supporting chat application. This information is not just available to devices actively browsing for and offering services, but to anybody passively listing to the network traffic.

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 record’s attribute and value pairs contain information on the characteristics of the corresponding service instance. This in turn reveals information about the devices that publish services. The amount of information varies widely with the particular service and its implementation:

- Some attributes like the paper size available in a printer, are the same on many devices, and thus only provide limited information to a tracker.

- 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.

Information contained in TXT records does not only breach privacy by making devices trackable, but might directly contain private information about a device user. For instance the _presence service reveals the "chat status" to everyone in the same network. Users might not be aware of that.

Further, TXT records often contain version information about services allowing potential attackers to identify devices running exploit-prone versions of a certain service.

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:

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

This combination of services and attributes 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.

There is however an argument that devices providing services can be discovered by observing the local traffic, because different services have different traffic patterns. The observation could in many cases also reveal some specificities of the service’s implementation. Even if the traffic is encrypted, the size and the timing of packets may be sufficient to reveal that information. This argument can be used to assess the priority of, for example, protecting the fact that a device publishes a particular service. However, we may assume that the developers of sensitive services will use counter-measures to defeat such traffic analysis.

2.5. Privacy Implication of Discovering Services

The consumers of services engage in discovery, and in doing so reveal some information such as the list of services 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. This can be benign if the service type is very common, but it could be more problematic for sensitive services, such as for example some private messaging services.

One way to protect clients would be to somehow encrypt the requested service types. Of course, just as we noted in Section 2.4, traffic analysis can often reveal the service.

3. Limits of a Simple Design

We first tried a simple design for mitigating the issues outlined in Section 2. The basic idea was to advertise obfuscated names, so as to not reveal the particularities of the service providers. This design is tempting, because it only requires minimal changes in the DNS-SD processing. However, as we will see in the following subsections, it has two important drawbacks:

- The simple design leads to UI issues, because users of unmodified DNS-SD agents will see a mix of clear text names and obfuscated names, which is unpleasant.

- With this simple design, there is no good way to hide the type of services provided or consumed by a specific node.

- The simple design either requires having a shared key between all "authorized users" of a service, which implies substandard key management practices, or publishing as many instances of a service as there are authorized users, which leads to the scaling issues discussed in Section 3.3.

Both issues are mitigated by the two-stage design presented in Section 4. The following subsections detail the simple design, and its drawbacks.

3.1. Obfuscated Instance Names

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

Authorized parties have to be able to "de-obfuscate" the names, while non-authorized third parties will not be. For example, if both Alice’s 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:

```
QwertyUiopAsdfghjk (Alice's Images)       . _imageStore._tcp . local
GobbeldygookBlaBla (Alice's Mobile Phone) . _presence._tcp   . local
MNbvCxzLkjhGfdEdhg (Alice's Notebook)     . _presence._tcp   . local
Abracadabragooklybok (Bob's Notebook)     . _presence._tcp   . local
Carol's Notebook                          . _presence._tcp   . local
```

Bob will see:

```
QwertyUiopAsdfghjk                    . _imageStore._tcp . local
GobbeldygookBlaBla                    . _presence._tcp   . local
MNbvCxzLkjhGfdEdhg                    . _presence._tcp   . local
Abracadabragooklybok (Bob's Notebook) . _presence._tcp   . local
Carol's Notebook                      . _presence._tcp   . local
```

Carol will see:

```
QwertyUiopAsdfghjk   . _imageStore._tcp . local
GobbeldygookBlaBla   . _presence._tcp   . local
MNbvCxzLkjhGfdEdhg   . _presence._tcp   . local
Abracadabragooklybok . _presence._tcp   . local
Carol's Notebook     . _presence._tcp   . 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.

3.2. Names of Obfuscated Services

Instead of publishing the actual service name in the SRV records, nodes could publish a randomized name. There are two plausible reasons for doing that:

- Having a different service name for privacy enhanced services will ensure that hosts that are not privacy aware are not puzzled by obfuscated service names.

- Using obfuscated service names prevents third parties from discovering which service a particular host is providing or consuming.

The first requirement can be met with a simple modification of an existing name. For example, instead of publishing:

```
QwertyUiopAsdfghjk . _imageStore._tcp . local
GobbeldygookBlaBla . _presence._tcp   . local
```

Alice could publish some kind of "translation" of the service name, such as:

```
QwertyUiopAsdfghjk . _vzntrFgber._tcp . local
GobbeldygookBlaBla . _cerfrapr._tcp   . local
```

The previous examples use rot13 translation. It does not provide any particular privacy, but it does ensure that obfuscated services are named differently from clear text services.

Making the service name actually private would require some actual encryption. The main problem with such solutions is that the client needs to know the service name in order to compose the DNS-SD query for services. There are several options:

- The service name is chosen by the client. For example, the client could encrypt the original service name and a nonce with a key shared between client and server. Upon receiving the queries, the server would attempt to decrypt the service name. If that succeeds, the server would respond with PTR records created on the fly for the new service name.

- The service name is chosen by the server and cannot be predicted in advance by the client. For example, the server could encrypt a nonce and the original service name. The client retrieves such services by doing a wild card query, then attempting to decrypt the received responses.

- The service name is chosen by the server in a way that can be predicted in advance by the client. For example, the server could encrypt some version of the data and time and the original service name. The data and time are encoded with a coarse precision, enabling the client to predict the value that the server is using, and to send the corresponding queries.

None of these solutions is very attractive. Creating records on the fly is a burden for the server. If clients must use wildcard queries, they will need to process lots of irrelevant data. If clients need to predict different instance names for each potential server, they will end up sending batches of queries with many different names. All of these solutions appear like big departures from the simplicity and robustness of the DNS-SD design.

3.3. Scaling Issues with Obfuscation

In Section 3.1, we assumed that each advertised record contains a name obfuscated with a shared key. This approach is easy to understand, but it contains hidden assumptions. Let's look at one of our examples:

Abracadabragooklybok (Bob's Notebook) . _presence._tcp . local

We only see one record for Bob's Notebook, obfuscated using the unique shared secret associated with Bob's Notebook. That means that every device paired with Bob's Notebook will have a copy of that shared secret. This is a possible solution, but there are known issues with having a secret shared with multiple entities:

- If for some reason the secret needs to be changed, every paired device will need a copy of the new secret before it can participate again in discovery.
- If one of the previous pairings becomes invalid, the only way to block the corresponding devices from discovery is to change the secret for all other devices.

Key management becomes much easier if it is strictly pair-wise. Two paired devices, or to pairs of users, can simply renew their pairing and get a new secret. If a device ceases to be trusted, the pairing data and the corresponding secret can just be deleted and forgotten. But using strictly pair-wise keys yields a scaling issue. Let's assume that:

- Each device maintains an average of N pairings.
- There are on average M devices present during discovery.

In the single key scenario, after issuing a broadcast query, the querier will receive a series of responses, each of which may well be obfuscated with a different key. If the receiver has N pre-existing pairings and receives M obfuscated responses, the cost will scale as O(M*N), i.e. try all N pairing keys for each of the M responses to see what matches. But if the keys are specific to each pair of devices, the obfuscation becomes complicated. When receiving a request, the publisher does not know which of its N keys the querier can decrypt. One simple solution would be to send N responses, but then the load on the querier will scale as O(M*N^2). That can go out of hand very quickly.

To solve the scaling issue, we consider a two-stage solution that uses an optimized discovery procedure to discover privacy-compatible

devices; and uses point to point encrypted exchanges to privately discover the available services.

4. Design of the Private DNS-SD Discovery Service

In this section, we present the design of a two-stage solution that enables private use of DNS-SD, without affecting existing users, and with better scalability than the simple solution presented in Section 3. The solution is largely based on the architecture proposed in [KW14b], which separates the general private discovery problems in three components: Pairing, discovery of a private discovery service, and actual service discovery through this private service. Pairing has to provide the private discovery servers with means for mutual authentication, e.g. with an authenticated shared secret. The private discovery servers provide actual service discovery with an authenticated connection. Our solution applies this architecture in the context of DNS-SD. It is based on the following components:

- Adding a pairing system to DNS-SD, described in Section 4.1, through which authorized peers can establish shared secrets;
- Defining the Private Discovery Service through which other services can be advertised in a private manner;
- And, publishing availability of the Private Discovery Service using DNS-SD, so that peers can discover their services without compromising their privacy.

These are independent with respect to means used for transmitting the necessary data.

4.1. Device Pairing

Any private discovery solution needs to differentiate between authorized devices, which are allowed to get information about discoverable entities, and other devices, which should not be aware of the availability of private entities. The commonly used solution to this problem is establishing a "device pairing". In our discovery scenarios, we envisage two kinds of pairings:

1. Inter-user pairing is a pairing between devices of "friends". Since it has to be performed manually, e.g. by the means described above, it is important to limit it to once per pair of friends.
2. Intra-user pairing is a pairing of devices of the same user. It can be performed without any configuration by a meta-service

(pairing data synchronization service) in a trusted (home) network.

The result of the pairing will be a shared secret, and optionally mutually authenticated public keys added to a local web of trust. Public key technology has many advantages, but shared secrets are typically easier to handle on small devices. We offer both a simple pairing just exchanging a shared secret, and an authenticated pairing using public key technology.

4.1.1. Shared Secret

Goal of the pairing process is establishing pairwise shared secrets. If two users can leverage a secure private off-channel, it suffices for one user to generate the shared secret and transmit it over this off-channel. It would be possible for the users to meet and orally agree on a password that both users enter in their devices. This has the disadvantage of user-chosen passwords to have low entropy and the inconvenience of having to type the password. Leveraging QR-codes can overcome these disadvantages: one user generates a shared secret, displays it in form of a QR-code, and the other user scans this code. Strictly speaking, displaying and scanning QR-codes does not establish a secure private channel, as others could also photograph this code; but it is reasonable secure for the application area of private service discovery. Using Bluetooth LE might also be considered satisfactory as a compromise between convenience and security.

4.1.2. Secure Authenticated Pairing Channel

Optionally, various versions of authenticated DH can be used to exchange a mutually authenticated shared secret (which among other possibilities can leverage QR-codes for key fingerprint verification). Using DH gives the benefit of provable security and the possibility to perform a pairing when not being able to meet in person. Further, using DH to generate the shared secret has the advantage of both parties contributing to the shared secret (multiparty computation).

4.1.3. Public Authentication Keys

The public/private key pair - if at all - is just used for the aforementioned authenticated DH to grant a mutually authenticated shared secret. Obtaining and verifying a friend’s public key can be achieved by different means. For obtaining the keys, we can either leverage an existing PKI, e.g. the PGP web of trust, or generate our own key pairs (and exchange them right before verifying). For authenticating the keys, which boils down to comparing fingerprints

on an off-channel, we distinguish between means that demand users to be in close proximity of each other, and means where users do not have to meet in person. The former can e.g. be realized by verifying a fingerprint leveraging QR-Codes, the latter by reading a fingerprint during a phone call or using the socialist millionaires protocol.

4.2. Discovery of the Private Discovery Service

The first stage of service discovery is to check whether instances of compatible Private Discovery Services are available in the local scope. The goal of that stage is to identify devices that share a pairing with the querier, and are available locally. The service instances can be discovered using regular DNS-SD procedures, but the list of discovered services will have to be filtered so only paired devices are retained.

We have demonstrated in Section 3.3 that simple obfuscation would require publishing as many records per publisher as there are pairings, which ends up scaling as O(M*N^2) in which M is the number of devices present and N is the number of pairings per device. We can mitigate this problem by using a special encoding of the instance name. Suppose that the publisher manages N pairings with the associated keys K1, K2, ... Kn. The instance name will be set to an encoding of N "proofs" of the N keys, where each proof is computed as function of the key and a nonce:

```
instance name = <nonce><F1><F2>..<Fn>

Fi = hash (nonce, Ki), where hash is a cryptographic hash
function.
```

The querier can test the instance name by computing the same "proof" for each of its own keys. Suppose that the receiver manages P pairings, with the corresponding keys X1, X2, .. Xp. The receiver verification procedure will be:

```
for each received instance name:
   retrieve nonce from instance name
   for (j = 1 to P)
      retrieve the key Xj of pairing number j
      compute F = hash(nonce, Xj)
      for (i=1 to N)
         retrieve the proof Fi
         if F is equal to Fi
            mark the pairing number j as available
```

The procedure presented here requires on average O(M*N) iterations of the hash function, which is the same scaling as the "shared secret" variant. It requires O(M*N^2) comparison operations, but these are less onerous than cryptographic operations. Further, when setting the nonce to a timestamp, the Fi have to be calculated only once per time interval.

The number of pairing proofs that can be encoded in a single record is limited by the maximum size of a DNS label, which is 63 bytes. Since this are characters and not pure binary values, nonce and proofs will have to be encoded using BASE64 ([RFC2045] section 6.8), resulting in at most 378 bits. The nonce should not be repeated, and the simplest way to achieve that is to set the nonce to a 32 bit timestamp value. The remaining 346 bits could encode up to 10 proofs of 32 bits each, which would be sufficient for many practical scenarios.

In practice, a 32 bit proof should be sufficient to distinguish between available devices. However, there is clearly a risk of collision. The Private Discovery Service as described here will find the available pairings, but it might also find a spurious number of "false positives." The chances of that happening are however quite small: less than 0.02% for a device managing 10 pairings and processing 10000 responses.

4.3. Private Discovery Service

The Private Discovery Service discovery allows discovering a list of available paired devices, and verifying that either party knows the corresponding shared secret. At that point, the querier can engage in a series of directed discoveries.

We have considered defining an ad-hoc protocol for the private discovery service, but found that just using TLS would be much simpler. The Directed Private Discovery service is just a regular DNS-SD service, accessed over TLS, using the encapsulation of DNS over TLS defined in [RFC7858]. The main difference with simple DNS over TLS is the need for authentication.

We assume that the pairing process has provided each pair of authorized client and server with a shared secret. We can use that shared secret to provide mutual authentication of clients and servers using "Pre Shared Key" authentication, as defined in [RFC4279] and incorporated in the latest version of TLS [I-D.ietf-tls-tls13].

One difficulty is the reliance on a key identifier in the protocol. For example, in TLS 1.3 the PSK extension is defined as:

```
opaque psk_identity<0..2^16-1>;

struct {
    select (Role) {
        case client:
            psk_identity identities<2..2^16-1>;

        case server:
            uint16 selected_identity;
    }
} PreSharedKeyExtension
```

According to the protocol, the PSK identity is passed in clear text at the beginning of the key exchange. This is logical, since server and clients need to identify the secret that will be used to protect the connection. But if we used a static identifier for the key, adversaries could use that identifier to track server and clients. The solution is to use a time-varying identifier, constructed exactly like the "hint" described in Section 4.2, by concatenating a nonce and the hash of the nonce with the shared secret.

4.3.1. A Note on Private DNS Services

Our solution uses a variant of the DNS over TLS protocol [RFC7858] defined by the DNS Private Exchange working group (DPRIVE). DPRIVE is also working on an UDP variant, DNS over DTLS [I-D.ietf-dprive-dnsodtls], which would also be a candidate.

DPRIVE and Private Discovery solve however two somewhat different problems. DPRIVE is concerned with the confidentiality to DNS transactions, addressing the problems outlined in [RFC7626]. However, DPRIVE does not address the confidentiality or privacy issues with publication of services, and is not a direct solution to DNS-SD privacy:

- 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.

- 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.

- Neither DNS over TLS nor DNS over DTLS applies to MDNS.

In contrast, we propose using mutual authentication of the client and server as part of the TLS solution, to ensure that only authorized parties learn the presence of a service.

4.4. Randomized Host Names

Instead of publishing their actual name in the SRV records, nodes could publish a randomized name. That is 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.

4.5. Timing of Obfuscation and Randomization

It is important that the obfuscation of instance names is 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. New instance names for the Private Discovery Services should be composed at the same time.

5. Private Discovery Service Specification

The proposed solution uses the following components:

- Host name randomization to prevent tracking.
- Device pairing yielding pairwise shared secrets.
- A Private Discovery Server (PDS) running on each host.
- Discovery of the PDS instances using DNS-SD.

These components are detailed in the following subsections.

5.1. Host Name Randomization

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 4.5. 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 [RFC7844]. 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.

5.2. Device Pairing

Nodes that want to leverage the Private Directory Service for private service discovery among peers MUST share a secret with each of these peers. The shared secret MUST be a 256 bit randomly chosen number. The secret SHOULD be exchanged via device Pairing. The pairing process SHALL establish a mutually authenticated secure channel to perform the shared secret exchange. It is RECOMMENDED for both parties to contribute to the shared secret, e.g. by using a Diffie-Hellman key exchange.

TODO: need to define the pairing service, or API. The API approach assumes that pairing is outside our scope, and is done using BT-LE, or any other existing mechanism. This is a bit of a cope-out. We could also define a pairing system that just sets the pairing with equivalent security as the "push button" or "PIN" solutions used for BT or Wi-Fi. And we could at this stage leverage a pre-existing security association, e.g. PGP identities or other certificates. If we do that, we should probably dedicate a top level section to specifying the minimal pairing service. Using a pre-existing asymmetric security association, we can use a key exchange similar to IKEv2 (RFC 7296). IKEv2 leverages the SIGMA protocols, which provide various methods of authenticated DH. It would also be possible to authenticate DH using symmetric passwords (e.g. Bellovin-Merritt).

5.3. Private Discovery Server

A Private Discovery Server (PDS) is a minimal DNS server running on each host. Its task is to offer resource records corresponding to private services only to authorized peers. These peers MUST share a secret with the host (see Section 5.2). To ensure privacy of the requests, the service is only available over TLS [RFC5246], and the shared secrets are used to mutually authenticate peers and servers.

The Private Name Server SHOULD support DNS push notifications [I-D.ietf-dnssd-push], e.g. to facilitate an up-to-date contact list in a chat application without polling.

5.3.1. Establishing TLS Connections

The PDS MUST only answer queries via DNS over TLS [RFC7858] and MUST use a PSK authenticated TLS handshake [RFC4279]. The client and server should negotiate a forward secure cypher suite such as DHE-PSK or ECDHE-PSK when available. The shared secret exchanged during pairing MUST be used as PSK.

When using the PSK based authentication, the "psk_identity" parameter identifying the pre-shared key MUST be composed as follow, using the conventions of TLS [RFC7858]:

```
struct {

          uint32 gmt_unix_time;

          opaque random_bytes[4];

} nonce;

long_proof = HASH(nonce  | pairing_key )
proof = first 12 bytes of long_proof
psk_identity = BASE64(nonce) "." BASE64(proof)
```

In this formula, HASH SHOULD be the function SHA256 defined in [RFC4055]. Implementers MAY eventually replace SHA256 with a stronger algorithm, in which cases both clients and servers will have to agree on that algorithm during the pairing process. The first 32 bits of the nonce are set to the current time and date in standard UNIX 32-bit format (seconds since the midnight starting Jan 1, 1970, UTC, ignoring leap seconds) according to the client’s internal clock. The next 32 bits of the nonce are set to a value generated by a secure random generator.

In this formula, the identity is finally set to a character string, using BASE64 ([RFC2045] section 6.8). This transformation is meant to comply with the PSK identity encoding rules specified in section 5.1 of [RFC4279].

The server will check the received key identity, trying the key against the valid keys established through pairing. If one of the key matches, the TLS connection is accepted, otherwise it is declined.

5.4. Publishing Private Discovery Service Instances

Nodes that provide the Private Discovery Service SHOULD advertise their availability by publishing instances of the service through DNS-SD.

The DNS-SD service type for the Private Discovery Service is "_pds._tls".

Each published instance describes one server and up to 10 pairings. In the case where a node manages more than 10 pairings, it should publish as many instances as necessary to advertise all available pairings.

Each instance name is composed as follows:

```
pick a 32 bit nonce, e.g. using the Unix GMT time.
set the binary identifier to the nonce.

for each of up to 10 pairings
   hint = first 32 bits of HASH(<nonce>|<pairing key>)
   concatenate the hint to the binary identifier

set instance-ID = BASE64(binary identifier)
```

In this formula, HASH SHOULD be the function SHA256 defined in [RFC4055], and BASE64 is defined in section 6.8 of [RFC2045]. The concatenation of a 32 bit nonce and up to 10 pairing hints result a bit string at most 332 bit long. The BASE64 conversion will produce a string that is up to 59 characters long, which fits within the 63 characters limit defined in [RFC6763].

5.5. Discovering Private Discovery Service Instances

Nodes that wish to discover Private Discovery Service Instances will issue a DNS-SD discovery request for the service type. These request will return a series of PTR records, providing the names of the instances present in the scope.

The querier SHOULD examine each instance to see whether it hints at one of its available pairings, according to the following conceptual algorithm:

```
for each received instance name:
   convert the instance name to binary using BASE64
   if the conversion fails,
      discard the instance.
   if the binary instance length is a not multiple of 32 bits,
      discard the instance.

