Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: May 14, 2015

S. Cheshire
Apple Inc.

November 10, 2014

# Hybrid Unicast/Multicast DNS-Based Service Discovery
draft-ietf-dnssd-hybrid-00

## 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) of services that are outside the local link. Using a very large local link with thousands of hosts improves service discovery, but at the cost of large amounts of multicast traffic.

Performing DNS-Based Service Discovery using purely Unicast DNS is more efficient, but requires configuration of DNS Update keys on the devices offering the services, which can be onerous for simple devices like printers and network cameras.

Hence a compromise is needed, that provides easy service discovery without requiring either large amounts of multicast traffic or onerous configuration.

## Status of this Memo



## Copyright Notice

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:

- o when any host A from that set sends a packet to any other host B in that set, using unicast, multicast, or broadcast, the entire link-layer packet payload arrives unmodified, and

- o a broadcast sent over that link by any host from that set of hosts can be received by every other host in that set

The link-layer *header* may be modified, such as in Token Ring Source Routing [802.5], but not the link-layer *payload*. In particular, if any device forwarding a packet modifies any part of the IP header or IP payload then the packet is no longer considered to be on the same link. This means that the packet may pass through devices such as repeaters, bridges, hubs or switches and still be considered to be on the same link for the purpose of this document, but not through a device such as an IP router that decrements the IP TTL or otherwise modifies the IP header.

## 3. Hybrid Proxy Operation

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 "4th Floor.Building 1.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.

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.

### 3.1. Domain Enumeration

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, today). The "lb" domain is usually the same as the "db" domain.

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

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

An Answer Section Resource Record consists of:
- o Owner name, encoded as a two-byte compression pointer
- o Two-byte rrtype (type PTR)
- o Two-byte rrclass (class IN)
- o Four-byte ttl
- o Two-byte rdlength
- o rdata (domain name, up to 256 bytes)

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

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

### 3.2. Delegated Subdomain for LDH Host Names

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

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 are often 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 software, 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 not advisable (because of the space in "Building 1").

To accomodate this difference in allowable characters, a Hybrid Proxy MUST support having two subdomains delegated to it, one to be used for host names (names of ’A’ and ’AAAA’ address records), which is restricted to the traditional letter-digit-hyphen rules, and another to be used for other records (including the PTR, SRV and TXT records used by DNS-SD), which is allowed to be arbitrary Net-Unicode text [RFC5198].

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").

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

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.

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 IPv4 or IPv6 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.

### 3.4. 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.

#### 3.4.1. 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 Unicast DNS client on a remote link does not get to participate in these Multicast DNS cache coherency mechanisms on the local link. For Unicast DNS requests received without any LLQ option the DNS TTLs reported in the resulting Unicast DNS response SHOULD be capped to be no more than ten seconds. For received Unicast DNS requests that contain an LLQ option, the Multicast DNS record’s TTL SHOULD be returned unmodified, because the LLQ notification channel exists to inform the remote client as records come and go. For further details about the LLQ option, see Section 3.5.

#### 3.4.2. 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 IPv4 link-local addresses [RFC3927] and IPv6 link-local addresses [RFC4862] 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.

### 3.4.3. 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. 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. Therefore, a Hybrid Proxy that is aware of the specifics of an application-layer protocol such as Apple’s AirPrint (which uses IPP) can elide unnecessary key/value pairs from the DNS-SD TXT record for better network efficiency.

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.

As in other similar situations, these layer violations optional. They are done only for efficiency reasons, and are not required for correct operation. A Hybrid Proxy can operate solely at the mDNS layer, without any knowledge of semantics at the DNS-SD layer or above.

### 3.5. 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 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 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.

These dilemmas are solved by use of DNS Long-Lived Queries (DNS LLQ) [I-D.sekar-dns-llq]. The Hybrid Proxy responds immediately to the Unicast DNS query 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 update messages. The timeliness of such LLQ updates is limited only by the timeliness of the device responding to the Multicast DNS query. If the Multicast DNS device responds quickly, then the LLQ update is delivered quickly. If the Multicast DNS device responds slowly, then the LLQ update is delivered slowly. The benefit of using LLQ 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 LLQs solves this dilemma: even very late responses are not lost; they are delivered in subsequent LLQ update messages.

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

The first factor is whether the query from the client included the LLQ 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 option are received directly from the client (see Section 3.5.1). Queries containing no LLQ option are generally received via the client's configured recursive (caching) name server.

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

- o No LLQ option; no answer in cache:
  Do local mDNS query up to three times, return answers if received, otherwise return negative response if no answer after three tries.
  DNS TTLs in responses are capped to at most ten seconds.

- o No LLQ 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.)

