Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: July 16, 2011

S. Cheshire
M. Krochmal
Apple Inc.

Jan 12, 2011

Multicast DNS
draft-cheshire-dnsext-multicastdns-13

Abstract

As networked devices become smaller, more portable, and more ubiquitous, the ability to operate with less configured infrastructure is increasingly important. In particular, the ability to look up DNS resource record data types (including, but not limited to, host names) in the absence of a conventional managed DNS server is useful.

Multicast DNS (mDNS) provides the ability to perform DNS-like operations on the local link in the absence of any conventional unicast DNS server. In addition, mDNS designates a portion of the DNS namespace to be free for local use, without the need to pay any annual fee, and without the need to set up delegations or otherwise configure a conventional DNS server to answer for those names.

The primary benefits of mDNS names are that (i) they require little or no administration or configuration to set them up, (ii) they work when no infrastructure is present, and (iii) they work during infrastructure failures.

Status of this Memo

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

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

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

This Internet-Draft will expire on July 16, 2011.

Copyright Notice

Table of Contents

1. Introduction

Multicast DNS and its companion technology DNS-based Service Discovery [DNS-SD] were created to provide IP networking with the ease-of-use and autoconfiguration for which AppleTalk was well known [NBP]. When reading this document, familiarity with the concepts of Zero Configuration Networking [Zeroconf] and automatic link-local addressing [RFC3927] [RFC4862] is helpful.

This document specifies no change to the structure of DNS messages, no new operation codes or response codes, and no new resource record types. This document describes how clients send DNS-like queries via IP multicast, and how a collection of hosts cooperate to collectively answer those queries in a useful manner.

2. Conventions and Terminology Used in this Document

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].

When this document uses the term "Multicast DNS", it should be taken to mean: "Clients performing DNS-like queries for DNS-like resource records by sending DNS-like UDP query and response packets over IP Multicast to UDP port 5353." The design rationale for selecting UDP port 5353 is discussed in Appendix A.

This document uses the term "host name" in the strict sense to mean a fully-qualified domain name that has an IPv4 or IPv6 address record. It does not use the term "host name" in the commonly used but incorrect sense to mean just the first DNS label of a host's fully-qualified domain name.

A DNS (or mDNS) packet contains an IP TTL in the IP header, which is effectively a hop-count limit for the packet, to guard against routing loops. Each Resource Record also contains a TTL, which is the number of seconds for which the Resource Record may be cached. This document uses the term "IP TTL" to refer to the IP header TTL (hop limit), and the term "RR TTL" or just "TTL" to refer to the Resource Record TTL (cache lifetime).

DNS-format messages contain a header, a Question Section, then Answer, Authority, and Additional Record Sections. The Answer, Authority, and Additional Record Sections all hold resource records in the same format. Where this document describes issues that apply equally to all three sections, it uses the term "Resource Record

Sections" to refer collectively to these three sections.

This document uses the terms "shared" and "unique" when referring to resource record sets [RFC1034]:

A "shared" resource record set is one where several Multicast DNS Responders may have records with the same name, rrtype, and rrclass, and several Responders may respond to a particular query.

A "unique" resource record set is one where all the records with that name, rrtype, and rrclass are conceptually under the control or ownership of a single Responder, and it is expected that at most one Responder should respond to a query for that name, rrtype, and rrclass. Before claiming ownership of a unique resource record set, a Responder MUST probe to verify that no other Responder already claims ownership of that set, as described in Section 8.1 "Probing". (For fault-tolerance and other reasons it is permitted sometimes to have more than one Responder answering for a particular "unique" resource record set, but such cooperating Responders MUST give answers containing identical rdata for these records. If they do not give answers containing identical rdata then the probing step will reject the data as being inconsistent with what is already being advertised on the network for those names.)

Strictly speaking the terms "shared" and "unique" apply to resource record sets, not to individual resource records, but it is sometimes convenient to talk of "shared resource records" and "unique resource records". When used this way, the terms should be understood to mean a record that is a member of a "shared" or "unique" resource record set, respectively.

3. Multicast DNS Names

A host that belongs to an organization or individual who has control over some portion of the DNS namespace can be assigned a globally unique name within that portion of the DNS namespace, such as, "cheshire.example.com." For those of us who have this luxury, this works very well. However, the majority of home computer users do not have easy access to any portion of the global DNS namespace within which they have the authority to create names. This leaves the majority of home computers effectively anonymous for practical purposes.

To remedy this problem, this document allows any computer user to elect to give their computers link-local Multicast DNS host names of the form: "single-dns-label.local." For example, a laptop computer may answer to the name "MyComputer.local." Any computer user is

granted the authority to name their computer this way, provided that the chosen host name is not already in use on that link. Having named their computer this way, the user has the authority to continue using that name until such time as a name conflict occurs on the link which is not resolved in the user's favor. If this happens, the computer (or its human user) MUST cease using the name, and SHOULD attempt to allocate a new unique name for use on that link. These conflicts are expected to be relatively rare for people who choose reasonably imaginative names, but it is still important to have a mechanism in place to handle them when they happen.

This document specifies that the DNS top-level domain ".local." is a special domain with special semantics, namely that any fully-qualified name ending in ".local." is link-local, and names within this domain are meaningful only on the link where they originate. This is analogous to IPv4 addresses in the 169.254/16 prefix, or IPv6 addresses in the FE80::/10 prefix, which are link-local and meaningful only on the link where they originate.

Any DNS query for a name ending with ".local." MUST be sent to the mDNS multicast address 224.0.0.251 (or its IPv6 equivalent FF02::FB). The design rationale for using a fixed multicast address instead of selecting from a range of multicast addresses using a hash function is discussed in Appendix B. Implementers MAY choose also to look up such names concurrently via other mechanisms (e.g. Unicast DNS) and coalesce the results in some fashion. Implementers choosing to do this should be aware of the potential for user confusion when a given name can produce different results depending on external network conditions (such as, but not limited to, which name lookup mechanism responds faster).

It is unimportant whether a name ending with ".local." occurred because the user explicitly typed in a fully-qualified domain name ending in ".local.", or because the user entered an unqualified domain name and the host software appended the suffix ".local." because that suffix appears in the user's search list. The ".local." suffix could appear in the search list because the user manually configured it, or because it was received via DHCP [RFC2132], or via any other mechanism for configuring the DNS search list. In this respect the ".local." suffix is treated no differently to any other search domain that might appear in the DNS search list.

DNS queries for names that do not end with ".local." MAY be sent to the mDNS multicast address, if no other conventional DNS server is available. This can allow hosts on the same link to continue communicating using each other's globally unique DNS names during network outages which disrupt communication with the greater Internet. When resolving global names via local multicast, it is even

more important to use DNSSEC [RFC4033] or other security mechanisms to ensure that the response is trustworthy. Resolving global names via local multicast is a contentious issue, and this document does not discuss it further, instead concentrating on the issue of resolving local names using DNS packets sent to a multicast address.

This document recommends a single flat namespace for dot-local host names, (i.e. the names of DNS "A" and "AAAA" records, which map names to IPv4 and IPv6 addresses), but other DNS record types (such as those used by DNS-based Service Discovery [DNS-SD]) may contain as many labels as appropriate for the desired usage, up to a maximum of 255 bytes, plus a terminating zero byte at the end. Name length issues are discussed further in Appendix C.