   nonce = first 32 bits of binary.
   for each 32 bit hint after the nonce
      for each available pairing
         retrieve the key Xj of pairing number j
         compute F = hash(nonce, Xj)
         if F is equal to the 32 bit hint
            mark the pairing number j as available
```

Once a pairing has been marked available, the querier SHOULD try connecting to the corresponding instance, using the selected key. The connection is likely to succeed, but it MAY fail for a variety of reasons. One of these reasons is the probabilistic nature of the hint, which entails a small chance of "false positive" match. This will occur if the hash of the nonce with two different keys produces the same result. In that case, the TLS connection will fail with an authentication error or a decryption error.

5.6. Using the Private Discovery Service

Once instances of the Private Discovery Service have been discovered, peers can establish TLS connections and send DNS requests over these connections, as specified in DNS-SD.

6. 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. Hiding the identity of the publishing nodes prevents some forms of "targeting" of high value nodes. However, adversaries can attempt various attacks to break the anonymity of the service, or to deny it. A list of these attacks and their mitigations are described in the following sections.

6.1. Attacks Against the Pairing System

There are a variety of attacks against pairing systems. They may result in compromised pairing keys. If an adversary manages to acquire a compromised key, the adversary will be able to perform private service discovery according to Section 5.5. This will allow tracking of the service. The adversary will also be able to discover which private services are available for the compromised pairing.

To mitigate such attacks, nodes MUST be able to quickly revoke a compromised pairing. This is however not sufficient, as the compromise of the pairing key could remain undetected for a long time. For further safety, nodes SHOULD assign a time limit to the validity of pairings, discard the corresponding keys when the time has passed, and establish new pairings.

This later requirement of limiting the Time-To-Live can raise doubts about the usability of the protocol. The usability issues would be mitigated if the initial pairing provided both a shared secret and the means to renew that secret over time, e.g. using authenticated public keys.

6.2. Denial of Discovery of the Private Discovery Service

The algorithm described in Section 5.5 scales as O(M*N), where M is the number of pairing per nodes and N is the number of nodes in the local scope. Adversaries can attack this service by publishing "fake" instances, effectively increasing the number N in that scaling equation.

Similar attacks can be mounted against DNS-SD: creating fake instances will generally increase the noise in the system and make discovery less usable. Private Discovery Service discovery SHOULD use the same mitigations as DNS-SD.

The attack is amplified because the clients need to compute proofs for all the nonces presented in Private Discovery Service Instance names. One possible mitigation would be to require that such nonces correspond to rounded timestamps. If we assume that timestamps must not be too old, there will be a finite number of valid rounded timestamps at any time. Even if there are many instances present, they would all pick their nonces from this small number of rounded timestamps, and a smart client could make sure that proofs are only computed once per valid time stamp.

6.3. Replay Attacks Against Discovery of the Private Discovery Service

Adversaries can record the service instance names published by Private Discovery Service instances, and replay them later in different contexts. Peers engaging in discovery can be misled into believing that a paired server is present. They will attempt to connect to the absent peer, and in doing so will disclose their presence in a monitored scope.

The binary instance identifiers defined in Section 5.4 start with 32 bits encoding the "UNIX" time. In order to protect against replay attacks, clients MAY verify that this time is reasonably recent.

TODO: should we somehow encode the scope in the identifier? Having both scope and time would really mitigate that attack.

6.4. Denial of Private Discovery Service

The Private Discovery Service is only available through a mutually authenticated TLS connection, which provides good protections. However, adversaries can mount a denial of service attack against the service. In the absence of shared secrets, the connections will fail, but the servers will expend some CPU cycles defending against them.

To mitigate such attacks, nodes SHOULD restrict the range of network addresses from which they accept connections, matching the expected scope of the service.

This mitigation will not prevent denial of service attacks performed by locally connected adversaries; but protecting against local denial of service attacks is generally very difficult. For example, local attackers can also attack mDNS and DNS-SD by generating a large number of multicast requests.

6.5. Replay Attacks against the Private Discovery Service

Adversaries may record the PSK Key Identifiers used in successful connections to a private discovery service. They could attempt to replay them later against nodes advertising the private service at other times or at other locations. If the PSK Identifier is still valid, the server will accept the TLS connection, and in doing so will reveal being the same server observed at a previous time or location.

The PSK identifiers defined in Section 5.3.1 start with 32 bits encoding the "UNIX" time. In order to mitigate replay attacks, servers SHOULD verify that this time is reasonably recent, and fail

the connection if it is too old, or if it occurs too far in the future.

The processing of timestamps is however mitigated by the accuracy of computer clocks. If the check is too strict, reasonable connections could fail. To further mitigate replay attacks, servers MAY record the list of valid PSK identifiers received in a recent past, and fail connections if one of these identifiers is replayed.

7. IANA Considerations

This draft does not require any IANA action. (Or does it? What about the _pds tag?)

8. Acknowledgments

This draft results from initial discussions with Dave Thaler, and encouragements from the DNS-SD working group members.

9. References

9.1. 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>.

[RFC4279] Eronen, P., Ed. and H. Tschofenig, Ed., "Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)", RFC 4279, DOI 10.17487/RFC4279, December 2005, <http://www.rfc-editor.org/info/rfc4279>.

[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>.

[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>.

9.2. Informative References

[I-D.ietf-dnssd-push]
Pusateri, T. and S. Cheshire, "DNS Push Notifications", draft-ietf-dnssd-push-07 (work in progress), April 2016.

[I-D.ietf-dprive-dnsodtls]
Reddy, T., Wing, D., and P. Patil, "DNS over DTLS (DNSoD)", draft-ietf-dprive-dnsodtls-06 (work in progress), April 2016.

[I-D.ietf-intarea-hostname-practice]
Huitema, C., Thaler, D., and R. Winter, "Current Hostname Practice Considered Harmful", draft-ietf-intarea-hostname-practice-02 (work in progress), May 2016.

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

[KW14a] Kaiser, D. and M. Waldvogel, "Adding Privacy to Multicast DNS Service Discovery", DOI 10.1109/TrustCom.2014.107, 2014, <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=7011331>.

[KW14b] Kaiser, D. and M. Waldvogel, "Efficient Privacy Preserving Multicast DNS Service Discovery", DOI 10.1109/HPCC.2014.141, 2014, <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=7056899>.

[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>.

[RFC7844] Huitema, C., Mrugalski, T., and S. Krishnan, "Anonymity Profiles for DHCP Clients", RFC 7844, DOI 10.17487/RFC7844, May 2016, <http://www.rfc-editor.org/info/rfc7844>.

[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, <http://www.rfc-editor.org/info/rfc7858>.

Authors' Addresses

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

Email: huitema@microsoft.com

Daniel Kaiser
University of Konstanz
Konstanz 78457
Germany

Email: daniel.kaiser@uni-konstanz.de

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.

Table of Contents

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:

- 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
- 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.

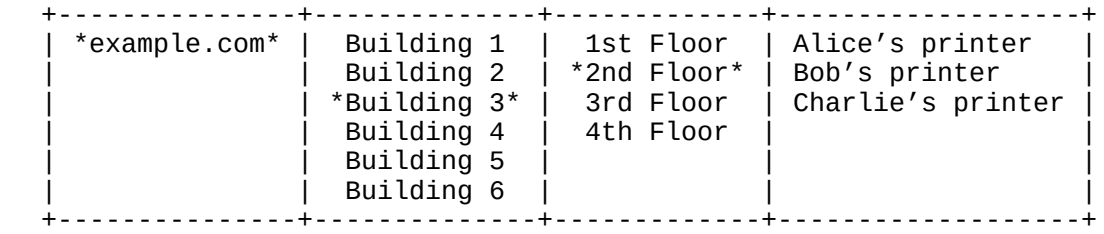


Figure 1: Illustrative GUI

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:
- Owner name, encoded as a two-byte compression pointer
- Two-byte rrtype (type PTR)
- Two-byte rrclass (class IN)
- Four-byte ttl
- Two-byte rdlength
- 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 SOULD 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 reasonble 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.

- 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.

- 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 RRSet TTL harmonisation, if the proxy has one Multicast DNS answer in its cache, it can reasonably assume that it has all of them.)

- 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.

- 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 visibile 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. Intelectual 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-hncp] Stenberg, M., Barth, S., and P. Pfister, "Home Networking Control Protocol", draft-ietf-homenet-hncp-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: January 9, 2017

T. Pusateri
Seeking affiliation
S. Cheshire
Apple Inc.

July 8, 2016

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

Abstract

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

Status of This Memo

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 January 9, 2017.

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

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

DNS records may be updated using DNS Update [RFC2136]. Other mechanisms such as a Hybrid Proxy [I-D.ietf-dnssd-hybrid] can also generate changes to a DNS zone. This document specifies a protocol for Unicast DNS clients to subscribe to receive asynchronous

notifications of changes to RRSets of interest. It is immediately relevant in the case of DNS Service Discovery [RFC6763] but is not limited to that use case, and provides a general DNS mechanism for DNS record change notifications. Familiarity with the DNS protocol and DNS packet formats is assumed [RFC1034] [RFC1035] [RFC6895].

1.1. Requirements Language

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

2. Motivation

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

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

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

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

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

Generally, a client SHOULD NOT keep a connection to a server open indefinitely if it has no active subscriptions on that connection. After 30 seconds with no active subscriptions the client SHOULD close the idle connection, and, if needed in the future, open a new connection.

3. Overview

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

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

The DNS Push Notification server for a zone is any server capable

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

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

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

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

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

This DNS Push Notification specification includes support for DNS classes, for completeness. However, in practice, it is anticipated that for the foreseeable future the only DNS class in use will be DNS class "IN", as it is today with existing DNS servers and clients. A DNS Push Notification server MAY choose to implement only DNS class "IN".

4. Transport

Implementations of DNS Update [RFC2136] MAY use either User Datagram Protocol (UDP) [RFC0768] or Transmission Control Protocol (TCP) [RFC0793] as the transport protocol, in keeping with the historical precedent that DNS queries must first be sent over UDP [RFC1123]. This requirement to use UDP has subsequently been relaxed [RFC7766].

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

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

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

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

4.1. Client-Initiated Termination

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

After terminating its last subscription on a connection via UNSUBSCRIBE, a client MAY close the connection immediately, or it may keep it open if it anticipates performing further operations on that connection in the future. If a client wishes to keep an idle connection open, it MUST continue to meet its keepalive obligations [RFC7828] or the server is entitled to close the connection (see below).

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

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

If a client has performed operations on this connection that it would not want lost (like DNS updates) then the client SHOULD do an orderly disconnect, sending a TCP FIN. In the BSD sockets API, sending a TCP FIN is achieved by calling "shutdown(s,SHUT_WR)" and keeping the socket open until all remaining data has been read from it.

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

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

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

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

4.2. Server-Initiated Termination

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

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

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

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

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

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

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

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

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

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

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

5. State Considerations

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

6. Protocol Operation

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

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

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

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

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

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

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

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

6.1. Discovery

The first step in DNS Push Notification subscription is to discover an appropriate DNS server that supports DNS Push Notifications for the desired zone. The client MUST also determine which TCP port on the server is listening for connections, which need not be (and often is not) the typical TCP port 53 used for conventional DNS, or TCP port 853 used for DNS over TLS [I-D.ietf-dprive-dns-over-tls].

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

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

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

4. Once the SOA is known (either by virtue of being seen in the Answer Section, or in the Authority Section), the client sends a DNS query with type SRV [RFC2782] for the record name "_dns-push-tls._tcp.<zone>", where <zone> is the owner name of the discovered SOA record.

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

6. If the zone in question is set up to offer DNS Push Notifications then this SRV record MUST exist. The SRV "target" contains the name of the server providing DNS Push Notifications for the zone. The port number on which to contact the server is in the SRV record "port" field. The address(es) of the target host MAY be included in the Additional Section, however, the address records SHOULD be authenticated before use as described below in Section 7.2 [RFC7673].

7. More than one SRV record may be returned. In this case, the "priority" and "weight" values in the returned SRV records are used to determine the order in which to contact the servers for subscription requests. As described in the SRV specification [RFC2782], the server with the lowest "priority" is first contacted. If more than one server has the same "priority", the "weight" indicates the weighted probability that the client should contact that server. Higher weights have higher probabilities of being selected. If a server is not reachable or is not willing to accept a subscription request, then a subsequent server is to be contacted.

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

6.2. DNS Push Notification SUBSCRIBE

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

SUBSCRIBE Response codes

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

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

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

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

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

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

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

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

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

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

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

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

6.3. DNS Push Notification UNSUBSCRIBE

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

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

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

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

6.4. DNS Push Notification Update Messages

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

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

The following figure shows the existing DNS Update header format:

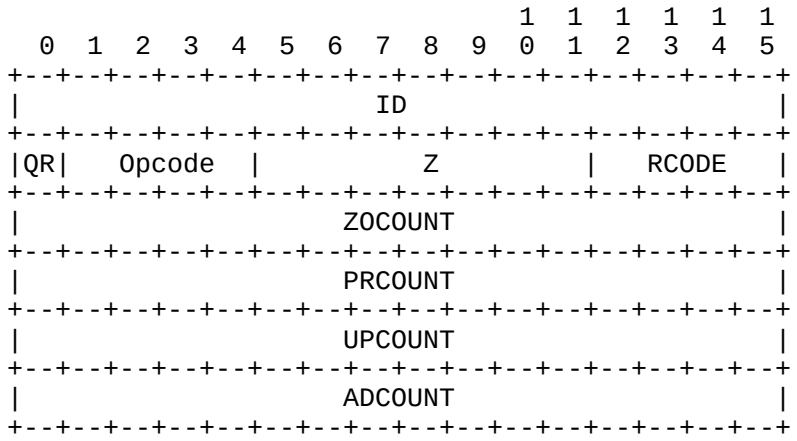


Figure 1

For DNS Push Notifications the following rules apply:

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

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

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

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

UPCOUNT specifies the number of records in the Update Section.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

6.5. DNS RECONFIRM

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

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

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

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

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

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

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

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

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

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

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

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

6.6. DNS Push Notification Termination Message

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

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

The following figure shows the existing DNS Update header format:

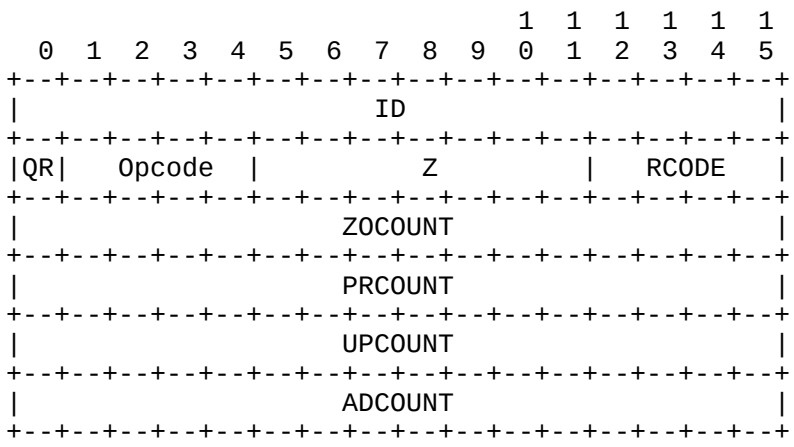


Figure 2

For Termination Messages the following rules apply:

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

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

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

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

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

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

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

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

Termination Response codes

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

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

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

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

7. Security Considerations

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

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

7.1. Security Services

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

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