- o Query contains LLQ option; no answer in cache:
  As above, do local mDNS query up to three times, and return answers if received.
  If no answer after three tries, return negative response.
  (Reasoning: We don't need to rush to send an empty answer.)
  In both cases the query remains active for as long as the client maintains the LLQ state, and if mDNS answers are received later, LLQ update messages are sent.
  DNS TTLs in responses are returned unmodified.

- o Query contains LLQ option; at least one answer in cache:
  As above, send response right away to minimise delay.
  The query remains active for as long as the client maintains the LLQ state, and if additional mDNS answers are received later, LLQ 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.

#### 3.5.1. Discovery of LLQ Service

To issue LLQ 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].

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 port 5352, but since SRV records are used, the LLQ 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.

## 4. 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.

### 4.1. 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 doing unicast DNS queries to the IETF DNS servers.

### 4.2. 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. It is possible that the IETF may choose to standardize a different or better Long-Lived Query mechanism. In that case, the pragmatic deployment approach would be 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 a new IETF Standard Long-Lived Query mechanism (as the future 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.

### 4.3. Not Yet Implemented

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.

## 5. IPv6 Considerations

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

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

# 6. Security Considerations

## 6.1. 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].

## 6.2. 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.

## 6.3. 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.

## 7. 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].

## 8. IANA Considerations

This document has no IANA Considerations.

## 9. 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 Andrew Yourtchenko for comments about privacy issues. [Partial list; more names to be added.]

## 10. References

### 10.1. 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.

[RFC1918] Rekhter, Y., Moskowitz, R., Karrenberg, D., Groot, G., and E. Lear, "Address Allocation for Private Internets", BCP 5, RFC 1918, February 1996.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC3927] Cheshire, S., Aboba, B., and E. Guttman, "Dynamic Configuration of IPv4 Link-Local Addresses", RFC 3927, May 2005.

[RFC4862] Thomson, S., Narten, T., and T. Jinmei, "IPv6 Stateless Address Autoconfiguration", RFC 4862, September 2007.

[RFC5198] Klensin, J. and M. Padlipsky, "Unicode Format for Network Interchange", RFC 5198, March 2008.

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

## 10.2. Informative References

[HOME] Cheshire, S., "Special Use Top Level Domain 'home'", draft-cheshire-homenet-dot-home (work in progress), November 2014.

[IPR2119] "Apple Inc.'s Statement about IPR related to Hybrid Unicast/Multicast DNS-Based Service Discovery", <https://datatracker.ietf.org/ipr/2119/>.

[RFC2136] Vixie, P., Thomson, S., Rekhter, Y., and J. Bound, "Dynamic Updates in the Domain Name System (DNS UPDATE)", RFC 2136, April 1997.

[RFC3007] Wellington, B., "Secure Domain Name System (DNS) Dynamic Update", RFC 3007, November 2000.

[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

DNSSD
Internet-Draft
Intended status: Informational
Expires: September 5, 2015

A. Sullivan
Dyn

March 4, 2015

# On Interoperation of Labels Between mDNS and DNS
draft-ietf-dnssd-mdns-dns-interop-00

## Abstract

Despite its name, DNS-Based Service Discovery can use naming systems other than the Domain Name System when looking for services. Different name systems use different conventions for the characters allowed in any name. In order for DNS-SD to be used effectively in environments where multiple different name systems are in use, it is important to attend to differences in the underlying technology. This memo presents an outline of the requirements for selection of labels for mDNS and DNS when they are expected to interoperate in this manner.

## Status of This Memo



## Copyright Notice

Table of Contents



## 1. Introduction

DNS-Based Service Discovery (DNS-SD, [RFC6763]) specifies a mechanism for discovering services using queries both to the Domain Name System (DNS, [RFC1034], [RFC1035]) and to Multicast DNS (mDNS, [RFC6762]). Conventional use of the DNS generally follows the host name rules [RFC0952] for labels -- the so-called LDH rule. That convention is the reason behind the development of Internationalized Domain Names for Applications (IDNA2008, [RFC5890], [RFC5891], [RFC5892], [RFC5893], [RFC5894], [RFC5895]). It is worth noting that the LDH rule is a convention, and not a strict rule of the DNS. It is assumed to be true widely enough, however, that in many circumstances names cannot be used unless they cleave to the LDH rule.

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

Users of applications are, of course, frequently unconcerned with (not to say oblivious to) the name-resolution system(s) in service at

any given moment, and are inclined simply to use the same names in different contexts. As a result, the same string might be tried as a name using different name resolution technologies. If DNS-SD is to be used in an environment where both mDNS and DNS are to be queried for services, then some parts of the names to be queried will need to be compatible with the rules and conventions for both DNS and mDNS.

One approach to interoperability under these circumstances is to use a single operational convention for names under the different naming systems. This memo assumes such a use profile, and outlines what is necessary to make it work.

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

### 1.1. Conventions and terms used in this document

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

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

The term "owner name" (common to the DNS vernacular) is used here to apply not just to the names to be looked up in the DNS, but to any name that might be looked up either in the DNS or using mDNS.