Enforcing uniqueness of host names is probably desirable in the common case, but this document does not mandate that. It is permissible for a collection of coordinated hosts to agree to maintain multiple DNS address records with the same name, possibly for load balancing or fault-tolerance reasons. This document does not take a position on whether that is sensible. It is important that both modes of operation are supported. The Multicast DNS protocol allows hosts to verify and maintain unique names for resource records where that behavior is desired, and it also allows hosts to maintain multiple resource records with a single shared name where that behavior is desired. This consideration applies to all resource records, not just address records (host names). In summary: It is required that the protocol have the ability to detect and handle name conflicts, but it is not required that this ability be used for every record.

4. Reverse Address Mapping

Like ".local.", the IPv4 and IPv6 reverse mapping domains are also defined to be link-local:

> Any DNS query for a name ending with "254.169.in-addr.arpa." MUST be sent to the IPv4 mDNS multicast address 224.0.0.251 or the IPv6 mDNS multicast address FF02::FB. Since names under this domain correspond to IPv4 link-local addresses, it is logical that the local link is the best place to find information pertaining to those names.
>
> Likewise, any DNS query for a name within the reverse mapping domains for IPv6 Link-Local addresses ("8.e.f.ip6.arpa.", "9.e.f.ip6.arpa.", "a.e.f.ip6.arpa.", and "b.e.f.ip6.arpa.") MUST be sent to the IPv6 mDNS link-local multicast address FF02::FB or the IPv4 mDNS multicast address 224.0.0.251.

5. Querying

There are two kinds of Multicast DNS Queries, one-shot queries of the kind made by legacy DNS resolvers, and continuous ongoing Multicast DNS Queries made by fully-compliant Multicast DNS Queriers, which support asynchronous operations including DNS-based Service Discovery [DNS-SD].

Except in the rare case of a Multicast DNS Responder that is advertising only shared resources records and no unique records, a Multicast DNS Responder MUST also implement a Multicast DNS Querier so that it can first verify the uniqueness of those records before it begins answering queries for them.

5.1. One-Shot Multicast DNS Queries

The most basic kind of Multicast DNS client may simply send standard DNS queries blindly to 224.0.0.251:5353, without necessarily even being aware of what a multicast address is. This change can typically be implemented with just a few lines of code in an existing DNS resolver library. Any time the name being queried for falls within one of the reserved mDNS domains (see Section 3 and Section 4) rather than using the configured unicast DNS server address, the query is instead sent to 224.0.0.251:5353 (or its IPv6 equivalent [FF02::FB]:5353). Typically the timeout would also be shortened to two or three seconds. It's possible to make a minimal mDNS resolver with only these simple changes. These queries are typically done using a high-numbered ephemeral UDP source port, but regardless of whether they are sent from a dynamic port or from a fixed port, these queries MUST NOT be sent using UDP source port 5353, since using UDP source port 5353 signals the presence of a fully-compliant Multicast DNS Querier, as described below.

A simple DNS resolver like this will typically just take the first response it receives. It will not listen for additional UDP responses, but in many instances this may not be a serious problem. If a user types "http://MyPrinter.local." into their web browser, and their simple DNS resolver just takes the first response it receives, and the user gets to see the status and configuration web page for their printer, then the protocol has met the user's needs in this case.

While a basic DNS resolver like this may be adequate for simple host name lookup, it may not get ideal behavior in other cases. Additional refinements to create a fully-compliant Multicast DNS Querier are described below.

5.2. Continuous Multicast DNS Querying

In One-Shot Queries the underlying assumption is that the transaction begins when the application issues a query, and ends when the first response is received. There is another type of query operation which is more asynchronous, in which having received one response is not necessarily an indication that there will be no more relevant responses, and the querying operation continues until no further responses are required. Determining when no further responses are required depends on the type of operation being performed. If the operation is looking up the IPv4 and IPv6 addresses of another host, then no further responses are required once a successful connection has been made to one of those IPv4 or IPv6 addresses. If the operation is browsing to present the user with a list of DNS-SD services found on the network [DNS-SD] then no further responses are required once the user indicates this to the user-interface software, e.g. by closing the network browsing window that was displaying the list of discovered services.

Imagine some hypothetical software which allows users to discover network printers. The user wishes to discover all printers on the local network, not only the printer which is quickest to respond. When the user is actively looking for a network printer to use, they open a network browsing window which displays the list of discovered printers. It would be convenient for the user if they could rely on this list of network printers to stay up to date as network printers come and go, rather than displaying out-of-date stale information, and requiring the user explicitly to click a "refresh" button any time they want to see accurate information (which, from the moment it is displayed, is itself already beginning to become out-of-date and stale). If we are to display a continuously-updated live list like this, we need to be able to do it efficiently, without naive constant polling which would be an unreasonable burden on the network. It is not expected that all users will be browsing to discover new printers all the time, but when a user is browsing to discover service instances for an extended period, we want to be able to support that operation efficiently.

Therefore, when retransmitting mDNS queries to implement this kind of continuous monitoring, the interval between the first two queries MUST be at least one second, the intervals between successive queries MUST increase by at least a factor of two, and the querier MUST implement Known-Answer Suppression, as described below in Section 7.1. Known-Answer Suppression indicates to Responders who have already replied that their responses have been received, and they don't need to send them again in response to this repeated query. Failure to implement Known-Answer Suppression can result in unacceptable levels of network traffic. When the interval between

queries reaches or exceeds 60 minutes, a querier MAY cap the interval to a maximum of 60 minutes, and perform subsequent queries at a steady-state rate of one query per hour. To avoid accidental synchronization when for some reason multiple clients begin querying at exactly the same moment (e.g. because of some common external trigger event), a Multicast DNS Querier SHOULD also delay the first query of the series by a randomly-chosen amount in the range 20-120ms.

When a Multicast DNS Querier receives an answer, the answer contains a TTL value that indicates for how many seconds this answer is valid. After this interval has passed, the answer will no longer be valid and SHOULD be deleted from the cache. Before this time is reached, a Multicast DNS Querier which has local clients with an active interest in the state of that record (e.g. a network browsing window displaying a list of discovered services to the user) SHOULD re-issue its query to determine whether the record is still valid.

To perform this cache maintenance, a Multicast DNS Querier should plan to retransmit its query after at least 50% of the record lifetime has elapsed. This document recommends the following specific strategy:

The Querier should plan to issue a query at 80% of the record lifetime, and then if no answer is received, at 85%, 90% and 95%. If an answer is received, then the remaining TTL is reset to the value given in the answer, and this process repeats for as long as the Multicast DNS Querier has an ongoing interest in the record. If after four queries no answer is received, the record is deleted when it reaches 100% of its lifetime. A Multicast DNS Querier MUST NOT perform this cache maintenance for records for which it has no local clients with an active interest. If the expiry of a particular record from the cache would result in no net effect to any client software running on the Querier device, and no visible effect to the human user, then there is no reason for the Multicast DNS Querier to waste network bandwidth checking whether the record remains valid.

To avoid the case where multiple Multicast DNS Queriers on a network all issue their queries simultaneously, a random variation of 2% of the record TTL should be added, so that queries are scheduled to be performed at 80-82%, 85-87%, 90-92% and then 95-97% of the TTL.

An additional efficiency optimization SHOULD be performed when a Multicast DNS response is received containing a unique answer (as indicated by the cache-flush bit being set, described in Section 10.2, "Announcements to Flush Outdated Cache Entries"). In this case, there is no need for the querier to continue issuing a stream of queries with exponentially-increasing intervals, since the

receipt of a unique answer is a good indication that no other answers will be forthcoming. In this case, the Multicast DNS Querier SHOULD plan to issue its next query for this record at 80-82% of the record's TTL, as described above.