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

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

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

7.2. TLS Name Authentication

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

record matching the target name is useable, an SNI extension MUST be used for the target name to ensure the client is connecting to the server it has authenticated. If the target name does not have a usable TLSA record, then the use of the SNI extension is optional.

7.3. TLS Compression

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

7.4. TLS Session Resumption

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

8. IANA Considerations

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

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

9. Acknowledgements

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

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

10. References

10.1. Normative References

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

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

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

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

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

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

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

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

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

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

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

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

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

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

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

[RFC7766] Dickinson, J., Dickinson, S., Bellis, R., Mankin, A., and D. Wessels, "DNS Transport over TCP - Implementation Requirements", RFC 7766, DOI 10.17487/RFC7766, March 2016, <http://www.rfc-editor.org/info/rfc7766>.

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

10.2. Informative References

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

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

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

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

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

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

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

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

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

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

[RFC7525] Sheffer, Y., Holz, R., and P. Saint-Andre, "Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", BCP 195, RFC 7525, DOI 10.17487/RFC7525, May 2015, <http://www.rfc-editor.org/info/rfc7525>.

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

Authors' Addresses

Tom Pusateri
Seeking affiliation
Hilton Head Island, SC
USA

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

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

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

Homenet Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 17, 2016

M. Stenberg
Independent

October 15, 2015

Auto-Configuration of a Network of Hybrid Unicast/Multicast DNS-Based Service Discovery Proxy Nodes
draft-ietf-homenet-hybrid-proxy-zeroconf-02

Abstract

This document describes how a proxy functioning between Unicast DNS-Based Service Discovery and Multicast DNS can be automatically configured using an arbitrary network-level state sharing mechanism.

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 April 17, 2016.

Copyright Notice

Table of Contents

1. Introduction

Section 3 ("Hybrid Proxy Operation") of [I-D.ietf-dnssd-hybrid] describes how to translate queries from Unicast DNS-Based Service Discovery described in [RFC6763] to Multicast DNS described in [RFC6762], and how to filter the responses and translate them back to unicast DNS.

This document describes what sort of configuration the participating hybrid proxy servers require, as well as how it can be provided using any network-wide state sharing mechanism such as link-state routing protocol or Home Networking Control Protocol [I-D.ietf-homenet-hncp]. The document also describes a naming scheme which does not even need to be same across the whole covered network to work as long as the specified conflict resolution works. The scheme can be used to provision both forward and reverse DNS zones which employ hybrid proxy for heavy lifting.

This document does not go into low level encoding details of the Type-Length-Value (TLV) data that we want synchronized across a network. Instead, we just specify what needs to be available, and assume every node that needs it has it available.

We go through the mandatory specification of the language used in Section 2, then describe what needs to be configured in hybrid proxies and participating DNS servers across the network in Section 3. How the data is exchanged using arbitrary TLVs is described in Section 4. Finally, some overall notes on desired behavior of different software components is mentioned in Section 5.

2. 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].

3. Hybrid proxy - what to configure

Beyond the low-level translation mechanism between unicast and multicast service discovery, the hybrid proxy draft [I-D.ietf-dnssd-hybrid] describes just that there have to be NS records pointing to hybrid proxy responsible for each link within the covered network.

In zero-configuration case, choosing the links to be covered is also non-trivial choice; we can use the border discovery functionality (if available) to determine internal and external links. Or we can use some other protocol’s presence (or lack of it) on a link to determine internal links within the covered network, and some other signs (depending on the deployment) such as DHCPv6 Prefix Delegation (as described in [RFC3633]) to determine external links that should not be covered.

For each covered link we want forward DNS zone delegation to an appropriate node which is connected to a link, and running hybrid proxy. Therefore the links’ forward DNS zone names should be unique

across the network. We also want to populate reverse DNS zone similarly for each IPv4 or IPv6 prefix in use.

There should be DNS-SD browse domain list provided for the network's domain which contains each physical link only once, regardless of how many nodes and hybrid proxy implementations are connected to it.

Yet another case to consider is the list of DNS-SD domains that we want hosts to enumerate for browse domain lists. Typically, it contains only the local network's domain, but there may be also other networks we may want to pretend to be local but are in different scope, or controlled by different organization. For example, a home user might see both home domain's services (TBD-TLD), as well as ISP's services under isp.example.com.

3.1. Conflict resolution within network

Any naming-related choice on node may have conflicts in the network given that we require only distributed loosely synchronized database. We assume only that the underlying protocol used for synchronization has some concept of precedence between nodes originating conflicting information, and in case of conflict, the higher precedence node MUST keep the name they have chosen. The one(s) with lower precedence MUST either try different one (that is not in use at all according to the current link state information), or choose not to publish the name altogether.

If a node needs to pick a different name, any algorithm works, although simple algorithm choice is just like the one described in Multicast DNS[RFC6762]: append -2, -3, and so forth, until there are no conflicts in the network for the given name.

3.2. Per-link DNS-SD forward zone names

How to name the links of a whole network in automated fashion? Two different approaches seem obvious:

1. Unique link name based - (unique-link).(domain).

2. Node and link name - (link).(unique-node).(domain).

The first choice is appealing as it can be much more friendly (especially given manual configuration). For example, it could mean just lan.example.com and wlan.example.com for a simple home network. The second choice, on the other hand, has a nice property of being local choice as long as node name can be made unique.

The type of naming scheme to use can be left as implementation option. And the actual names themselves SHOULD be also overridable, if the end-user wants to customize them in some way.

3.3. Reasonable defaults

Note that any manual configuration, which SHOULD be possible, MUST override the defaults provided here or chosen by the creator of the implementation.

3.3.1. Network-wide unique link name (scheme 1)

It is not obvious how to produce network-wide unique link names for the (unique-link).(domain) scheme. One option would be to base it on type of physical network layer, and then hope that the number of the networks won't be significant enough to confuse (e.g. "lan", or "wlan").

The network-wide unique link names should be only used in small networks. Given a larger network, after conflict resolution, identifying which link is 'lan-42.example.com' may be challenging.

3.3.2. Node name (scheme 2)

Our recommendation is to use some short form which indicates the type of node it is, for example, "openwrt.example.com". As the name is visible to users, it should be kept as short as possible. In theory even more exact model could be helpful, for example, "openwrt-buffalo-wzr-600-dhr.example.com". In practice providing some other records indicating exact node information (and access to management UI) is more sensible.

3.3.3. Link name (scheme 2)

Recommendation for (link) portion of (link).(node).(domain) is to use physical network layer type as base, or possibly even just interface name on the node if it's descriptive enough. For example, "eth0.openwrt.example.com" and "wlan0.openwrt.example.com" may be good enough.

4. TLVs

To implement this specification fully, support for following three different TLVs is needed. However, only the DNS Delegated Zone TLVs MUST be supported, and the other two SHOULD be supported.

4.1. DNS Delegated Zone TLV

This TLV is effectively a combined NS and A/AAAA record for a zone. It MUST be supported by implementations conforming to this specification. Implementations SHOULD provide forward zone per link (or optimizing a bit, zone per link with Multicast DNS traffic). Implementations MAY provide reverse zone per prefix using this same mechanism. If multiple nodes advertise same reverse zone, it should be assumed that they all have access to the link with that prefix. However, as noted in Section 5.3, mainly only the node with highest precedence on the link should publish this TLV.

Contents:

- Address field is IPv6 address (e.g. 2001:db8::3) or IPv4 address mapped to IPv6 address (e.g. ::FFFF:192.0.2.1) where the authoritative DNS server for Zone can be found. If the address field is all zeros, the Zone is under global DNS hierarchy and can be found using normal recursive name lookup starting at the authoritative root servers (This is mostly relevant with the S bit below).

- S-bit indicates that this delegated zone consists of a full DNS-SD domain, which should be used as base for DNS-SD domain enumeration (that is, (field)._dns-sd._udp.(zone) exists). Forward zones MAY have this set. Reverse zones MUST NOT have this set. This can be used to provision DNS search path to hosts for non-local services (such as those provided by ISP, or other manually configured service providers).

- B-bit indicates that this delegated zone should be included in network's DNS-SD browse list of domains at b._dns-sd._udp.(domain). Local forward zones SHOULD have this set. Reverse zones SHOULD NOT have this set.

- L-bit indicates that this delegated zone should be included in the network's DNS-SD legacy browse list of domains at lb._dns-sd._udp.(DOMAIN-NAME). Local forward zones SHOULD have this bit set, reverse zones SHOULD NOT.

- Zone is the label sequence of the zone, encoded according to section 3.1. ("Name space definitions") of [RFC1035]. Note that name compression is not required here (and would not have any point in any case), as we encode the zones one by one. The zone MUST end with an empty label.

In case of a conflict (same zone being advertised by multiple parties with different address or bits), conflict should be addressed according to Section 3.1.

4.2. Domain Name TLV

This TLV is used to indicate the base (domain) to be used for the network. If multiple nodes advertise different ones, the conflict resolution rules in Section 3.1 should result in only the one with highest precedence advertising one, eventually. In case of such conflict, user SHOULD be notified somehow about this, if possible, using the configuration interface or some other notification mechanism for the nodes. Like the Zone field in Section 4.1, the Domain Name TLV's contents consist of a single DNS label sequence.

This TLV SHOULD be supported if at all possible. It may be derived using some future DHCPv6 option, or be set by manual configuration. Even on nodes without manual configuration options, being able to read the domain name provided by a different node could make the user experience better due to consistent naming of zones across the network.

By default, if no node advertises domain name TLV, hard-coded default (TBD) should be used.

4.3. Node Name TLV

This TLV is used to advertise a node's name. After the conflict resolution procedure described in Section 3.1 finishes, there should be exactly zero to one nodes publishing each node name. The contents of the TLV should be a single DNS label.

This TLV SHOULD be supported if at all possible. If not supported, and another node chooses to use the (link).(node) naming scheme with this node's name, the contents of the network's domain may look misleading (but due to conflict resolution of per-link zones, still functional).

If the node name has been configured manually, and there is a conflict, user SHOULD be notified somehow about this, if possible, using the configuration interface or some other notification mechanism for the nodes.

5. Desirable behavior

5.1. DNS search path in DHCP requests

The nodes following this specification SHOULD provide the used (domain) as one item in the search path to it's hosts, so that DNS-SD browsing will work correctly. They also SHOULD include any DNS Delegated Zone TLVs' zones, that have S bit set.

5.2. Hybrid proxy

The hybrid proxy implementation SHOULD support both forward zones, and IPv4 and IPv6 reverse zones. It SHOULD also detect whether or not there are any Multicast DNS entities on a link, and make that information available to the network zeroconf daemon (if implemented separately). This can be done by (for example) passively monitoring traffic on all covered links, and doing infrequent service enumerations on links that seem to be up, but without any Multicast DNS traffic (if so desired).

Hybrid proxy nodes MAY also publish it's own name via Multicast DNS (both forward A/AAAA records, as well as reverse PTR records) to facilitate applications that trace network topology.

5.3. Hybrid proxy network zeroconf daemon

The daemon should avoid publishing TLVs about links that have no Multicast DNS traffic to keep the DNS-SD browse domain list as concise as possible. It also SHOULD NOT publish delegated zones for links for which zones already exist by another node with higher precedence.

The daemon (or other entity with access to the TLVs) SHOULD generate zone information for DNS implementation that will be used to serve the (domain) zone to hosts. Domain Name TLV described in Section 4.2 should be used as base for the zone, and then all DNS Delegated Zones described in Section 4.1 should be used to produce the rest of the entries in zone (see Appendix A.4 for example interpretation of the TLVs in Appendix A.3.

6. Limited zone stitching for host name resolution

Section 4.1 of the hybrid proxy specification [I-D.ietf-dnssd-hybrid] notes that the stitching of multiple .local zones into a single DNS-SD zone is to be defined later. This specification does not even attempt that, but for the purpose of host name resolution, it is possible to use the set of DNS Delegated Zone TLVs with S-bit or B-bit set to also provide host naming for the (domain). It is done by simply rewriting A/AAAA queries for (name).(domain) to every (name).(ddz-subdomain).(domain), and providing response to the host

when the first non-empty one is received, rewritten back to (name).(domain).

While this scheme is not very scalable, as it multiplies the number of queries by the number of links (given no response in cache), it does work in small networks with relatively few sub-domains.

7. Security Considerations

There is a trade-off between security and zero-configuration in general; if used network state synchronization protocol is not authenticated (and in zero-configuration case, it most likely is not), it is vulnerable to local spoofing attacks. We assume that this scheme is used either within (lower layer) secured networks, or with not-quite-zero-configuration initial set-up.

If some sort of dynamic inclusion of links to be covered using border discovery or such is used, then effectively service discovery will share fate with border discovery (and also security issues if any).

8. IANA Considerations

This document has no actions for IANA.

9. References

9.1. Normative references

[I-D.ietf-dnssd-hybrid]
Cheshire, S., "Hybrid Unicast/Multicast DNS-Based Service Discovery", draft-ietf-dnssd-hybrid-00 (work in progress), November 2014.

[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>.

[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>.

[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>.

9.2. Informative references

[I-D.ietf-homenet-hncp] Stenberg, M., Barth, S., and P. Pfister, "Home Networking Control Protocol", draft-ietf-homenet-hncp-09 (work in progress), August 2015.

[RFC3633] Troan, O. and R. Droms, "IPv6 Prefix Options for Dynamic Host Configuration Protocol (DHCP) version 6", RFC 3633, DOI 10.17487/RFC3633, December 2003, <http://www.rfc-editor.org/info/rfc3633>.

[RFC3646] Droms, R., Ed., "DNS Configuration options for Dynamic Host Configuration Protocol for IPv6 (DHCPv6)", RFC 3646, DOI 10.17487/RFC3646, December 2003, <http://www.rfc-editor.org/info/rfc3646>.

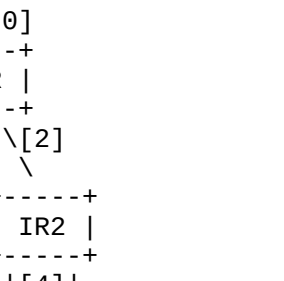
9.3. URIs

[1] https://github.com/sbyx/hnetd/

Appendix A. Example configuration

A.1. Used topology

Let’s assume home network that looks like this:



We’re not really interested about links [0], [1] and [2], or the links between IRs. Given the optimization described in Section 4.1, they should not produce anything to network’s Multicast DNS state (and therefore to DNS either) as there isn’t any Multicast DNS traffic there.

The user-visible set of links are [3] and [4]; each consisting of a LAN and WLAN link. We assume that ISP provides 2001:db8:1234::/48 prefix to be delegated in the home via [0].

A.2. Zero-configuration steps

Given implementation that chooses to use the second naming scheme (link).(node).(domain), and no configuration whatsoever, here's what happens (the steps are interleaved in practice but illustrated here in order):

1. Network-level state synchronization protocol runs, nodes get effective precedences. For ease of illustration, CER winds up with 2, IR1 with 3, and IR2 with 1.

2. Prefix delegation takes place. IR1 winds up with 2001:db8:1234:11::/64 for LAN and 2001:db8:1234:12::/64 for WLAN. IR2 winds up with 2001:db8:1234:21::/64 for LAN and 2001:db8:1234:22::/64 for WLAN.

3. IR1 is assumed to be reachable at 2001:db8:1234:11::1 and IR2 at 2001:db8:1234:21::1.

4. Each node wants to be called 'node' due to lack of branding in drafts. They announce that using the node name TLV defined in Section 4.3. They also advertise their local zones, but as that information may change, it's omitted here.

5. Conflict resolution ensues. As IR1 has precedence over the rest, it becomes "node". CER and IR2 have to rename, and (depending on timing) one of them becomes "node-2" and other one "node-3". Let us assume IR2 is "node-2". During conflict resolution, each node publishes TLVs for it's own set of delegated zones.

6. CER learns ISP-provided domain "isp.example.com" using DHCPv6 domain list option defined in [RFC3646]. The information is passed along as S-bit enabled delegated zone TLV.

A.3. TLV state

Once there is no longer any conflict in the system, we wind up with following TLVs (NN is used as abbreviation for Node Name, and DZ for Delegated Zone TLVs):