## 2. Requirements for a profile for label interoperation

Any interoperability between mDNS and DNS will require interoperability across some of the portions of a DNS-SD Service Instance Name (see Section 3) that are implicated in regular mDNS and

DNS lookups. Only some portions are implicated. In any case, if a given portion is implicated, the profile will need to apply to all labels in that portion.

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

Because the profile will need to apply to names that might need to interoperate with names in the DNS, and because mDNS permits labels that IDNA does not, the profile might reduce the labels that could be used with mDNS. Consequently, some recommendations from [RFC6763] will not really be possible to implement using names subject to the profile. In particular, [RFC6763], section 4.1.3 recommends that labels always be stored and communicated as UTF-8, even in the DNS. Because IDNA2008 libraries will treat any Unicode-encoded labels as candidate U-labels and attempt to perform resolution in A-label form, the advice to store and transmit labels as UTF-8 in the DNS is likely to encounter problems. In particular, the <Domain> part of a Service Instance Name is unlikely to be found in its UTF-8 form in the public DNS tree for zones that are using IDNA2008. By contrast, mDNS normally uses UTF-8.

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

## 3. DNS-SD portions

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

### 3.1. The <Instance> Portion of the Service Instance Name

[RFC6763] is clear that the <Instance> portion of the Service Instance Name is intended for presentation to users, and therefore

virtually any character is permitted in it. There are two ways that a profile might address this portion.

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

Instead, DNS-SD implementations can intercept the <Instance> portion of a Service Instance Name and ensure that those labels are never handed to IDNA-aware resolvers that might attempt to convert these labels into A-labels. Under this approach, the DNS-SD <Instance> portion works as it always does, but at the cost of using special resolution code built into the DNS-SD system.

### 3.2. The <Service> Portion of the Service Instance Name

DNS-SD includes a <Service> component in the Service Instance Name. This component is not really user-facing data, but is instead control data embedded in the Service Instance Name. This component includes so-called "underscore labels", which are labels prepended with U+005F (_). The underscore label convention was established by DNS SRV ([RFC2782]) for identifying metadata inside DNS names. A system-wide resolver (or DNS middlebox) that cannot handle underscore labels will not work with DNS-SD at all, so it is safe to suppose that such resolvers will not attempt to do special processing on these labels. Therefore, the <Service> portion of the Service Instance Name will not be subject to the profile.

### 3.3. The <Domain> Portion of the Service Instance Name

The <Domain> portion of the Service Instance Name forms an integral part of the QNAME submitted for DNS resolution, and a system-wide resolver that is IDNA2008-aware is likely to interpret labels with UTF-8 in the QNAME as candidates for IDNA2008 processing. Operators of Internationalized Domain Names will frequently publish them in the DNS as A-labels. Therefore, these labels will need to be subject to the profile. DNS-SD implementations ought to identify the <Domain> portion of the Service Instance Name and treat it subject to IDNA2008 in case the domain is to be queried from the global DNS. In the event that the <Domain> portion of the Service Instance Name fails to resolve, it is acceptable to substitute labels with plain UTF-8, starting at the lowest label in the DNS tree and working toward the root. This approach different to the rule for resolution published

in [RFC6763], because it privileges IDNA2008-compatible labels over UTF-8 labels.

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

1. It is possible the names may be in the DNS in UTF-8, and RFC 6763 already specifies a fallback strategy of progressively attempting first the U-label lookup and then the A-label lookup.

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

The first of these is rejected because it represents a potentially significant increase in DNS lookup traffic for no value. It is possible for a DNS-SD application to identify the <Domain> portion of the Service Instance Name. The standard way to publish IDNs on the Internet uses IDNA. Therefore, additional lookups should not be encouraged. When [RFC6763] was published, the bulk of IDNs were lower in the tree, but now that there are internationalized labels in the root zone, it seems reasonable to use only the single lookup strategy.

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

## 4. Acknowledgements

The author gratefully acknowledges the insights of Stuart Cheshire, Kerry Lynn, and Dave Thaler.

## 5. IANA Considerations

This memo makes no requests of IANA.

## 6. Security Considerations

This memo presents some requirements for future development, but does not specify anything. Therefore, it has no implications for security.

## 7. Informative References

[RFC0952] Harrenstien, K., Stahl, M., and E. Feinler, "DoD Internet host table specification", RFC 952, October 1985.

[RFC1034] Mockapetris, P., "Domain names - concepts and facilities", STD 13, RFC 1034, November 1987.

[RFC1035] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, November 1987.

[RFC2782] Gulbrandsen, A., Vixie, P., and L. Esibov, "A DNS RR for specifying the location of services (DNS SRV)", RFC 2782, February 2000.

[RFC5198] Klensin, J. and M. Padlipsky, "Unicode Format for Network Interchange", RFC 5198, March 2008.