A compliant Multicast DNS Querier, which implements the rules specified in this document, MUST send its Multicast DNS Queries from UDP source port 5353 (the well-known port assigned to mDNS), and MUST listen for Multicast DNS Replies sent to UDP destination port 5353 at the mDNS multicast address (224.0.0.251 and/or its IPv6 equivalent FF02::FB).

5.3. Multiple Questions per Query

Multicast DNS allows a querier to place multiple questions in the Question Section of a single Multicast DNS query packet.

The semantics of a Multicast DNS query packet containing multiple questions is identical to a series of individual DNS query packets containing one question each. Combining multiple questions into a single packet is purely an efficiency optimization, and has no other semantic significance.

5.4. Questions Requesting Unicast Responses

Sending Multicast DNS responses via multicast has the benefit that all the other hosts on the network get to see those responses, and can keep their caches up to date, and can detect conflicting responses.

However, there are situations where all the other hosts on the network don't need to see every response. Some examples are a laptop computer waking from sleep, or the Ethernet cable being connected to a running machine, or a previously inactive interface being activated through a configuration change. At the instant of wake-up or link activation, the machine is a brand new participant on a new network. Its Multicast DNS cache for that interface is empty, and it has no knowledge of its peers on that link. It may have a significant number of questions that it wants answered right away, to discover information about its new surroundings and present that information to the user. As a new participant on the network, it has no idea whether the exact same questions may have been asked and answered just seconds ago. In this case, triggering a large sudden flood of multicast responses may impose an unreasonable burden on the network.

To avoid large floods of potentially unnecessary responses in these cases, Multicast DNS defines the top bit in the class field of a DNS question as the "unicast response" bit. When this bit is set in a

question, it indicates that the Querier is willing to accept unicast responses instead of the usual multicast responses. These questions requesting unicast responses are referred to as "QU" questions, to distinguish them from the more usual questions requesting multicast responses ("QM" questions). A Multicast DNS Querier sending its initial batch of questions immediately on wake from sleep or interface activation SHOULD set the "QU" bit in those questions.

When a question is retransmitted (as described in Section 5.2) the "QU" bit SHOULD NOT be set in subsequent retransmissions of that question. Subsequent retransmissions SHOULD be usual "QM" questions. After the first question has received its responses, the querier should have a large Known-Answer list (Section 7.1) so that subsequent queries should elicit few, if any, further responses. Reverting to multicast responses as soon as possible is important because of the benefits that multicast responses provide (see Appendix D). In addition, the "QU" bit SHOULD be set only for questions that are active and ready to be sent the moment of wake from sleep or interface activation. New questions created by local clients afterwards should be treated as normal "QM" questions and SHOULD NOT have the "QU" bit set on the first question of the series.

When receiving a question with the "unicast response" bit set, a Responder SHOULD usually respond with a unicast packet directed back to the querier. However, if the Responder has not multicast that record recently (within one quarter of its TTL), then the Responder SHOULD instead multicast the response so as to keep all the peer caches up to date, and to permit passive conflict detection. In the case of answering a probe question (Section 8.1) with the "unicast response" bit set, the Responder should always generate the requested unicast response, but may also send a multicast announcement too if the time since the last multicast announcement of that record is more than a quarter of its TTL.

Unicast replies are subject to all the same packet generation rules as multicast replies, including the cache-flush bit (Section 10.2) and (except when defending a unique name against a probe from another host) randomized delays to reduce network collisions (Section 6).

5.5. Direct Unicast Queries to port 5353

In specialized applications there may be rare situations where it makes sense for a Multicast DNS Querier to send its query via unicast to a specific machine. When a Multicast DNS Responder receives a query via direct unicast, it SHOULD respond as it would for a "QU" query, as described above in Section 5.4. Since it is possible for a unicast query to be received from a machine outside the local link, Responders SHOULD check that the source address in the query packet

matches the local subnet for that link (or, in the case of IPv6, the source address has an on-link prefix) and silently ignore the packet if not.

There may be specialized situations, outside the scope of this document, where it is intended and desirable to create a Responder that does answer queries originating outside the local link. Such a Responder would need to ensure that these non-local queries are always answered via unicast back to the Querier, since an answer sent via link-local multicast would not reach a Querier outside the local link.

6. Responding

When a Multicast DNS Responder constructs and sends a Multicast DNS response packet, the Resource Record Sections of that packet must contain only records for which that Responder is explicitly authoritative. These answers may be generated because the record answers a question received in a Multicast DNS query packet, or at certain other times that the Responder determines than an unsolicited announcement is warranted. A Multicast DNS Responder MUST NOT place records from its cache, which have been learned from other Responders on the network, in the Resource Record Sections of outgoing response packets. Only an authoritative source for a given record is allowed to issue responses containing that record.

The determination of whether a given record answers a given question is done using the standard DNS rules: The record name must match the question name, the record rrtype must match the question qtype unless the qtype is "ANY" (255) or the rrtype is "CNAME" (5), and the record rrclass must match the question qclass unless the qclass is "ANY" (255). As with unicast DNS, generally only DNS class 1 ("Internet") is used, but should client software use classes other than 1 the matching rules described above MUST be used.

A Multicast DNS Responder MUST only respond when it has a positive non-null response to send, or it authoritatively knows that a particular record does not exist. For unique records, where the host has already established sole ownership of the name, it MUST return negative answers to queries for records that it knows not to exist. For example, a host with no IPv6 address, that has claimed sole ownership of the name "host.local." for all rrtypes, MUST respond to AAAA queries for "host.local." by sending a negative answer indicating that no AAAA records exist for that name. See Section 6.1 "Negative Responses". For shared records, which are owned by no single host, the nonexistence of a given record is ascertained by the failure of any machine to respond to the Multicast DNS query, not by

any explicit negative response. NXDOMAIN and other error responses MUST NOT be sent.

Multicast DNS Responses MUST NOT contain any questions in the Question Section. Any questions in the Question Section of a received Multicast DNS Response MUST be silently ignored. Multicast DNS Queriers receiving Multicast DNS Responses do not care what question elicited the response; they care only that the information in the response is true and accurate.

A Multicast DNS Responder on Ethernet [IEEE.802.3] and similar shared multiple access networks SHOULD have the capability of delaying its responses by up to 500ms, as described below.

If a large number of Multicast DNS Responders were all to respond immediately to a particular query, a collision would be virtually guaranteed. By imposing a small random delay, the number of collisions is dramatically reduced. On a full-sized Ethernet using the maximum cable lengths allowed and the maximum number of repeaters allowed, an Ethernet frame is vulnerable to collisions during the transmission of its first 256 bits. On 10Mb/s Ethernet, this equates to a vulnerable time window of 25.6us. On higher-speed variants of Ethernet, the vulnerable time window is shorter.

In the case where a Multicast DNS Responder has good reason to believe that it will be the only Responder on the link that will send a response (i.e. because it is able to answer every question in the query packet, and for all of those answer records it has previously verified that the name, rrtype and rrclass are unique on the link) it SHOULD NOT impose any random delay before responding, and SHOULD normally generate its response within at most 10ms. In particular, this applies to responding to probe queries with the "unicast response" bit set. Since receiving a probe query gives a clear indication that some other Responder is planning to start using this name in the very near future, answering such probe queries to defend a unique record is a high priority and needs to be done without delay. A probe query can be distinguished from a normal query by the fact that a probe query contains a proposed record in the Authority Section which answers the question in the Question Section (for more details, see Section 8.2, "Simultaneous Probe Tie-Breaking").