```
(from CER)
DZ {s=1,zone="isp.example.com"}

(from IR1)
NN {name="node"}

DZ {address=2001:db8:1234:11::1, b=1,
    zone="lan.node.example.com."}
DZ {address=2001:db8:1234:11::1,
    zone="1.1.0.0.4.3.2.1.8.b.d.0.1.0.0.2.ip6.arpa."}

DZ {address=2001:db8:1234:11::1, b=1,
    zone="wlan.node.example.com."}
DZ {address=2001:db8:1234:11::1,
    zone="2.1.0.0.4.3.2.1.8.b.d.0.1.0.0.2.ip6.arpa."}

(from IR2)
NN {name="node-2"}

DZ {address=2001:db8:1234:21::1, b=1,
    zone="lan.node-2.example.com."}
DZ {address=2001:db8:1234:21::1,
    zone="1.2.0.0.4.3.2.1.8.b.d.0.1.0.0.2.ip6.arpa."}

DZ {address=2001:db8:1234:21::1, b=1,
    zone="wlan.node-2.example.com."}
DZ {address=2001:db8:1234:21::1,
    zone="2.2.0.0.4.3.2.1.8.b.d.0.1.0.0.2.ip6.arpa."}
```

A.4. DNS zone

In the end, we should wind up with following zone for (domain) which is example.com in this case, available at all nodes, just based on dumping the delegated zone TLVs as NS+AAAA records, and optionally domain list browse entry for DNS-SD:

```
b._dns_sd._udp PTR lan.node
b._dns_sd._udp PTR wlan.node

b._dns_sd._udp PTR lan.node-2
b._dns_sd._udp PTR wlan.node-2

node AAAA 2001:db8:1234:11::1
node-2 AAAA 2001:db8:1234:21::1

node NS node
node-2 NS node-2

1.1.0.0.4.3.2.1.8.b.d.0.1.0.0.2.ip6.arpa. NS node.example.com.
2.1.0.0.4.3.2.1.8.b.d.0.1.0.0.2.ip6.arpa. NS node.example.com.
1.2.0.0.4.3.2.1.8.b.d.0.1.0.0.2.ip6.arpa. NS node-2.example.com.
2.2.0.0.4.3.2.1.8.b.d.0.1.0.0.2.ip6.arpa. NS node-2.example.com.
```

Internally, the node may interpret the TLVs as it chooses to, as long as externally defined behavior follows semantics of what's given in the above.

A.5. Interaction with hosts

So, what do the hosts receive from the nodes? Using e.g. DHCPv6 DNS options defined in [RFC3646], DNS server address should be one (or multiple) that point at DNS server that has the zone information described in Appendix A.4. Domain list provided to hosts should contain both "example.com" (the hybrid-enabled domain), as well as the externally learned domain "isp.example.com".

When hosts start using DNS-SD, they should check both b._dns-sd._udp.example.com, as well as b._dns-sd._udp.isp.example.com for list of concrete domains to browse, and as a result services from two different domains will seem to be available.

Appendix B. Implementation

There is an prototype implementation of this draft at hnetd github repository [1] which contains variety of other homenet WG-related things' implementation too.

Appendix C. Why not just proxy Multicast DNS?

Over the time number of people have asked me about how, why, and if we should proxy (originally) link-local Multicast DNS over multiple links.

At some point I meant to write a draft about this, but I think I'm too lazy; so some notes left here for general amusement of people (and to be removed if this ever moves beyond discussion piece).

C.1. General problems

There are two main reasons why Multicast DNS is not proxyable in the general case.

First reason is the conflict resolution depends on the RRsets staying constant. That is not possible across multiple links (due to e.g. link-local addresses having to be filtered). Therefore, conflict resolution breaks, or at least requires ugly hacks to work around.

A simple, but not really working workaround for this is to make sure that in conflict resolution, propagated resources always loses. Given that the proxy function only removes records, the result SHOULD be consistently original set of records winning. Even with that, the conflict resolution will effectively cease working, allowing for two instances of same name to exist (as both think they 'own' the name due to locally seen higher precedence).

Given some more extra logic, it is possible to make this work by having proxies be aware of both the original record sets, and effectively enforcing the correct conflict resolution results by (for example) passing the unfiltered packets to the losing party just to make sure they renumber, or by altering the RR sets so that they will consistently win (by inserting some lower rrclass/rrtype records). As the conflicts happen only in rrclass=1/rrtype=28, it is easy enough to add e.g. extra TXT record (rrtype 16) to force precedence even when removing the later rrtype 28 record. Obviously, this new RRset must never wind up near the host with the higher precedence, or it will cause spurious renaming loops.

Second reason is timing, which is relatively tight in the conflict resolution phase, especially given lossy and/or high latency networks.

C.2. Stateless proxying problems

In general, typical stateless proxy has to involve flooding, as Multicast DNS assumes that most messages are received by every host. And it won't scale very well, as a result.

The conflict resolution is also harder without state. It may result in Multicast DNS responder being in constant probe-announce loop, when it receives altered records, notes that it's the one that should own the record. Given stateful proxying, this would be just a

transient problem but designing stateless proxy that won't cause this is non-trivial exercise.

C.3. Stateful proxying problems

One option is to write proxy that learns state from one link, and propagates it in some way to other links in the network.

A big problem with this case lies in the fact that due to conflict resolution concerns above, it is easy to accidentally send packets that will (possibly due to host mobility) wind up at the originator of the service, who will then perform renaming. That can be alleviated, though, given clever hacks with conflict resolution order.

The stateful proxying may be also too slow to occur within the timeframe allocated for announcing, leading to excessive later renamings based on delayed finding of duplicate services with same name

A work-around exists for this though; if the game doesn't work for you, don't play it. One option would be simply not to propagate ANY records for which conflict has seen even once. This would work, but result in rather fragile, lossy service discovery infrastructure.

There are some other small nits too; for example, Passive Observation Of Failure (POOF) will not work given stateful proxying. Therefore, it leads to requiring somewhat shorter TTLs, perhaps.

Appendix D. Acknowledgements

Thanks to Stuart Cheshire for the original hybrid proxy draft and interesting discussion in Orlando, where I was finally convinced that stateful Multicast DNS proxying is a bad idea.

Also thanks to Mark Baugher, Ole Troan, Shwetha Bhandari and Gert Doering for review comments.

Appendix E. Changelog [RFC Editor: please remove]

draft-ietf-homenet-hybrid-proxy-zeroconf-02:

- Added subsection on simple zone stitching for host naming purposes.

draft-ietf-homenet-hybrid-proxy-zeroconf-01:

- Refreshed the draft while waiting on progress of draft-ietf-dnssd-hybrid.

Author's Address

Markus Stenberg
Independent
Helsinki 00930
Finland

Email: markus.stenberg@iki.fi

Network Working Group
Internet-Draft
Intended status: Informational
Expires: January 9, 2017

T. Lemon
Nominum, Inc.

July 8, 2016

Homenet Naming and Service Discovery Architecture
draft-lemon-homenet-naming-architecture-01

Abstract

This document recommends a naming and service discovery resolution architecture for homenets. This architecture covers local and global publication of names, discusses security and privacy implications, and addresses those implications. The architecture also covers name resolution and service discovery for hosts on the homenet, and for hosts that roam off of the homenet and still need access to homenet services.

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 January 9, 2017.

Copyright Notice

Table of Contents

1. Introduction

Associating domain names with hosts on the Internet is a key factor in enabling communication with hosts, particularly for service discovery. In order to provide name service, several provisioning mechanisms must be available:

- Provisioning of a domain name under which names can be published and services advertised
- Associating names that are subdomains of that name with hosts.
- Advertising services available on the local network by publishing resource records on those names.
- Distribution of names published in that namespace to servers that can be queried in order to resolve names
- Correct advertisement of name servers that can be queried in order to resolve names
- Timely removal of published names and resource records when they are no longer in use

Homenet adds the following considerations:

1. Some names may be published in a broader scope than others. For example, it may be desirable to advertise some homenet services to users who are not connected to the homenet. However, it is unlikely that all services published on the home network would be appropriate to publish outside of the home network. In many cases, no services will be appropriate to publish outside of the network, but the ability to do so is required.
2. Users cannot be assumed to be skilled or knowledgeable in name service operation, or even to have any sort of mental model of how these functions work. With the possible exception of policy decisions, all of the operations mentioned here must reliably function automatically, without any user intervention or debugging.
3. Even to the extent that users may provide input on policy, such as whether a service should or should not be advertised outside of the home, the user must be able to safely provide such input without having a correct mental model of how naming and service discovery work, and without being able to reason about security in a nuanced way.
4. Because user intervention cannot be required, naming conflicts must be resolved automatically, and, to the extent possible, transparently.
5. Where services are advertised both on and off the home network, differences in naming conventions that may vary depending on the user’s location must likewise be transparent to the end user.

6. Hosts that do not implement any homenet-specific capabilities must still be able to discover and access services on the homenet, to the extent possible.

7. Devices that provide services must be able to publish those services on the homenet, and those services must be available from any part of the homenet, not just the link to which the device is attached.

8. Homenet explicitly supports multihoming--connecting to more than one Internet Service Provider--and therefore support for multiple provisioning domains [9] is required to deal with situations where the DNS may give a different answer depending on whether caching resolvers at one ISP or another are queried.

9. Multihomed homenets may treat all service provider links as equivalent, or may treat some links as primary and some as backup, either because of differing transit costs or differing performance. Services advertised off-network may therefore be advertised for some links and not others.

10. To the extent possible, the homenet should support DNSSEC. If the homenet local domain is not unique, there should still be a mechanism that homenet-aware devices can use to bootstrap trust for a particular homenet.

In addition to these considerations, there may be a need to provide for secure communication between end users and the user interface of the home network, as well as to provide secure name validation (e.g., DNSSEC). Secure communications require that the entity being secured have a name that is unique and can be cryptographically authenticated within the scope of use of all devices that must communicate with that entity. Because it is very likely that devices connecting to one homenet will be sufficiently portable that they may connect to many homenets, the scope of use must be assumed to be global. Therefore, each homenet must have a globally unique identifier.

1.1. Existing solutions

Previous attempts to automate naming and service discovery in the context of a home network are able to function with varying degrees of success depending on the topology of the home network. For example, Multicast DNS [7] can provide naming and service discovery [8], but only within a single multicast domain.

The Domain Name System provides a hierarchical namespace [1], a mechanism for querying name servers to resolve names [2], a mechanism for updating namespaces by adding and removing names [4], and a

mechanism for discovering services [8]. Unfortunately, DNS provides no mechanism for automatically provisioning new namespaces, and secure updates to namespaces require pre-shared keys, which won't work for an unmanaged network. DHCP can be used to populate names in a DNS namespace; however at present DHCP cannot provision service discovery information.

Hybrid Multicast DNS [10] proposes a mechanism for extending multicast DNS beyond a single multicast domain.. However, it has serious shortcomings as a solution to the Homenet naming problem. The most obvious shortcoming is that it requires that every multicast domain have a separate name. This then requires that the homenet generate names for every multicast domain, and requires that the end user have a mental model of the topology of the network in order to guess on which link a given service may appear. [xxx is this really true at the UI?]

2. Terminology

This document uses the following terms and abbreviations:

HNR Homenet Router

ISP Internet Service Provider

GNRP Global Name Registration Provider

3. Homenet Naming Database

In order to resolve names, there must be a place where names are stored. There are two ways to go about this: either names are stored on the devices that own them, or they are stored in the network infrastructure. This isn't a clean division of responsibility, however. It's possible for the device to maintain change control over its own name, while still performing name resolution for that name in the network infrastructure.

If devices maintain change control on their own names, conflicts can arise. Two devices might present the same name, either because their default names or the same, or as a result of accidental. Devices can be attached to more than one link, in which case we want the same name to identify them on both networks. Although homenets are self-configuring, user customization is permitted and useful, and while some devices may provide a user interface for setting their name, it may be worthwhile to provide a user interface and underlying support for allowing the user to specify a device's name in the homenet infrastructure.

In order to achieve this, the Homenet Naming Database (HNDB) provides a persistent central store into which names can be registered.

3.1. Global Name

Every homenet must be able to have a name in the global DNS hierarchy which serves as the root of the zone in which the homenet publishes its public namespaces. Homenets that do not yet have a name in the global namespace use the homenet special-use top-level name [TBD1] as their "global name" until they are configured with a global name.

A homenet's global name can be a name that the homenet user has registered on their own in the DNS using a public DNS registrar. However, this is not required and, indeed, presents some operational challenges. It can also be a subdomain of a domain owned by one of the user's ISP, or managed by some DNS service provider that specifically provides homenet naming services.

For most end-users, the second or third options will be preferable. It will allow them to choose an easily-remembered homenet domain name under an easily-remembered service provider subdomain, and will not require them to maintain a DNS registration.

Homenets must support automatic configuration of the homenet global name in a secure manner, as well as manual configuration of the name. The solution must allow a user with a smartphone application or a user with a web browser to successfully configure the homenet's global name without manual data entry. The security implications of this process must be identified and, to the extent possible, addressed.

3.2. Local namespaces

Every homenet has two or more non-hierarchical local namespaces, one for names of hosts--the host namespace--and one or more for IP addresses--the address namespaces. A namespace is a database table mapping each of a set keys to its value. "Local" in this context means "visible to users of the homenet," as opposed to "public," meaning visible to anyone.

For the host namespace, the key is the set of labels in a name, excluding whatever labels represent the domain name of the namespace. So for example if the homenet's global name is "dog-pixel.example.com" and the name being looked up is "alice.dog-pixel.example.com", the key will be "alice".

The local namespace may be available both in the global DNS namespace and under the [TBD1] special-use name. The set of keys is the same