[RFC5890] Klensin, J., "Internationalized Domain Names for Applications (IDNA): Definitions and Document Framework", RFC 5890, August 2010.

[RFC5891] Klensin, J., "Internationalized Domain Names in Applications (IDNA): Protocol", RFC 5891, August 2010.

[RFC5892] Faltstrom, P., "The Unicode Code Points and Internationalized Domain Names for Applications (IDNA)", RFC 5892, August 2010.

[RFC5893] Alvestrand, H. and C. Karp, "Right-to-Left Scripts for Internationalized Domain Names for Applications (IDNA)", RFC 5893, August 2010.

[RFC5894] Klensin, J., "Internationalized Domain Names for Applications (IDNA): Background, Explanation, and Rationale", RFC 5894, August 2010.

[RFC5895] Resnick, P. and P. Hoffman, "Mapping Characters for Internationalized Domain Names in Applications (IDNA) 2008", RFC 5895, September 2010.

[RFC6672] Rose, S. and W. Wijngaards, "DNAME Redirection in the DNS", RFC 6672, June 2012.

[RFC6762] Cheshire, S. and M. Krochmal, "Multicast DNS", RFC 6762, February 2013.

[RFC6763] Cheshire, S. and M. Krochmal, "DNS-Based Service Discovery", RFC 6763, February 2013.

## Appendix A. Change History

### A.1. draft-ietf-dnssd-mdns-dns-interop-00

1st WG version

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

### A.2. draft-sullivan-dnssd-mdns-dns-interop-01

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

### A.3. draft-sullivan-dnssd-mdns-dns-interop-00

Initial version.

## Author’s Address

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

Email: asullivan@dyn.com

Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: September 10, 2015

T. Pusateri
Seeking affiliation
S. Cheshire
Apple Inc.

March 9, 2015

# DNS Push Notifications
# draft-ietf-dnssd-push-00

## Abstract

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

## Status of This Memo



## Copyright Notice

Table of Contents



## 1. Introduction

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

### 1.1. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "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 [XEP-0060] 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.

## 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. Adopting this existing syntax and semantics for DNS Push Notifications allows for messages going in the other direction, from server to client, to communicate changes to a zone. The client first must subscribe 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].

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 on the server, and that concept has no application when it comes to an authoritative server telling a client of changes to DNS records.

## 4. Transport

Implementations of DNS Update [RFC2136] MAY use either User Datagram Protocol (UDP) [RFC0768] or Transmission Control Protocol (TCP) [RFC0793] as the transport protocol, in keeping with the historical precedent that DNS queries must first be sent over UDP [RFC1123]. This requirement to use UDP has subsequently been relaxed [RFC5966]. DNS Push Notification is defined only for TCP. DNS Push Notification clients MUST use TCP.

Either end of the TCP connection can terminate all of the subscriptions on that connection by simply closing the connection abruptly with a TCP RST. (An individual subscription is terminated by sending an UNSUBSCRIBE message for that specific subscription.)

If a client closes the connection, it is signaling that it is no longer interested in receiving updates to any of the records it has subscribed. It is informing the server that the server may release all state information it has been keeping with regards to this client. This may occur because the client computer has been disconnected from the network, has gone to sleep, or the application requiring the records has terminated.

If a server closes the connection, it is informing the client that it can no longer provide updates for the subscribed records. This may occur because the server application software or operating system is restarting, the application terminated unexpectedly, the server is

undergoing maintenance procedures, or the server is overloaded and can no longer provide the information to all the clients that wish to receive it. The client can try to re-subscribe at a later time or connect to another server supporting DNS Push Notifications for the zone.

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.

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. Additional security measures such as authentication during TLS negotiation MAY also be employed to increase the trust relationship between client and server. 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].

## 5. State Considerations

Each DNS Push Notification server is capable and 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 connecting 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. The exchange terminates when either end closes the TCP connection with a TCP RST.

### 6.1. Discovery

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

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

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

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

4. Once the SOA is known, the client sends a DNS query with type SRV [RFC2782] for the record name "_dns-push._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 9.2 [I-D.ietf-dane-srv].

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" is 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.

If a server closes a DNS Push Notification subscription connection, the client 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 is a wildcard subscription to changes for the given name for any type and/or class, as appropriate.

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

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

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

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

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

ARCOUNT MUST be zero, and the Additional Section MUST be empty. Any records in the Additional Section MUST be silently ignored.

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

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 |
| NOTIMP | 4 | Server does not implement DNS Push Notifications |
| REFUSED | 5 | Server refuses to process request for policy or security reasons |

Table 1: Response codes

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

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

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

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

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

ARCOUNT MUST be zero, and the Additional Section MUST be empty.

Any records in the Additional Section MUST be silently ignored.

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.