Responding without delay is appropriate for records like the address record for a particular host name, when the host name has been previously verified unique. Responding without delay is *not* appropriate for things like looking up PTR records used for DNS-based Service Discovery [DNS-SD], where a large number of responses may be anticipated.

In any case where there may be multiple responses, such as queries where the answer is a member of a shared resource record set, each Responder SHOULD delay its response by a random amount of time selected with uniform random distribution in the range 20-120ms. The reason for requiring that the delay be at least 20ms is to accommodate the situation where two or more query packets are sent back-to-back, because in that case we want a Responder with answers to more than one of those queries to have the opportunity to aggregate all of its answers into a single response packet.

In the case where the query has the TC (truncated) bit set, indicating that subsequent Known-Answer packets will follow, Responders SHOULD delay their responses by a random amount of time selected with uniform random distribution in the range 400-500ms, to allow enough time for all the Known-Answer packets to arrive, as described in Section 7.2 "Multi-Packet Known-Answer Suppression".

The source UDP port in all Multicast DNS Responses MUST be 5353 (the well-known port assigned to mDNS). Multicast DNS implementations MUST silently ignore any Multicast DNS Responses they receive where the source UDP port is not 5353.

The destination UDP port in all Multicast DNS Responses MUST be 5353 and the destination address must be the multicast address 224.0.0.251 or its IPv6 equivalent FF02::FB, except when a unicast response has been explicitly requested:

* via the "unicast response" bit,
* by virtue of being a Legacy Query (Section 6.7), or
* by virtue of being a direct unicast query.

The benefits of sending Responses via multicast are discussed in Appendix D.

To protect the network against excessive packet flooding due to software bugs or malicious attack, a Multicast DNS Responder MUST NOT (except in the one special case of answering probe queries) multicast a record on a given interface until at least one second has elapsed since the last time that record was multicast on that particular interface. A legitimate Querier on the network should have seen the previous transmission and cached it. A Querier that did not receive and cache the previous transmission will retry its request and receive a subsequent response. In the special case of answering probe queries, because of the limited time before the probing host will make its decision about whether or not to use the name, a Multicast DNS Responder MUST respond quickly. In this special case only, when responding via multicast to a probe, a Multicast DNS Responder is only required to delay its transmission as necessary to ensure an

interval of at least 250ms since the last time the record was multicast on that interface.

6.1. Negative Responses

In the early design of Multicast DNS it was assumed that explicit negative responses would never be needed. Hosts can assert the existence of the set of records which that host claims to exist, and the union of all such sets on a link is the set of Multicast DNS records that exist on that link. Asserting the non-existence of every record in the complement of that set -- i.e. all possible Multicast DNS records that could exist on this link but do not at this moment -- was felt to be impractical and unnecessary. The non-existence of a record would be ascertained by a Querier querying for it and failing to receive a response from any of the hosts currently attached to the link.

However, operational experience showed that explicit negative responses can sometimes be valuable. One such example is when a Querier is querying for a AAAA record, and the host name in question has no associated IPv6 addresses. In this case the responding host knows it currently has exclusive ownership of that name, and it knows that it currently does not have any IPv6 addresses, so an explicit negative response is preferable to the Querier having to retransmit its query multiple times and eventually give up with a timeout before it can conclude that a given AAAA record does not exist.

Any time a Responder receives a query for a name for which it has verified exclusive ownership, for a type for which that name has no records, the Responder MUST (except as allowed in (a) below) respond asserting the nonexistence of that record using a DNS NSEC record [RFC4034]. In the case of Multicast DNS the NSEC record is not being used for its usual DNSSEC [RFC4033] security properties, but simply as a way of expressing which records do or do not exist with a given name.

On receipt of a question for a particular name/rrtype/rrclass for which a Responder does have one or more unique answers, the Responder MAY also include an NSEC record in the additional section indicating the non-existence of other rrtypes for that name.

Implementers working with devices with sufficient memory and CPU resources MAY choose to implement code to handle the full generality of the DNS NSEC record [RFC4034], including bitmaps up to 65,536 bits long. To facilitate use by devices with limited memory and CPU resources, Multicast DNS Queriers are only REQUIRED to be able to parse a restricted form of the DNS NSEC record. All compliant Multicast DNS implementations MUST at least correctly generate and

parse the restricted DNS NSEC record format described below:

- The 'Next Domain Name' field contains the record's own name. When used with name compression, this means that the 'Next Domain Name' field always takes exactly two bytes in the packet.

- The Type Bit Map block number is 0.

- The Type Bit Map block length byte is a value in the range 1-32.

- The Type Bit Map data is 1-32 bytes, as indicated by length byte.

Because this restricted form of the DNS NSEC record is limited to Type Bit Map block number zero, it cannot express the existence of rrtypes above 255. Because of this, if a Multicast DNS Responder were to have records with rrtypes above 255, it MUST NOT generate these restricted-form NSEC records for those names, since to do so would imply that the name has no records with rrtypes above 255, which would be false. In such cases a Multicast DNS Responder MUST either (a) emit no NSEC record for that name, or (b) emit a full NSEC record containing the appropriate Type Bit Map block(s) with the correct bits set for all the record types that exist. In practice this is not a significant limitation, since rrtypes above 255 are not currently in widespread use.

If a Multicast DNS implementation receives an NSEC record where the 'Next Domain Name' field is not the record's own name, then the implementation SHOULD ignore the 'Next Domain Name' field and process the remainder of the NSEC record as usual. In Multicast DNS the 'Next Domain Name' field is not currently used, but it could be used in a future version of this protocol, which is why a Multicast DNS implementation MUST NOT reject or ignore an NSEC record it receives just because it finds an unexpected value in the 'Next Domain Name' field.

If a Multicast DNS implementation receives an NSEC record containing more than one Type Bit Map, or where the Type Bit Map block number is not zero, or where the block length is not in the range 1-32, then the Multicast DNS implementation MAY silently ignore the entire NSEC record. A Multicast DNS implementation MUST NOT ignore an entire packet just because that packet contains one or more NSEC record(s) that the Multicast DNS implementation cannot parse. This provision is to allow future enhancements to the protocol to be introduced in a backwards-compatible way that does not break compatibility with older Multicast DNS implementations.

To help differentiate these synthesized NSEC records (generated programmatically on-the-fly) from conventional Unicast DNS NSEC

records (which actually exist in a signed DNS zone) the synthesized Multicast DNS NSEC records MUST NOT have the 'NSEC' bit set in the Type Bit Map, whereas conventional Unicast DNS NSEC records do have the 'NSEC' bit set.

The TTL of the NSEC record indicates the intended lifetime of the negative cache entry. In general, the TTL given for an NSEC record SHOULD be the same as the TTL that the record would have had, had it existed. For example, the TTL for address records in Multicast DNS is typically 120 seconds (see Section 10) so the negative cache lifetime for an address record that does not exist should also be 120 seconds.

A Responder MUST only generate negative responses to queries for which it has legitimate ownership of the name/rrtype/rrclass in question, and can legitimately assert that no record with that name/rrtype/rrclass exists. A Responder can assert that a specified rrtype does not exist for one of its names if it knows a priori that it has exclusive ownership of that name (e.g. names of reverse address mapping PTR records, which are derived from IP addresses, which should be unique on the local link) or if it previously claimed unique ownership of that name using probe queries for rrtype "ANY". (If it were to use probe queries for a specific rrtype, then it would only own the name for that rrtype, and could not assert that other rrtypes do not exist.)

