

**Auto-Configuration of a Network of Hybrid Unicast/Multicast DNS-Based
Service Discovery Proxy Nodes
draft-stenberg-homenet-dnssd-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

This Internet-Draft is submitted in full conformance with the provisions of [BCP 78](#) and [BCP 79](#).

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

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

This Internet-Draft will expire on August 09, 2014.

Copyright Notice

Copyright (c) 2014 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1.	Introduction	2
2.	Requirements language	3
3.	Hybrid proxy - what to configure	3
3.1.	Conflict resolution within network	4
3.2.	Per-link DNS-SD forward zone names	4
3.3.	Reasonable defaults	4
3.3.1.	Network-wide unique link name (scheme 1)	5
3.3.2.	Node name (scheme 2)	5
3.3.3.	Link name (scheme 2)	5
4.	TLVs	5
4.1.	DNS Delegated Zone TLV	5
4.2.	Domain Name TLV	6
4.3.	Node Name TLV	7
5.	Desirable behavior	7
5.1.	DNS search path in DHCP requests	7
5.2.	Hybrid proxy	7
5.3.	Hybrid proxy network zeroconf daemon	8
6.	Security Considerations	8
7.	References	8
7.1.	Normative references	8
7.2.	Informative references	9
Appendix A.	Example configuration	9
A.1.	Topology	9
A.2.	OSPFv3-DNS interaction	9
A.3.	TLV state	10
A.4.	DNS zone	11
A.5.	Interaction with hosts	12
Appendix B.	Implementation	12
Appendix C.	Why not just proxy Multicast DNS?	12
C.1.	General problems	12
C.2.	Stateless proxying problems	13
C.3.	Stateful proxying problems	13
Appendix D.	Acknowledgements	14
	Author's Address	14

[1. Introduction](#)

[Section 3](#) ("Hybrid Proxy Operation") of [[I-D.cheshire-mdnsexthybrid](#)] 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-level state sharing mechanism (such as routing protocol)

Stenberg

Expires August 09, 2014

[Page 2]

and a naming scheme which does not even need to be same across the whole covered network to work (given working conflict resolution does work). 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 that 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

In this document, the key words "MAY", "MUST", "MUST NOT", "OPTIONAL", "RECOMMENDED", "SHOULD", and "SHOULD NOT", 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-mdnsext-hybrid](#)] describes just that there have to be NS records pointing to hybrid proxy responsible for each link within the covered network.

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).(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 larger network, after conflict resolution, identifying which network 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. If 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 either physical network layer type as base, possibly even just interface name on the node, if it's descriptive enough, for example, eth0.openwrt.example.com and wlan0.openwrt.example.com may be good enough.

4. TLVs

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

4.1. DNS Delegated Zone TLV

This TLV is effectively a combined NS and A/AAAA record for a zone. It MUST be supported by implementations conforming to this specification. Implementations SHOULD provide forward zone per link (or optimizing a bit, zone per link with Multicast DNS traffic). Implementations MAY provide reverse zone per prefix using this same mechanism. If multiple nodes advertise same reverse zone, it should

be assumed that they all have access to the link with that prefix. However, as noted in [Section 5.3](#), mainly only the node with highest precedence on the link should publish this TLV.

Contents:

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

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

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

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 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 its 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. References

7.1. Normative references

- [I-D.cheshire-mdnsexthybrid]
Cheshire, S., "Hybrid Unicast/Multicast DNS-Based Service Discovery", [draft-cheshire-mdnsexthybrid-01](#) (work in progress), January 2013.
- [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.

[7.2.](#) Informative references

[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.

[Appendix A.](#) Example configuration

[A.1.](#) 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::/48 prefix to be delegated in the home via [0].

[A.2.](#) OSPFv3-DNS interaction

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 practise 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:db8:1:1::/64 for LAN and 2001:db8:1:2::/64 for WLAN. IR2 winds up with 2001:db8:2:1::/64 for LAN and 2001:db8:2:2::/64 for WLAN.
3. IR1 is assumed to be reachable at 2001:db8:1:1::1 and IR2 at 2001:db8:2:1::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:db8:1:1::1, b=1,  
    zone="lan.node.example.com."}
```

```
DZ {address=2001:db8:1:1::1,  
    zone="1.0.0.0.1.0.0.0.8.b.d.0.1.0.0.2.ip6.arpa."}
```

```
DZ {address=2001:db8:1:1::1, b=1,  
    zone="wlan.node.example.com."}
```



```

DZ {address=2001:db8:1:1::1,
    zone="2.0.0.0.1.0.0.0.8.b.d.0.1.0.0.2.ip6.arpa."}

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

DZ {address=2001:db8:2:1::1, b=1,
    zone="lan.node-2.example.com."}
DZ {address=2001:db8:2:1::1,
    zone="1.0.0.0.2.0.0.0.8.b.d.0.1.0.0.2.ip6.arpa."}

DZ {address=2001:db8:2:1::1, b=1,
    zone="wlan.node-2.example.com."}
DZ {address=2001:db8:2:1::1,
    zone="2.0.0.0.2.0.0.0.8.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:db8:1:1::1
node-2 AAAA 2001:db8:2:1::1

node NS node
node-2 NS node-2

1.0.0.0.1.0.0.0.8.b.d.0.1.0.0.2.ip6.arpa. NS node.example.com.
2.0.0.0.1.0.0.0.8.b.d.0.1.0.0.2.ip6.arpa. NS node.example.com.
1.0.0.0.2.0.0.0.8.b.d.0.1.0.0.2.ip6.arpa. NS node-2.example.com.
2.0.0.0.2.0.0.0.8.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 ordering which 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 workaround for this is to make sure that in conflict resolution, propagated resources always loses. Due to conflict handling ordering logic, and the arbitrary order in which the original records may be in, this is non-trivial.

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 and Shwetha Bhandari for review comments.

Author's Address

Markus Stenberg
Helsinki 00930
Finland

Email: markus.stenberg@iki.fi