A client MUST not send a SUBSCRIBE message that duplicates the name, type and class of an existing active subscription. 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.

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.

A client may SUBSCRIBE to records that are unknown to the server at the time of the request 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.

### 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. If a server receives such an UNSUBSCRIBE message this is an error and the server MUST immediately close the TCP connection.

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

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

### 6.4. DNS Push Notification Update Messages

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

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

The following figure shows the existing DNS Update header format:

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

Figure 1

For DNS Push Notifications the following rules apply:

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

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

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

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

ADCOUNT MUST be zero, and the Additional Data Section MUST be empty. Any records in the Additional Data Section MUST be silently ignored.

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 an individual RR to a name:
TTL, CLASS, TYPE, RDLENGTH and RDATA specifies the RR being added.
```

Upon reception of a Push Notification Update Message, the client receiving the message 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 was not ANY (255) then the TYPE of the record must match the QTYPE given in the SUBSCRIBE request. If the QCLASS 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.

Reception of a Push Notification Update Message results in no response back to the server.

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.

## 7. 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.

## 8. IANA Considerations

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

This document defines two DNS OpCodes: SUBSCRIBE with (tentative) value 6 and UNSUBSCRIBE with (tentative) value 7.

## 9. Security Considerations

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

### 9.1. 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 [I-D.ietf-uta-tls-bcp] 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.

### 9.2. 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._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 [I-D.ietf-dane-srv]. 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.

### 9.3. 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.

### 9.4. TLS Session Resumption

TLS Session Resumption MUST be disabled on DNS Push Notification servers. It is not useful to have subscription state cached for long periods of time. It is also not desirable for subscription state to be maintained while the client is not connected.

## 10. References

### 10.1. Normative References

[I-D.ietf-dane-srv]
Finch, T., Miller, M., and P. Saint-Andre, "Using DNS-Based Authentication of Named Entities (DANE) TLSA Records with SRV Records", draft-ietf-dane-srv-11 (work in progress), February 2015.

[I-D.ietf-tls-tls13]
Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.3", draft-ietf-tls-tls13-04 (work in progress), January 2015.

[I-D.ietf-uta-tls-bcp]
Sheffer, Y., Holz, R., and P. Saint-Andre, "Recommendations for Secure Use of TLS and DTLS", draft-ietf-uta-tls-bcp-11 (work in progress), February 2015.

[RFC0768] Postel, J., "User Datagram Protocol", STD 6, RFC 768, August 1980.

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981.

[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.

[RFC1123] Braden, R., "Requirements for Internet Hosts - Application and Support", STD 3, RFC 1123, October 1989.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2136] Vixie, P., Thomson, S., Rekhter, Y., and J. Bound, "Dynamic Updates in the Domain Name System (DNS UPDATE)", RFC 2136, April 1997.

[RFC2782] Gulbrandsen, A., Vixie, P., and L. Esibov, "A DNS RR for specifying the location of services (DNS SRV)", RFC 2782, February 2000.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

[RFC5966] Bellis, R., "DNS Transport over TCP - Implementation Requirements", RFC 5966, August 2010.

[RFC6066] Eastlake, D., "Transport Layer Security (TLS) Extensions: Extension Definitions", RFC 6066, January 2011.

[RFC6195] Eastlake, D., "Domain Name System (DNS) IANA Considerations", RFC 6195, March 2011.

## 10.2. Informative 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.

[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, August 1996.

[RFC4287] Nottingham, M., Ed. and R. Sayre, Ed., "The Atom Syndication Format", RFC 4287, December 2005.

[RFC6762] Cheshire, S. and M. Krochmal, "Multicast DNS", RFC 6762, February 2013.

[RFC6763] Cheshire, S. and M. Krochmal, "DNS-Based Service Discovery", RFC 6763, February 2013.

[XEP-0060]
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: September 6, 2015

M. Stenberg

March 5, 2015

# Auto-Configuration of a Network of Hybrid Unicast/Multicast DNS-Based Service Discovery Proxy Nodes

draft-ietf-homenet-hybrid-proxy-zeroconf-00

## 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



## Copyright Notice

Table of Contents



## 1. Introduction

Section 3 ("Hybrid Proxy Operation") of [I-D.cheshire-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.cheshire-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:

- o 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).

- o 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).

- o 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.

- o 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.

- o 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. 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).

## 7. IANA Considerations

This document has no actions for IANA.

## 8. References

### 8.1. Normative references

[I-D.cheshire-dnssd-hybrid]
Cheshire, S., "Hybrid Unicast/Multicast DNS-Based Service Discovery", draft-cheshire-dnssd-hybrid-01 (work in progress), January 2014.

[RFC1035] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, November 1987.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC6762] Cheshire, S. and M. Krochmal, "Multicast DNS", RFC 6762, February 2013.

[RFC6763] Cheshire, S. and M. Krochmal, "DNS-Based Service Discovery", RFC 6763, February 2013.

### 8.2. Informative references

[I-D.ietf-homenet-hncp]
Stenberg, M., Barth, S., and P. Pfister, "Home Networking Control Protocol", draft-ietf-homenet-hncp-03 (work in progress), January 2015.

[RFC3633] Troan, O. and R. Droms, "IPv6 Prefix Options for Dynamic Host Configuration Protocol (DHCP) version 6", RFC 3633, December 2003.

[RFC3646] Droms, R., "DNS Configuration options for Dynamic Host Configuration Protocol for IPv6 (DHCPv6)", RFC 3646, December 2003.

### 8.3. URIs

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

# Appendix A. Example configuration

## A.1. Used topology

Let’s assume home network that looks like this:

```
           |[0]
        +-----+
        | CER |
        +-----+
     [1]/    \[2]
       /      \
    +-----+ +-----+
    | IR1 |-| IR2 |
    +-----+ +-----+
     |[3]|   |[4]|
```

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.

Author’s Address

Markus Stenberg
Helsinki 00930
Finland

Email: markus.stenberg@iki.fi

DNSSD
INTERNET-DRAFT
Intended Status: Informational
Expires: April 27, 2015

H. Rafiee

October 27, 2014

# Multicast DNS (mDNS) Threat Model and Security Consideration
<draft-rafiee-dnssd-mdns-threatmodel-01.txt>

## Abstract

This document describes threats associated with extending multicast DNS (mDNS) across layer 3.

## 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 27, 2015.

## Copyright Notice

Table of Contents

## 1. Introduction

Multicast DNS (mDNS) was proposed in [RFC6762] to allow nodes in local links to use DNS-like names for their communication without the need for global DNS servers, infrastructure and administration processes for configuration. mDNS along with service discovery (DNS-SD) [RFC6763] provides nodes with the possibility to discover other services and the names of other nodes with zero configuration, i.e., connect a node into a local link and use resources such as a printer that are available in that network.

mDNS and service discovery (SD) use DNS- like query messages. The main assumption is that these services also use DNS security protocols such as DNSSEC. However, it cannot use DNSSEC for security because DNSSEC is not zero configuration service. This is why the current implementations use no security in local links and are vulnerable to several attacks.

The purpose of this document is to introduce threat models for service discovery and allow implementers to be aware of the possible attacks in order to mitigate them with possible solutions. Since there are already old lists of known DNS threats available in [RFC3833], here we only analyze the ones that are applicable to DNS-SD. We also introduce new possible threats that could result from extending DNS-SD scope.

## 2. Terminology

Node: any host and routers in the network

Attack: an action to exploit a node and allow the attacker to gain access to that node. It can be also an action to prevent a node from providing a service or using a service on the network

Attacker: a person who uses any node in the network to attack other nodes using known or unknown threats

Threat: Anything that has a potential to harm a node in the network

Local link vulnerability: Any flaws that are the result of the assumption that a malicious node could gain access to legitimate nodes inside a local link network

Wide Area Network (WAN) vulnerability: Any flaws that are the result of the assumption that a malicious node could gain access to legitimate nodes inside any local links in an enterprise network with multiple Local Area Networks (LANs) or Virtual LANs (VLANs).

Host name: Fully qualified DNS Name (FQDN) of a node in the network

Constrained device: a small device with limited resources (battery,

memory, etc.)

Service Providers: a node that offer a service to other nodes. One example of a service provider in DNS-SD is a printer.

Service Requester: a node in the network that requests a service by the use of DNS-SD protocols. One example of service requester is a computer that discovers a printer in the network and tries to use it.

## 3. Threat Analysis

DNS-SD cannot use DNSSEC approaches for security purposes. This is because, as mentioned earlier, DNSSEC is not a zero config protocol and it is not compatible with the plug and play nature of DNS-SD. This is why DNS-SD is vulnerable to several attacks. Most threats in this section are a result of spoofing, Denial of Service (DoS), or a combination of them. Here we explain them in different example scenarios. The definition of different use case scenarios are defined in [requirement].

There are several scenarios associated with the Large Traffic Production case.

First scenario: a malicious node in any of the subnets that the gateway connects can advertise different fake services or spoof the information of the real services and replay the messages. This causes large traffic either in the local link or in other links since the gateway was also supposed to replicate the traffic to other links.

Second scenario : a malicious node spoofs the legitimate service advertisements of different nodes in the network and changes the Time To Leave (TTL) value to zero. This will result in producing large traffic since the mDNS gateway needs to ask all of the service advertisers to re-advertise their service. This is an especially effective attack in a network of constrained devices because it causes more energy consumption.

### 3.1. DoS attack on any node in the DNS-SD enabled network

#### 3.1.1. Personal Area Network (PAN)

When service provider and service requester are connected via a network cable or USB, then the only threat is virus or other malware that might infect any of these nodes. This might cause DoS.

Wireless PAN (WPAN) is where service provider and service requester are connected via Bluetooth or wireless. Since WPANs are short range and their coverage are usually limited, the attacker should be so close to any of those nodes to be able to perform any attacks. If this happens, the attacker might be able to forge the identity of the