The design rationale for this mechanism for encoding Negative Responses is discussed further in Appendix E.

6.2. Responding to Address Queries

In Multicast DNS, whenever a Responder places an IPv4 or IPv6 address record (rrtype "A" or "AAAA") into a response packet, it SHOULD also place the corresponding other address type into the additional section, if there is space in the packet.

This is to provide fate sharing, so that all a device's addresses are delivered atomically in a single packet, to reduce the risk that packet loss could cause a querier to receive only the IPv4 addresses and not the IPv6 addresses, or vice versa.

In the event that a device has only IPv4 addresses but no IPv6 addresses, or vice versa, then the appropriate NSEC record SHOULD be placed into the additional section, so that queriers can know with certainty that the device has no addresses of that kind.

Some Multicast DNS Responders treat a physical interface with both IPv4 and IPv6 address as a single interface with two addresses. Other Multicast DNS Responders treat this case as logically two interfaces,

each with one address, but Responders that operate this way MUST NOT put the corresponding automatic NSEC records in replies they send (i.e. a negative IPv4 assertion in their IPv6 responses, and a negative IPv6 assertion in their IPv4 responses) because this would cause incorrect operation in Responders on the network that work the former way.

6.3. Responding to Multi-Question Queries

Multicast DNS Responders MUST correctly handle DNS query packets containing more than one question, by answering any or all of the questions to which they have answers. Unlike single-question queries where responding without delay is allowed in appropriate cases, for query packets containing more than one question all (non-defensive) answers SHOULD be randomly delayed in the range 20-120ms, or 400-500ms if the TC (truncated) bit is set. This is because when a query packet contains more than one question a Multicast DNS Responder cannot generally be certain that other Responders will not also be simultaneously generating answers to other questions in that query packet. (Answers defending a name, in response to a probe for that name, are not subject to this delay rule and are still sent immediately.)

6.4. Response Aggregation

When possible, a Responder SHOULD, for the sake of network efficiency, aggregate as many responses as possible into a single Multicast DNS response packet. For example, when a Responder has several responses it plans to send, each delayed by a different interval, then earlier responses SHOULD be delayed by up to an additional 500ms if that will permit them to be aggregated with other responses scheduled to go out a little later.

6.5. Wildcard Queries (qtype "ANY" and qclass "ANY")

When responding to queries using qtype "ANY" (255) and/or qclass "ANY" (255), a Multicast DNS Responder MUST respond with *ALL* of its records that match the query. This is subtly different to how qtype "ANY" and qclass "ANY" work in Unicast DNS.

A common misconception is that a Unicast DNS query for qtype "ANY" will elicit a response containing all matching records. This is incorrect. If there are any records that match the query, the response is required only to contain at least one of them, not necessarily all of them.

This somewhat surprising behavior is commonly seen with caching (i.e. "recursive") name servers. If a caching server receives a qtype "ANY"

query for which it has at least one valid answer, it is allowed to return only those matching answers it happens to have already in its cache, and is not required to reconsult the authoritative name server to check if there are any more records that also match the qtype "ANY" query.

For example, one might imagine that a query for qtype "ANY" for name "host.example.com" would return both the IPv4 (A) and the IPv6 (AAAA) address records for that host. In reality what happens is that it depends on the history of what queries have been previously received by intervening caching servers. If a caching server has no records for "host.example.com" then it will consult another server (usually the authoritative name server for the name in question) and in that case it will typically return all IPv4 and IPv6 address records. If however some other host has recently done a query for qtype "A" for name "host.example.com", so that the caching server already has IPv4 address records for "host.example.com" in its cache, but no IPv6 address records, then it will return only the IPv4 address records it already has cached, and no IPv6 address records.

Multicast DNS does not share this property that qtype "ANY" and qclass "ANY" queries return some undefined subset of the matching records. When responding to queries using qtype "ANY" (255) and/or qclass "ANY" (255), a Multicast DNS Responder MUST respond with *ALL* of its records that match the query.

6.6. Cooperating Multicast DNS Responders

If a Multicast DNS Responder ("A") observes some other Multicast DNS Responder ("B") send a Multicast DNS Response packet containing a resource record with the same name, rrtype and rrclass as one of A's resource records, but different rdata, then:

- If A's resource record is intended to be a shared resource record, then this is no conflict, and no action is required.

- If A's resource record is intended to be a member of a unique resource record set owned solely by that Responder, then this is a conflict and MUST be handled as described in Section 9 "Conflict Resolution".

If a Multicast DNS Responder ("A") observes some other Multicast DNS Responder ("B") send a Multicast DNS Response packet containing a resource record with the same name, rrtype and rrclass as one of A's resource records, and identical rdata, then:

- If the TTL of B's resource record given in the packet is at least half the true TTL from A's point of view, then no action is

required.

- If the TTL of B's resource record given in the packet is less than half the true TTL from A's point of view, then A MUST mark its record to be announced via multicast. Queriers receiving the record from B would use the TTL given by B, and hence may delete the record sooner than A expects. By sending its own multicast response correcting the TTL, A ensures that the record will be retained for the desired time.

These rules allow multiple Multicast DNS Responders to offer the same data on the network (perhaps for fault tolerance reasons) without conflicting with each other.

6.7. Legacy Unicast Responses

If the source UDP port in a received Multicast DNS Query is not port 5353, this indicates that the Querier originating the query is a simple resolver such as described in Section 5.1 "One-Shot Multicast DNS Queries", which does not fully implement all of Multicast DNS. In this case, the Multicast DNS Responder MUST send a UDP response directly back to the Querier, via unicast, to the query packet's source IP address and port. This unicast response MUST be a conventional unicast response as would be generated by a conventional unicast DNS server; for example, it MUST repeat the query ID and the question given in the query packet. In addition, the "cache-flush" bit described in Section 10.2 "Announcements to Flush Outdated Cache Entries" MUST NOT be set in legacy unicast responses.

The resource record TTL given in a legacy unicast response SHOULD NOT be greater than ten seconds, even if the true TTL of the Multicast DNS resource record is higher. This is because Multicast DNS Responders that fully participate in the protocol use the cache coherency mechanisms described in Section 10 "Resource Record TTL Values and Cache Coherency" to update and invalidate stale data. Were unicast responses sent to legacy resolvers to use the same high TTLs, these legacy resolvers, which do not implement these cache coherency mechanisms, could retain stale cached resource record data long after it is no longer valid.

7. Traffic Reduction

A variety of techniques are used to reduce the amount of redundant traffic on the network.

7.1. Known-Answer Suppression

When a Multicast DNS Querier sends a query to which it already knows some answers, it populates the Answer Section of the DNS query message with those answers.

Generally this applies only to Shared records, not Unique records, since if a Multicast DNS Querier already has at least one Unique record in its cache then it should not be expecting further different answers to this question, since the Unique record(s) it already has comprise the complete answer, so it has no reason to be sending the query at all. In contrast, having some Shared records in its cache does not necessarily imply that a Multicast DNS Querier will not receive further answers to this query, and it is in this case that it is beneficial to use the Known-Answer list to suppress repeated sending of redundant answers that the Querier already knows.