in both cases--in the above example, the name could be either 'alice.dog-pixel.example.com' or 'alice.[TBD1]'. Whichever one of the two representations is used, the key is simply 'alice'.

For each address namespace, the key is the locally-significant portion of the IP address. For example, if the local prefix assigned by an ISP is 2001:DB8:bee7::/48, the name of that address namespace will be '7.e.e.b.8.b.d.0.1.0.0.2.ip6.arpa'. An IP address of 2001:db8:bee7::1 would therefore yield a key of '1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0'

Every prefix in use on the homenet has an address namespace, whether its subdomain is delegated in the DNS or not. This includes any public or private IPv4 prefixes in use [3] as well as any ULA prefixes in use [5], which can't be delegated [6]. When the valid lifetime for a prefix that had been in use on the homenet ends, the address namespace for that prefix is discarded. Namespaces for prefixes that are manually configured, like IPv4 public prefixes and IPv4 private prefixes, persist as long as the prefix is configured. Since ULA prefixes have lifetimes, the lifetime rule applies to their address namespaces.

In all namespaces, the value that the key addresses is a sub-table containing one or more RRsets, each of which is identified by its RRtype. In the terminology of the DNS protocol, each of these namespaces is analogous to a DNS zone (but bear in mind that from the perspective of DNS queries, the namespace for names may appear to hosts connected to the homenet as two different zones containing identical data.

However, in addition to DNS zone data, each RRset also has two metadata flags: the public flag and the critical flag. The public flag indicates whether the data in this RRset should be publicly visible. The critical flag indicates whether the service should be advertised even on high-cost internet links.

Each RR that contains a name (e.g, a CNAME or SRV record) either contains a local name or a name in the public DNS. Local names can be subdomains of the homenet's global name, yet not be public, if no RRsets in the namespace for names is marked public. Local names can also be subdomains of [TBD1]. Names in the public DNS that are not subdomains of the homenet's global name can only be added by explicit action in one of the management interfaces described in Section 6.

Each local namespace is maintained as a distributed database with copies on every homenet router. No copy is the master copy. Although the local namespace is non-hierarchical, it is permissible for it to contain RRtypes that contain delegations. However, from an

operational perspective is is most likely better for the local namespace to be at the bottom of the delegation hierarchy, and so we do not recommend the use of such delegations.

3.3. Public namespaces

Every homenet has one or more public namespaces. These are subsets of the local namespaces with the following modifications:

1. Names with no RRsets whose public bits are set are not included in the public namespace.

2. RRs that contain IP addresses in the homenet's ULA prefix are omitted.

3. By default, RRs that contain IPv4 addresses are omitted, because IPv4 doesn't support renumbering. However, there should be a whitelist of IPv4 addresses that may be published, so that if the end user has static IPv4 addresses, those can be published. Private IPv4 addresses, however, are never published.

4. If an RRset is marked best-effort rather than critical, RRs containing IP addresses that have prefixes assigned by backup links are omitted.

5. If an RRset contains names, names that are subdomains of either the homenet's global name or [TBD1] are checked in the local host namespace to see if they are marked public. If not, they are omitted.

Because the public namespaces are subsets of the local namespaces, replication is not necessary: each homenet router automatically produces public namespaces by deriving them from the local namespaces using the above rules. Answers to queries in the public namespaces can be generated on demand. However, it may be preferable to maintain these namespaces as if they were DNS zones. This makes it possible to use DNS zone transfers to offload the contents of public zones to a secondary service provider, eliminating the need to handle arbitrary numbers of queries from off of the homenet.

A mechanism will be present that allows devices that have been configured to publicly advertise services to indicate to the homenet that the public bit and/or the backup bit will be set in RRsets that they publish.

3.4. Maintaining Namespaces

Homenets support three methods for maintaining local namespaces. These rely on Multicast DNS, DNS updates, and any of the management mechanisms mentioned in Section 6.

3.4.1. Multicast DNS

HNRs cooperate to maintain a DNS mirror of the set of names published by mDNS. This works similarly to the Multicast DNS Hybrid Proxy [10]. However, the DNSSD hybrid proxy exposes the topology of the network in which it operates to the user.

In order to avoid this, the homenet solution maintains a host namespace for each non-edge link in the homenet. Queries for names in the host namespace are looked up in the per-link host namespaces as well (and trigger mDNS queries as in the hybrid solution). When a cross-link name conflict is present for a name, the name is presented with a short modifier identifying the link.

For example, if two devices on two separate links both advertise the name 'janus' using mDNS, and the name 'janus' is not present in the host namespace, the two hosts' names are modified to, for example, 'janus-1' and 'janus-2'. If both devices present the human readable name 'Janus', then that name is presented as 'Janus (1)' and 'Janus (2)'. If the name 'janus' appears in the host namespace, then that name is presented just as 'janus'.

If a mDNS service advertises a name that appears in the host namespace, the HNR that hears the advertisement will defend the name, forcing the mDNS service to choose a different name.

This solution shares a problem that mdns hybrid has: user interfaces on hosts that present mDNS names in their mDNS format (e.g., 'janus.local') will not have a DNS entry for 'janus.local'. Connections to such hosts using the name presented in the UI will work when both hosts are attached to the same link, but not otherwise.

It is preferable that devices that are homenet-aware publish their names using DNS updates rather than using mDNS. mDNS is not supported as a query mechanism on homenets, other than in the sense that homeneds do not filter mDNS traffic on the local link. Service discovery is instead done using DNS service discovery [8]. This mechanism is supported on all modern devices that do service discovery, so there is no need to rely on mDNS.

3.4.2. DNS Update

DNS updates to the resolver on the local link are supported for adding names to local zones. When an update is received, if the name being updated does not exist, or if the update contains the same information as is present in the existing record, then the update is accepted. If a conflicting entry exists, the update is rejected.

This update procedure is available to hosts that implement DNS update for DNS service discovery, but are not homenet-aware. Hosts cannot delete records they have added, nor modify them; such records can only time out. Updates to server list records require that the host referenced by the update exist, and that the update come from that host. Such updates are additive, and are removed automatically when they become stale.

Hosts that are homenet-aware generate a KEY record containing a public key for which they retain the private key. They then publish their name in the host namespace, with whatever data they intend to publish on the name, and include the KEY record they have generated. The update is signed using SIG(0) on the provided key. If a record already exists, and does not contain the same KEY record, the update is refused. Otherwise it is accepted.

Homenet-aware hosts can then update their entries in the address table and in service tables by using their KEY record with SIG(0). Entries can be added _and_ deleted. However, only modifications to RRs that reference the name in the host namespace are allowed; all other RRs must be left as they are.

3.5. Recovery from loss

In principle the names in the zone aren’t precious. If there are multiple HNRs and one is replaced, the replacement recovers by copying the local namespaces and other info from the others. If all are lost, there are a few pieces of persistent data that need to be recovered:

- The global name
- The ZSK for both local namespaces
- Names configured statically through the UI

All other names were acquired dynamically, and recovery is simply a matter of waiting for the device to re-announce its name, which will happen when the device is power cycled, and also may happen when it

sees a link state transition. The hybrid mDNS implementation will also discover devices automatically when service queries are made.

Devices that maintain their state using DNS update, but that are not homenet-aware, may or may not update their information when they see a link state transition. Homenet-aware devices will update whenever they see a link-state transition, and also update periodically. When the Homenet configuration has been lost, HNRs advertise a special ND option that indicates that naming and service discovery on the homenet is in a recovery state. Homenet-aware devices will be sensitive to this ND option, and will update when it is seen.

Homenets will present an standard management API, reachable through any homenet router, that allows a device that has stored the DNSSEC ZSK and KSK to re-upload it when it has been lost. This is safest solution for the end user: the keys can be stored on some device they control, under password protection.

ZSKs and KSKs can also be saved by the ISP or GNRP and re-installed using one of the management APIs. This solution is not preferable, since it means that the end user's security is reliant on the security of the GNRP or ISP's infrastructure.

If the ZSK and KSK are lost, they can be regenerated. This requires that the homenet's global name change: there is no secure way to re-key in this situation. Once the homenet has been renamed and re-keyed, all devices that use the homenet will simply see it as a different homenet.

3.6. Well-known names

Homenets serve a zone under the special-use top-level name [TBD2] that answers queries for local configuration information and can be used to advertise services provided by the homenet (as opposed to services present on the homenet). This provides a standard means for querying the homenet that can be assumed by management functions and homenet clients. A registry of well-known names for this zone is defined in IANA considerations (Section 9). Names and RRs in this zone are only ever provided by the homenet--this is not a general purpose service discovery zone.

All resolvers on the homenet will answer questions about names in this zone. Entries in the zone are guaranteed not to be globally unique: different homenets are guaranteed to give independent and usually different answers to queries against this zone. Hosts and services that use the special names under this TLD are assumed to be aware that it is a special TLD. If such hosts cache DNS entries, DNS

entries under this TLD are discarded whenever the host detects a network link state transition.

The uuid.[TBD2] name contains a TXT RR that contains the UUID of the homenet. Each homenet generates its own distinct UUID; homenet routers on any particular homenet all use the same UUID, which is agreed upon using HNCP. If the homenet has not yet generated a UUID, queries against this name will return NXDOMAIN.

The global-name.[TBD2] name contains a PTR record that contains the global name of the homenet. If the homenet does not have a global name, queries against this name will return NXDOMAIN.

The global-name-register.[TBD2] name contains one or more A and/or AAAA records referencing hosts (typically HNRs) that provide a RESTful API over HTTP that can be used to register the global name of the homenet, once that name has been configured.

The all-resolver-names.[TBD2] name contains an NS RRset listing a global name for each HNR. It will return NXDOMAIN if the homenet has no global name. These names are generated automatically by each HNR when joining the homenet, or when a homenet to which the HNR is connected establishes a global name.

4. Name Resolution

4.1. Configuring Resolvers

Hosts on the homenet receive a set of resolver IP addresses using either DHCP or RA. IPv4-only hosts will receive IPv4 addresses of resolvers, if available, over DHCP. IPv6-only hosts will receive resolver IPv6 addresses using either stateful (if available) or stateless DHCPv6, or through the domain name option in router advertisements. All homenet routers provide resolver information using both stateless DHCPv6 and RA; support for stateful DHCPv6 and DHCPv4 is optional, however if either service is offered, resolver addresses will be provided using that mechanism as well. Resolver IP addresses will always be IP addresses on the local link: every HNR is required to provide name resolution service. This is necessary to allow DNS update using presence on-link as a mechanism for rejecting off-network attacks.

4.2. Configuring Service Discovery

DNS-SD uses several default domains for advertising local zones that are available for service discovery. These include the ’.local’ domain, which is searched using mDNS, and also the IPv4 and IPv6 reverse zone corresponding to the prefixes in use on the local

network. For the homenet, no support for queries against the ".local" zone is provided by HNRs: a ".local" query will be satisfied or not by services present on the local link. This should not be an issue: all known implementations of DNSSD will do unicast queries using the DNS protocol.

Service discovery is configured using the technique described in Section 11 of DNS-Based Service Discovery [8]. HNRs will answer domain enumeration ueries against every IPv4 address prefix advertised on a homenet link, and every IPv6 address prefix advertised on a homenet link, including prefixes derived from the homenet's ULA(s). Whenever the "<domain>" sequence appears in this section, it references each of the domains mentioned in this paragraph.

Homenets advertise the availability of several browsing zones in the "b._dns_sd.<domain>" subdomain. The zones advertised are the "well known" zone (TBD2) and the zone containing the local namespace. If the global name is available, only that name is advertised for the local namespace; otherwise [TBD1] is advertised. Similarly, if the global name is available, it is advertised as the default browsing and service registration domain under "db._dns_sd.<domain>", "r._dns_sd.<domain>", "dr._dns_sd.<domain>" and "lb._dns_sd.<domain>"; otherwise, the name [TBD1] is advertised as the default.

4.3. Resolution of local namespaces

The local namespace appears in two places, under [TBD1] and, if the homenet has a global name, under the global name. Resolution from inside the homenet yields the contents of the local namespaces; resolution outside of the homenet yields the contents of the public namespaces. If there is a global name for the homenet, RRs containing names in both instances of the local namespace are qualified with the global name; otherwise they are qualified with [TBD1].

4.4. Service Discovery Resolution

Because homenets provide service discovery over DNS, rather than over mDNS, support for DNS push notifications [11]. When a query arrives for a local namespace, and no data exists in that namespace to answer the query, that query is retransmitted as an mDNS query. Data that exists to answer the query in mdns cached namespaces does not prevent an mDNS query being issued.