service provider or perform DoS attack.

#### 3.1.2. Temporary Public Hotspot

A malicious node can spoof the source IP address of a legitimate victim node and question several services in the link. This will result in a large traffic return to the victim node from both gateway and also service owner.

### 3.2. Node compromising

#### 3.2.1. Home, Enterprise, Mesh networks

When ISP, home router/gateway and service provider (like a printer) support IPv6 address, then service providers usually automatically sets an IPv6 address. Since this address is global, this node is accessible over the internet. If the address of this service provider is known to the attacker, then it might be able to compromise this service provider and access to this network (because service providers usually supports weak security features).

### 3.3. Spoofing Attacks & forge the Identity

#### 3.3.1. Public Hotspot, Home, Enterprise, Mesh networks

Scenario 1: A malicious node can spoof the source IP address of a legitimate victim node advertises fake services in the network. This might result in compromising the victim nodes or having malicious access to the victim nodes’ resources.

Scenario2: A malicious node spoofs the content of Dynamic Host Configuration Protocol (DHCP) server messages and offers its own malicious information to the nodes in the network.

#### 3.3.2. Enterprise network

A virus or any malware can compromise a legitimate node in this network. Then this node can forge the identity of service providers or perform DoS attack on this network.

### 3.4. Malicious update on unicast DNS

A malicious node can spoof the content of DNS update message and add malicious records to unicast DNS. This attack is applicable on enterprise networks.

### 3.5. Cache Poisoning

Usually a list of service providers is cached in the service requester. When a malicious node has a chance to compromise this cache by advertising fake services, then the service requester might always connect to this fake service provider. This attack is applicable to temporary public hotspot, home, enterprise, Mesh and 6LowPAN networks.

### 3.6. Harming Privacy

If a malicious node is in any subnet (WLAN and WAN) of a network, it can learn about all services available in this network. The DNS-SD discloses some critical information about resources in this network which might be harmful to privacy. This attack is applicable to temporary public hotspot and enterprise networks.

### 3.7. Resource spoofing

Resource owners in the network have permission to have the same name for load balancing. A malicious node can claim to be one of the load balanced resource devices and maliciously respond to requests. This is applicable to temporary public hotspot and enterprise networks.

### 3.8. Dual stack attacks

Having both IPv4 and IPv6 in the same network and trying to aggregate service discovery traffic on both IP stacks might cause new security flaws during the conversion or aggregation of this traffic. It can be similar to what explained here as an aggregated traffic or lead to a wide range of spoofing attacks. This attack is applicable to home, enterprise and temporary public hotspots.

### 3.9. MAC address spoofing

In a wireless environment where MAC address filtering is in use to avoid any malicious node joining to the network, a malicious node can easily spoof the MAC address of a legitimate node and join the network and perform malicious activities. This attack is applicable to temporary public networks and enterprise networks.

### 3.10. Privacy Protection Mechanisms

#### 3.10.1. The Use of Random Data

Using a random name for services or devices or the use of random

numbers wherever possible, might prevent exposing the exact model or exact information regarding the DNS-SD service providers (e.g. printers, etc.) in the network to the attackers. However, this approach cannot be used for some standard information that the protocol needs to carry in order to offer service to other nodes. Otherwise, this random information was exchanged and agreed on between service providers and service requesters beforehand. This is exactly against the nature of zero conf protocols, i.e., DNS-SD

#### 3.10.2. Data Encryption

Encrypting the whole DNS-SD message is another way to hide the critical information in the network. But this approach might not fit well to the nature of this protocol. The reason is because these devices usually respond to anonymous service discovery requests. So, the attacker can also submit and request the same information. In other words, encryption in this stage is only extra efforts without having any benefit from it.

### 3.11. Authorization of a Service Requester

#### 3.11.1. The Use of an Access List

There can be an access list on each service providers with the list of IP addresses that can use these services. Then the service providers can use mechanisms to authorize the service requesters or to securely authenticate them with minimum interaction (zero configuration). This approach prevents the service providers from unauthorized use by an attacker. There are currently some mechanisms available -- SAVI-DHCP, CGA-TSIG, etc.