A Multicast DNS Responder MUST NOT answer a Multicast DNS Query if the answer it would give is already included in the Answer Section with an RR TTL at least half the correct value. If the RR TTL of the answer as given in the Answer Section is less than half of the true RR TTL as known by the Multicast DNS Responder, the Responder MUST send an answer so as to update the Querier's cache before the record becomes in danger of expiration.

Because a Multicast DNS Responder will respond if the remaining TTL given in the Known-Answer list is less than half the true TTL, it is superfluous for the Querier to include such records in the Known-Answer list. Therefore a Multicast DNS Querier SHOULD NOT include records in the Known-Answer list whose remaining TTL is less than half their original TTL. Doing so would simply consume space in the packet without achieving the goal of suppressing responses, and would therefore be a pointless waste of network bandwidth.

A Multicast DNS Querier MUST NOT cache resource records observed in the Known-Answer Section of other Multicast DNS Queries. The Answer Section of Multicast DNS Queries is not authoritative. By placing information in the Answer Section of a Multicast DNS Query the querier is stating that it *believes* the information to be true. It is not asserting that the information *is* true. Some of those records may have come from other hosts that are no longer on the network. Propagating that stale information to other Multicast DNS Queriers on the network would not be helpful.

7.2. Multi-Packet Known-Answer Suppression

Sometimes a Multicast DNS Querier will already have too many answers to fit in the Known-Answer Section of its query packets. In this case, it should issue a Multicast DNS Query containing a question and as many Known-Answer records as will fit. It MUST then set the TC (Truncated) bit in the header before sending the Query. It MUST then immediately follow the packet with another query packet containing no questions, and as many more Known-Answer records as will fit. If there are still too many records remaining to fit in the packet, it again sets the TC bit and continues until all the Known-Answer records have been sent.

A Multicast DNS Responder seeing a Multicast DNS Query with the TC bit set defers its response for a time period randomly selected in the interval 400-500ms. This gives the Multicast DNS Querier time to send additional Known-Answer packets before the Responder responds. If the Responder sees any of its answers listed in the Known-Answer lists of subsequent packets from the querying host, it MUST delete that answer from the list of answers it is planning to give (provided that no other host on the network has also issued a query for that record and is waiting to receive an answer).

If the Responder receives additional Known-Answer packets with the TC bit set, it SHOULD extend the delay as necessary to ensure a pause of 400-500ms after the last such packet before it sends its answer. This opens the potential risk that a continuous stream of Known-Answer packets could, theoretically, prevent a Responder from answering indefinitely. In practice answers are never actually delayed significantly, and should a situation arise where significant delays did happen, that would be a scenario where the network is so overloaded that it would be desirable to err on the side of caution. The consequence of delaying an answer may be that it takes a user longer than usual to discover all the services on the local network; in contrast, the consequence of incorrectly answering before all the Known-Answer packets have been received would be wasted bandwidth sending unnecessary answers on an already overloaded network. In this (rare) situation, sacrificing speed to preserve reliable network operation is the right trade-off.

7.3. Duplicate Question Suppression

If a host is planning to transmit (or retransmit) a query, and it sees another host on the network send a QM query containing the same question, and the Known-Answer Section of that query does not contain any records which this host would not also put in its own Known-Answer Section, then this host SHOULD treat its own query as having been sent. When multiple Queriers on the network are querying for the

same resource records, there is no need for them to all be repeatedly asking the same question.

7.4. Duplicate Answer Suppression

If a host is planning to send an answer, and it sees another host on the network send a response packet containing the same answer record, and the TTL in that record is not less than the TTL this host would have given, then this host SHOULD treat its own answer as having been sent, and not also send an identical answer itself. When multiple Responders on the network have the same data, there is no need for all of them to respond.

This occurs when a host has received a query, and is delaying its response for some pseudo-random interval up to 500ms, as described elsewhere in this document, and then, before the host sends its response, it sees some other host on the network send a response packet containing the same answer record.

This feature is particularly useful when Multicast DNS Proxy Servers are in use, where there could be more than one proxy on the network giving Multicast DNS answers on behalf of some other host (e.g. because that other host is currently asleep and is not itself responding to queries).

8. Probing and Announcing on Startup

Typically a Multicast DNS Responder should have, at the very least, address records for all of its active interfaces. Creating and advertising an HINFO record on each interface as well can be useful to network administrators.

Whenever a Multicast DNS Responder starts up, wakes up from sleep, receives an indication of a network interface "Link Change" event, or has any other reason to believe that its network connectivity may have changed in some relevant way, it MUST perform the two startup steps below: Probing (Section 8.1) and Announcing (Section 8.3).

8.1. Probing

The first startup step is that for all those resource records that a Multicast DNS Responder desires to be unique on the local link, it MUST send a Multicast DNS Query asking for those resource records, to see if any of them are already in use. The primary example of this is a host's address records which map its unique host name to its unique IPv4 and/or IPv6 addresses. All Probe Queries SHOULD be done using the desired resource record name and class (usually class 1, "Internet"), and query type "ANY" (255), to elicit answers for all types of records with that name. This allows a single question to be used in place of several questions, which is more efficient on the network. It also allows a host to verify exclusive ownership of a name for all rrtypes, which is desirable in most cases. It would be confusing, for example, if one host owned the "A" record for "myhost.local.", but a different host owned the "AAAA" record for that name.

The ability to place more than one question in a Multicast DNS Query is useful here, because it can allow a host to use a single packet to probe for all of its resource records instead of needing a separate packet for each. For example, a host can simultaneously probe for uniqueness of its "A" record and all its SRV records [DNS-SD] in the same query packet.

When ready to send its mDNS probe packet(s) the host should first wait for a short random delay time, uniformly distributed in the range 0-250ms. This random delay is to guard against the case where a group of devices are powered on simultaneously, or a group of devices are connected to an Ethernet hub which is then powered on, or some other external event happens that might cause a group of hosts to all send synchronized probes.

250ms after the first query the host should send a second, then 250ms after that a third. If, by 250ms after the third probe, no

conflicting Multicast DNS responses have been received, the host may move to the next step, announcing. (Note that probing is the one exception from the normal rule that there should be at least one second between repetitions of the same question, and the interval between subsequent repetitions should at least double.)

When sending probe queries, a host MUST NOT consult its cache for potential answers. Only conflicting Multicast DNS responses received "live" from the network are considered valid for the purposes of determining whether probing has succeeded or failed.

In order to allow services to announce their presence without unreasonable delay, the time window for probing is intentionally set quite short. As a result of this, from the time the first probe packet is sent, another device on the network using that name has just 750ms to respond to defend its name. On networks that are slow, or busy, or both, it is possible for round-trip latency to account for a few hundred milliseconds, and software delays in slow devices can add additional delay. For this reason, it is important that when a device receives a probe query for a name that it is currently using it SHOULD generate its response to defend that name immediately and send it as quickly as possible. The usual rules about random delays before responding, to avoid sudden bursts of simultaneous answers from different hosts, do not apply here since normally at most one host should ever respond to a given probe question. Even when a single DNS query packet contains multiple probe questions, it would be unusual for that packet to elicit a defensive response from more than one other host. Because of the mDNS multicast rate limiting rules, the probes SHOULD be sent as "QU" questions with the "unicast response" bit set, to allow a defending host to respond immediately via unicast, instead of potentially having to wait before replying via multicast.