If there is data available to answer the query in the host namespace or any of the dnssd cached namespaces, that data is aggregated and

returned immediately. If the host that sent the query requested push notification, then any mDNS responses that come in subsequent to the initial answer are sent as soon as they are received, and also added to the cache. This means that if a name has been published directly using DNS, no mDNS query for that name is ever generated.

4.5. Local and Public Zones

The homenet's global name serves both as a unique identifier for the homenet and as a delegation point in the DNS for the zone containing the homenet's forward namespace. There are two versions of the forward namespace: the public version and the private version. Both of these versions of the namespace appear under the global name delegation, depending on which resolver a host is querying.

The homenet provides two versions of the zone. One is the public version, and one is the local version. The public version is never visible on the homenet (could be an exception for a guest net). The public version is available outside of the homenet. The local version is visible on the homenet. Whenever the zone is updated, it is signed with the ZSK. Both versions of the zone are signed; the local signed version always has a serial number greater than the public signed version. [we want to not re-sign the public zone if no public names in the private zone changed.]

This dual publication model relies on hosts connected to the homenet using the local resolver and not some external resolver. Hosts that use an external resolver will see the public version of the namespace. From a security UI design perspective, allowing queries from hosts on the homenet to resolvers off the homenet is risky, and should be prevented by default. This is because if the user sees inconsistent behavior on hosts that have external resolvers configured, they may attempt to fix this by making all local names public. If an alternate external resolver is to be used, it should be configured on the homenet, not on the individual host.

One way to make this work is to intercept all DNS queries to non-homenet IP addresses, check to see if they reference the local namespace, and if so resolve them locally, answering as if from the remote cache. If the query does not reference a local namespace, and is listed as "do not forward" in RFC 6761 or elsewhere, it can be sent to the intended cache server for resolution without any special handling for the response. This functionality is not required for homenet routers, but is likely to present a better user experience.

4.6. DNSSEC Validation

All namespaces are signed using the same ZSK. The ZSK is signed by a KSK, which is ideally kept offline. Validation for the global name is done using the normal DNSSEC trust hierarchy. Validation for the [TBD1] and [TBD2] zones can be done by fetching the global name from the [TBD2] zone, fetching and validating the ZSK using DNSSEC, and then using that as a trust anchor.

Only homenet-aware hosts will be able to validate names in the [TBD1] and [TBD2] zones. The homenet-aware host validates non-global zones by determining which homenet it is connected to querying the uuid.[TBD2] and global-name.[TBD2] names. If there is an answer for the global-name.[TBD2] query, validation can proceed using the trust anchor published in the zone that delegates the global name. If only the uuid is present, then the homenet-aware host can use trust-on-first-use to validate that an answer came from the homenet that presented that UUID. This provides only a limited degree of trustworthiness.

4.7. Support for Multiple Provisioning Domains

Homenets must support the Multiple Provisioning Domain Architecture [9]. In order to support this architecture, each homenet router that provides name resolution must provide one resolver for each provisioning domain (PvD). Each homenet router will advertise one resolver IP address for each PvD. DNS requests to the resolver associated with a particular PvD, e.g. using RA options [12] will be resolved using the external resolver(s) provisioned by the service provider responsible for that PvD.

The homenet is a separate provisioning domain from any of the service providers. The global name of the homenet can be used as a provisioning domain identifier, if one is configured. Homenets should allow the name of the local provisioning domain to be configured; otherwise by default it should be "Home Network xxx", where xxx is the generated portion of the homenet’s ULA prefix, represented as a base64 string.

The resolver for the homenet PvD is offered as the primary resolver in RAs and through DHCPv4 and DHCPv6. When queries are made to the homenet-PvD-specific resolver for names that are not local to the homenet, the resolver will use a round-robin technique, alternating between service providers with each step in the round-robin process, and then also between external resolvers at a particular service provider if a service provider provides more than one. The round-robining should be done in such a way that no service provider is preferred, so if service provider A provides one caching resolver

(A), and service provider B provides two (B1, B2), the round robin order will be (A, B1, A, B2), not (A, B1, B2).

Every resolver provided by the homenet, regardless of which provisioning domain it is intended to serve, will accept updates for services in the local service namespace from hosts on the local link.

4.8. Using the Local Namespace While Away From Home

Homenet routers do not answer unauthenticated DNS queries from off the local network. However, some applications may benefit from the ability to resolve names in the local namespace while off-network. Therefore hosts connected to the homenet can register keys in the host namespace using DNS Update. Such keys must be validated by the end user before queries against the local namespace can be authenticated using that key. A host that will make remote queries to the local namespace caches the names of all DNS servers on the homenet by querying all-resolver-names.[TBD2].

Hosts that require name resolution from the local network must have a stub resolver configured to contact the dns server on one or more routers in the homenet when resolving names in the host or address namespaces. To do this, resolvers must know the global name of the local namespace, which they can retain from previous connections to the homenet.

The homenet may not have a stable IP address, so such resolvers cannot merely cache the IP address of the homenet routers. Instead, they cache the NS record listing the HNRs and use those names to determine the IP addresses of the homenet routers at the time of resolution. Such IP addresses can be safely cached for the duration of the TTL of the A or AAAA record that contained them. The names of the homenet router DNS servers should be randomly generated so that they can’t be guessed by off-network attackers.

To make a homenet DNS query, the host signs the request using SIG(0) with the key that they registered to the homenet. The homenet router first checks the question in the query for validity: it must be a subdomain of the global name. The homenet router then checks the name of the signing key against the list of cached, validated keys; if that key is cached and validated, then the homenet router attempts to validate the SIG(0) signature using that key. If the signature is valid, then the homenet router answers the query. If the zone doesn’t have a trust anchor in the parent zone, the responding server signs the answer with its own ZSK. The resolver that sent the query validates the response using DNSSEC if possible, and otherwise using the ZSK directly.

5. Publishing the Public Namespace

5.1. Acquiring the Global Name

There are two ways to acquire a global name: the end-user can register a domain name using a public domain name registry, or the end-user can be assigned a subdomain of a registered domain by a homenet global name service provider. We will refer to this as the Global Name Registration Provider [GNRP]. In either case, the registration process can either be manual or automatic. Homenet routers support automatic registration regardless of the source of the homenet's global name, using a RESTful API.

5.2. Hidden Primary/Public Secondaries

The default configuration for a homenet's external name service is that the primary server for the zone is not published in an NS record in the zone's delegation. Instead, the GNRP provides authoritative name service for the zone. Whenever the public zone is updated, the hidden primary sends NOTIFY messages to all the secondaries, using the zone's ZSK to sign the message.

When any of the GNRP secondary servers receives a notify for the zone, it checks to see that the notify is signed with a valid ZSK for that zone. If so, it contacts the IP address from which the NOTIFY was send and initiates a zone transfer. Using this IP address avoids renumbering issues. Upon finishing the zone transfer, the zone is validated using each ZSK used to sign it. If any validation fails, the new version of the zone is discarded. If updates have been recevied, but no valid updates received, over a user-settable interval defaulting to a day (or?), the GNRP will communicate to the registered user that there is a problem.

The reverse zone for any prefix delegated by an ISP should be delegated by that ISP to the home gateway to which the delegation was sent. The list of secondaries for that zone is sent to the home gateway using DHCPv6 prefix delegation. The ZSK is announced to the ISP in each DHCP PD message sent by the home gateway. Whenever an update is made to this zone, the home gateway sends a NOTIFY to each of the listed secondaries for the delegation, and updates occur as described above. Once the delegation is established, the ISP will not accept a different ZSK unless the prefix and its delegated zone are reassigned.

5.3. PKI security

All communication with the homenet using HTTP is encrypted using opportunistic security. If the homenet is configured with PKI, then the PKI certificate is used. Homenets should automatically acquire a PKI certificate when a global name is established. This certificate should be published in a TLSA record in the host namespace on any hostnames on which HTTP service is offered by HNRs.

5.4. Renumbering

The homenet may renumber at any time. IP address RRs published in any namespace must never have a TTL that is longer than the valid lifetime for the prefix from which the IP address was allocated. If a particular ISP has deprecated a prefix (its preferred lifetime is zero), IP addresses derived from that prefix are not published in the any namespace. If more than one prefix is provided by the same ISP and some have different valid lifetimes, only IP addresses in the prefix or prefixes with the longest valid lifetime are published.

5.5. ULA

Homenets have at least one ULA prefix. If a homenet has two ULA prefixes, and one is deprecated, addresses in the second ULA prefix are not published. The default source address selection algorithm ensures that if a service is available on a ULA, that ULA will be used rather than the global address. Therefore, no special effort is made in the DNS to offer only ULAs in response to local queries.

6. Management

6.1. End-user management

Homenets provide two management mechanisms for end users: an HTTP-based user interface and an HTTP-based RESTful API [tbw].

Homenets also provide a notification for end users. By default, when an event occurs that requires user attention, the homenet will attract the user’s attention by triggering captive portal detection on user devices. Users can also configure specific devices to received management alerts using the RESTful management API; in this case, no captive portal notification is performed.

6.2. Central management

Possibly can be done mostly through RESTful API, but might want Netconf/Yang as well. Should be possible to have the local namespace mastered on an external DNS auth server, e.g. in case a bunch of HNRs

are actually set up in an org, or in case an ISP wants to provide a service package for users who would rather not have an entirely self-operating network.

7. Privacy Considerations

Private information must not leak out as a result of publishing the public namespace. The 'public' flag on RRsets in homenet-managed namespaces prevents leakage of information that has not been explicitly marked for publication.

The privacy of host information on the local net is left to hosts. Various mechanisms are available to hosts to ensure that tracking does not occur if it is not desired. However, devices that need to have special permission to manage the homenet will inevitably reveal something about themselves when doing so. It may be possible to use something like HTTP token binding[13] to mitigate this risk.

8. Security Considerations

There are some clear issues with the security model described in this document, which will be documented in a future version of this section. A full analysis of the avenues of attack for the security model presented here have not yet been done, and must be done before the document is published.

9. IANA considerations

IANA will add a new registry titled Homenet Management Well-Known Names, which initially contains:

uuid Universally Unique Identifier--TXT record containing, in base64 encoding, a stable, randomly generated identifier for the homenet that is statistically unlikely to be shared by any other homenet.

global-name The homenet's global name, represented as a PTR record to that name.

global-name-register The hostname of the homenet's global name registry service, with A and/or AAAA records.

all-resolver-names A list of all the names of the homenet's resolvers for the homenet PvD, represented as an RRset containing one or more PTR records.

The IANA will allocate two names out of the Special-Use Domain Names registry:

TBD1 Suggested value: "homenet"

TBD2 Suggested value: "_hnsd"

10. Normative References

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

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

[3] 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>.

[4] 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>.

[5] 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>.

[6] Andrews, M., "Locally Served DNS Zones", BCP 163, RFC 6303, DOI 10.17487/RFC6303, July 2011, <http://www.rfc-editor.org/info/rfc6303>.

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

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

[9] Anipko, D., Ed., "Multiple Provisioning Domain Architecture", RFC 7556, DOI 10.17487/RFC7556, June 2015, <http://www.rfc-editor.org/info/rfc7556>.

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

[11] Pusateri, T. and S. Cheshire, "DNS Push Notifications", draft-ietf-dnssd-push-07 (work in progress), April 2016.

[12] Korhonen, J., Krishnan, S., and S. Gundavelli, "Support for multiple provisioning domains in IPv6 Neighbor Discovery Protocol", draft-ietf-mif-mpvd-ndp-support-03 (work in progress), February 2016.

[13] Popov, A., Nystrom, M., Balfanz, D., Langley, A., and J. Hodges, "Token Binding over HTTP", draft-ietf-tokbind-https-05 (work in progress), July 2016.

Author's Address

Ted Lemon
Nominum, Inc.
800 Bridge Parkway
Redwood City, California 94065
United States of America

Phone: +1 650 381 6000
Email: ted.lemon@nominum.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

Table of Contents

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. 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., Tschofenig, 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-F4813999999
 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