##### 3.11.1.1. SAVI-DHCP

SAVI-DHCP [DHCP-SAVI] approach uses a simple mechanism in switches or devices that knows information about the ports of switches to filter any malicious traffic. This mitigates attacks on DHCP server spoofing and can make sure that nobody can spoof the IP address of the service providers.

##### 3.11.1.2. CGA-TSIG

CGA-TSIG [cga-tsig] is another possible solution that can provide the node with secure authentication, data integrity and data confidentiality. It provides the node with zero or minimal configuration and prevents IP spoofing. This is useful when the node needs to update any record on an unicast DNS or there is an access list on service providers. This approach can be used to authenticate and authorize a node to use a service or a device.

#### 3.11.1.3. DNS over DTLS

### 3.11.2. The Use of Shared Secret

A shared secret (e.g. a password) can be shared among the service requesters. Then this value can be used to access the service providers and authenticated on them. However, this approach has a disadvantage when one of the nodes in this network that carries this shared secret is compromised then the attacker can also have unauthorized access to these services. Sharing and re-sharing this shared secret does not fit to the zero conf nature of DNS-SD protocol.

## 3.12. Authorization of a Service Provider

It is really important for the service requesters to ensure that the one claim to be a service provider (e.g. a printer) is really a service provider and its identity has not been forged by the attacker. The service requester needs to receive the IP address of service providers in a secure manner. There are some approaches that can be used for this purpose such as SAVI-DHCP, Router Advertisement. There are also some mechanisms that can be used in service requesters to complete this authentication and authorization processes such as CGA-TSIG, DNS over TLS

### 3.12.1. SAVI-DHCP

The DHCP server can carry this information and send it to the service requesters at the same time as the service requesters receive a new IP address from the DHCP servers.

### 3.12.2. Router advertisement

If Neighbor Discovery Protocol (NDP) [RFC4861] or Secure Neighbor Discovery (SeND) [RFC3971] are in use, then an option can be added to a router advertisement message which carries required information regarding the IP addresses of service providers. This is especially secure when SeND is in use.

## 3.13. Other Security Considerations

Since a WLAN might also cover a part of city, it is really important to make sure that there is required filtering in edge networks to avoid distribution of mDNS/DNS-SD messages beyond the enterprise networks.

### 3.14. Not Usable Security Mechanisms

There are some other security mechanisms that are not fit to the zero conf nature of DNS-SD protocol but might be useable in future.

#### 3.14.1. DNSSEC

Due to the pre-configuration required for DNSSEC on each nodes and DNS servers, it is not an ideal solution mechanism for zero config services. It might also necessary to access to internet to verify the DNSSEC keys and prevent IP spoofing (ask the trusted anchors the validity of the DNSSEC keys)

#### 3.14.2. IPsec

IPsec is another security protection mechanism. Similar to DNSSEC, it requires manual step for the configuration of the nodes. However, recently there are some new drafts to automate this process. This is, of course, might not be an ideal solution for DNS-SD. This is because as explained in section 4.1.2 encryption of the whole message might not be really helpful since the attacker can also request the same service.

## 4. Security Considerations

This document documents the security of mDNS and DNS-SD. It does not introduce any additional security considerations

## 5. IANA Considerations

There is no IANA consideration

## 6. Acknowledgements

The author would like to thank all those people who directly helped in improving this draft, especially John C. Klensin, Douglas Otis and Dan York

## 7. References

### 7.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC6762] Cheshire, S., Krochmal, M.,"Multicast DNS", RFC 6762, February 2013

[RFC6763] Cheshire, S., Krochmal, M., "DNS-Based Service Discovery", RFC 6763, February 2013

[RFC6275] Perkins, C., Johnson, D., Arkko, J., "Mobility Support in IPv6", RFC 6275, July 2011

[RFC3833] Atkins, D., Austein, R., "Threat Analysis of the Domain Name System (DNS)", RFC 3833, August 2004

[RFC3971] Arkko, J., Kempf, J., Zill, B., and Nikander, P., "SEcure Neighbor Discovery (SEND)", RFC 3971, March 2005.

[RFC4861] Narten, T., Nordmark, E., Simpson, W., Soliman, H., "Neighbor Discovery for IP version 6 (IPv6)", RFC 4861, September 2007.

## 7.2. Informative References

[requirement] Lynn, K., Cheshire, S., Blanchet, M., Migault, D., " Requirements for Scalable DNS-SD/mDNS Extensions", http://tools.ietf.org/html/draft-ietf-dnssd-requirements-04, October 2014

[DHCP-SAVI] Bi, J., Wu, J., Yao, G, Baker, F.,"SAVI Solution for DHCP", http://tools.ietf.org/html/draft-ietf-savi-dhcp-23, April 2014

[cga-tsig] Rafiee, H., Loewis, M., Meinel, C.,"Transaction SIGnature (TSIG) using CGA Algorithm in IPv6", http://tools.ietf.org/html/draft-rafiee-intarea-cga-tsig , June 2014

Authors' Addresses

Hosnieh Rafiee
HUAWEI TECHNOLOGIES Duesseldorf GmbH
Riesstrasse 25, 80992
Munich, Germany
Phone: +49 (0)162 204 74 58
Email: ietf@rozanak.com