If during probing, from the time the first probe packet is sent until 250ms after the third probe, any conflicting Multicast DNS response is received, then the probing host MUST defer to the existing host, and SHOULD choose new names for some or all of its resource records as appropriate. Apparently conflicting Multicast DNS responses received *before* the first probe packet is sent MUST be silently ignored (see discussion of stale probe packets in Section 8.2 "Simultaneous Probe Tie-Breaking" below). In the case of a host probing using query type "ANY" as recommended above, any answer containing a record with that name, of any type, MUST be considered a conflicting response and handled accordingly.

If fifteen conflicts occur within any ten-second period, then the host MUST wait at least five seconds before each successive additional probe attempt. This is to help ensure that in the event of

software bugs or other unanticipated problems, errant hosts do not flood the network with a continuous stream of multicast traffic. For very simple devices, a valid way to comply with this requirement is to always wait five seconds after any failed probe attempt before trying again.

If a Responder knows by other means that its unique resource record set name, rrtype and rrclass cannot already be in use by any other Responder on the network, then it SHOULD skip the probing step for that resource record set. For example, when creating the reverse address mapping PTR records, the host can reasonably assume that no other host will be trying to create those same PTR records, since that would imply that the two hosts were trying to use the same IP address, and if that were the case, the two hosts would be suffering communication problems beyond the scope of what Multicast DNS is designed to solve. Similarly, if a Responder is acting as a proxy, taking over from another Multicast DNS Responder that has already verified the uniqueness of the record, then the proxy SHOULD NOT repeat the probing step for those records.

8.2. Simultaneous Probe Tie-Breaking

The astute reader will observe that there is a race condition inherent in the previous description. If two hosts are probing for the same name simultaneously, neither will receive any response to the probe, and the hosts could incorrectly conclude that they may both proceed to use the name. To break this symmetry, each host populates the Query packets's Authority Section with the record or records with the rdata that it would be proposing to use, should its probing be successful. The Authority Section is being used here in a way analogous to the way it is used as the "Update Section" in a DNS Update packet [RFC2136].

When a host is probing for a group of related records with the same name (e.g. the SRV and TXT record describing a DNS-SD service), only a single question need be placed in the Question Section, since query type "ANY" (255) is used, which will elicit answers for all records with that name. However, for tie-breaking to work correctly in all cases, the Authority Section must contain *all* the records and proposed rdata being probed for uniqueness.

When a host that is probing for a record sees another host issue a query for the same record, it consults the Authority Section of that query. If it finds any resource record(s) there which answers the query, then it compares the data of that (those) resource record(s) with its own tentative data. We consider first the simple case of a host probing for a single record, receiving a simultaneous probe from another host also probing for a single record. The two records are

compared and the lexicographically later data wins. This means that if the host finds that its own data is lexicographically later, it simply ignores the other host's probe. If the host finds that its own data is lexicographically earlier, then it defers to the winning host by waiting one second, and then begins probing for this record again. The logic for waiting one second and then trying again is to guard against stale probe packets on the network (possibly even stale probe packets sent moments ago by this host itself, before some configuration change, which may be echoed back after a short delay by some Ethernet switches and some 802.11 base stations). If the winning simultaneous probe was from a real other host on the network, then after one second it will have completed its probing, and will answer subsequent probes. If the apparently winning simultaneous probe was in fact just an old stale packet on the network (maybe from the host itself), then when it retries its probing in one second, its probes will go unanswered, and it will successfully claim the name.

The determination of "lexicographically later" is performed by first comparing the record class (excluding the cache-flush bit described in Section 10.2), then the record type, then raw comparison of the binary content of the rdata without regard for meaning or structure. If the record classes differ, then the numerically greater class is considered "lexicographically later". Otherwise, if the record types differ, then the numerically greater type is considered "lexicographically later". If the rrtype and rrclass both match then the rdata is compared.

In the case of resource records containing rdata that is subject to name compression [RFC1035], the names MUST be uncompressed before comparison. (The details of how a particular name is compressed is an artifact of how and where the record is written into the DNS message; it is not an intrinsic property of the resource record itself.)

The bytes of the raw uncompressed rdata are compared in turn, interpreting the bytes as eight-bit UNSIGNED values, until a byte is found whose value is greater than that of its counterpart (in which case the rdata whose byte has the greater value is deemed lexicographically later) or one of the resource records runs out of rdata (in which case the resource record which still has remaining data first is deemed lexicographically later). The following is an example of a conflict:

```
MyPrinter.local. A 169.254.99.200
MyPrinter.local. A 169.254.200.50
```

In this case 169.254.200.50 is lexicographically later (the third byte, with value 200, is greater than its counterpart with value 99), so it is deemed the winner.

Note that it is vital that the bytes are interpreted as UNSIGNED values in the range 0-255, or the wrong outcome may result. In the example above, if the byte with value 200 had been incorrectly interpreted as a signed eight-bit value then it would be interpreted as value -56, and the wrong address record would be deemed the winner.

8.2.1. Simultaneous Probe Tie-Breaking for Multiple Records

When a host is probing for a set of records with the same name, or a packet is received containing multiple tie-breaker records answering a given probe question in the Question Section, the host's records and the tie-breaker records from the packet are each sorted into order, and then compared pairwise, using the same comparison technique described above, until a difference is found.

The records are sorted using the same lexicographical order as described above, that is: if the record classes differ, the record with the lower class number comes first. If the classes are the same but the rrtypes differ, the record with the lower rrtype number comes first. If the class and rrtype match, then the rdata is compared bytewise until a difference is found. For example, in the common case of advertising DNS-SD services with a TXT record and an SRV record, the TXT record comes first (the rrtype value for TXT is 16) and the SRV record comes second (the rrtype value for SRV is 33).

When comparing the records, if the first records match perfectly, then the second records are compared, and so on. If either list of records runs out of records before any difference is found, then the list with records remaining is deemed to have won the tie-break. If both lists run out of records at the same time without any difference being found, then this indicates that two devices are advertising identical sets of records, as is sometimes done for fault tolerance, and there is in fact no conflict.

8.3. Announcing

The second startup step is that the Multicast DNS Responder MUST send an unsolicited Multicast DNS Response containing, in the Answer Section, all of its newly registered resource records (both shared records, and unique records that have completed the probing step). If there are too many resource records to fit in a single packet, multiple packets should be used.

In the case of shared records (e.g. the PTR records used by DNS-based Service Discovery [DNS-SD]), the records are simply placed as-is into the Answer Section of the DNS Response.

In the case of records that have been verified to be unique in the previous step, they are placed into the Answer Section of the DNS Response with the most significant bit of the rrclass set to one. The most significant bit of the rrclass for a record in the Answer Section of a response packet is the mDNS "cache-flush" bit and is discussed in more detail below in Section 10.2 "Announcements to Flush Outdated Cache Entries".

The Multicast DNS Responder MUST send at least two unsolicited responses, one second apart. To provide increased robustness against packet loss a Responder MAY send up to eight unsolicited Responses, provided that the interval between unsolicited responses increases by at least a factor of two with every response sent.

A Multicast DNS Responder MUST NOT send announcements in the absence of information that its network connectivity may have changed in some relevant way. In particular, a Multicast DNS Responder MUST NOT send regular periodic announcements as a matter of course.

Whenever a Multicast DNS Responder receives any Multicast DNS response (solicited or otherwise) containing a conflicting resource record, the conflict MUST be resolved as described in Section 9 "Conflict Resolution".

8.4. Updating

At any time, if the rdata of any of a host's Multicast DNS records changes, the host MUST repeat the Announcing step described above to update neighboring caches. For example, if any of a host's IP addresses change, it MUST re-announce those address records. The host does not need to repeat the Probing step because it has already established unique ownership of that name.

In the case of shared records, a host MUST send a "goodbye" announcement with RR TTL zero (see Section 10.1 "Goodbye Packets") for the old rdata, to cause it to be deleted from peer caches, before announcing the new rdata. In the case of unique records, a host SHOULD omit the "goodbye" announcement, since the cache-flush bit on the newly announced records will cause old rdata to be flushed from peer caches anyway.

A host may update the contents of any of its records at any time, though a host SHOULD NOT update records more frequently than ten times per minute. Frequent rapid updates impose a burden on the network. If a host has information to disseminate which changes more frequently than ten times per minute, then it may be more appropriate to design a protocol for that specific purpose.

9. Conflict Resolution

A conflict occurs when a Multicast DNS Responder has a unique record for which it is currently authoritative, and it receives a Multicast DNS response packet containing a record with the same name, rrtype and rrclass, but inconsistent rdata. What may be considered inconsistent is context sensitive, except that resource records with identical rdata are never considered inconsistent, even if they originate from different hosts. This is to permit use of proxies and other fault-tolerance mechanisms that may cause more than one Responder to be capable of issuing identical answers on the network.

A common example of a resource record type that is intended to be unique, not shared between hosts, is the address record that maps a host's name to its IP address. Should a host witness another host announce an address record with the same name but a different IP address, then that is considered inconsistent, and that address record is considered to be in conflict.

Whenever a Multicast DNS Responder receives any Multicast DNS response (solicited or otherwise) containing a conflicting resource record in any of the Resource Record Sections, the Multicast DNS Responder MUST immediately reset its conflicted unique record to probing state, and go through the startup steps described above in Section 8, "Probing and Announcing on Startup". The protocol used in the Probing phase will determine a winner and a loser, and the loser MUST cease using the name, and reconfigure.

It is very important that any host receiving a resource record that conflicts with one of its own MUST take action as described above. In the case of two hosts using the same host name, where one has been configured to require a unique host name and the other has not, the one that has not been configured to require a unique host name will not perceive any conflict, and will not take any action. By reverting to Probing state, the host that desires a unique host name will go through the necessary steps to ensure that a unique host name is obtained.

The recommended course of action after probing and failing is as follows:

1. Programmatically change the resource record name in an attempt to find a new name that is unique. This could be done by adding some further identifying information (e.g. the model name of the hardware) if it is not already present in the name, or appending the digit "2" to the name, or incrementing a number at the end of the name if one is already present.

2. Probe again, and repeat as necessary until a unique name is found.

3. Once an available unique name has been determined, by probing without receiving any conflicting response, record this newly chosen name in persistent storage so that the device will use the same name the next time it is power-cycled.

4. Display a message to the user or operator informing them of the name change. For example:

   > The name "Bob's Music" is in use by another music server on the network. Your music has been renamed to "Bob's Music (2)". If you want to change this name, use [describe appropriate menu item or preference dialog here].

   The details of how the user or operator is informed of the new name depends on context. A desktop computer with a screen might put up a dialog box. A headless server in the closet may write a message to a log file, or use whatever mechanism (email, SNMP trap, etc.) it uses to inform the administrator of error conditions. On the other hand a headless server in the closet may not inform the user at all -- if the user cares, they will notice the name has changed, and connect to the server in the usual way (e.g. via web browser) to configure a new name.

5. If after one minute of probing the Multicast DNS Responder has been unable to find any unused name, it should log an error message to inform the user or operator of this fact. This situation should never occur in normal operation. The only situations that would cause this to happen would be either a deliberate denial-of-service attack, or some kind of very obscure hardware or software bug that acts like a deliberate denial-of-service attack.

These considerations apply to address records (i.e. host names) and to all resource records where uniqueness (or maintenance of some other defined constraint) is desired.

10. Resource Record TTL Values and Cache Coherency

As a general rule, the recommended TTL value for Multicast DNS resource records with a host name as the resource record's name (e.g. A, AAAA, HINFO, etc.) or a host name contained within the resource record's rdata (e.g. SRV, reverse mapping PTR record, etc.) SHOULD be 120 seconds.

The recommended TTL value for other Multicast DNS resource records is 75 minutes.

A Querier with an active outstanding query will issue a query packet when one or more of the resource record(s) in its cache is (are) 80% of the way to expiry. If the TTL on those records is 75 minutes, this ongoing cache maintenance process yields a steady-state query rate of one query every 60 minutes.

Any distributed cache needs a cache coherency protocol. If Multicast DNS resource records follow the recommendation and have a TTL of 75 minutes, that means that stale data could persist in the system for a little over an hour. Making the default RR TTL significantly lower would reduce the lifetime of stale data, but would produce too much extra traffic on the network. Various techniques are available to minimize the impact of such stale data, outlined in the five subsections below:

10.1. Goodbye Packets

In the case where a host knows that certain resource record data is about to become invalid (for example when the host is undergoing a clean shutdown) the host SHOULD send an unsolicited mDNS response packet, giving the same resource record name, rrtype, rrclass and rdata, but an RR TTL of zero. This has the effect of updating the TTL stored in neighboring hosts' cache entries to zero, causing that cache entry to be promptly deleted.

Queriers receiving a Multicast DNS Response with a TTL of zero SHOULD NOT immediately delete the record from the cache, but instead record a TTL of 1 and then delete the record one second later. In the case of multiple Multicast DNS Responders on the network described in Section 6.6 above, if one of the Responders shuts down and incorrectly sends goodbye packets for its records, it gives the other cooperating Responders one second to send out their own response to "rescue" the records before they expire and are deleted.

10.2. Announcements to Flush Outdated Cache Entries

Whenever a host has a resource record with new data, or with what might potentially be new data (e.g. after rebooting, waking from sleep, connecting to a new network link, changing IP address, etc.), the host needs to inform peers of that new data. In cases where the host has not been continuously connected and participating on the network link, it MUST first Probe to re-verify uniqueness of its unique records, as described above in Section 8.1 "Probing".

Having completed the Probing step if necessary, the host MUST then send a series of unsolicited announcements to update cache entries in its neighbor hosts. In these unsolicited announcements, if the record is one that has been verified unique, the host sets the most significant bit of the rrclass field of the resource record. This bit, the "cache-flush" bit, tells neighboring hosts that this is not a shared record type. Instead of merging this new record additively into the cache in addition to any previous records with the same name, rrtype and rrclass, all old records with that name, type and class that were received more than one second ago are declared invalid, and marked to expire from the cache in one second.

The semantics of the cache-flush bit are as follows: Normally when a resource record appears in a Resource Record Section of the DNS Response, it means, "This is an assertion that this information is true." When a resource record appears in a Resource Record Section of the DNS Response with the "cache-flush" bit set, it means, "This is an assertion that this information is the truth and the whole truth, and anything you may have heard more than a second ago regarding records of this name/rrtype/rrclass is no longer true".

To accommodate the case where the set of records from one host constituting a single unique RRSet is too large to fit in a single packet, only cache records that are more than one second old are flushed. This allows the announcing host to generate a quick burst of packets back-to-back on the wire containing all the members of the RRSet. When receiving records with the "cache-flush" bit set, all records older than one second are marked to be deleted one second in the future. One second after the end of the little packet burst, any records not represented within that packet burst will then be expired from all peer caches.

Any time a host sends a response packet containing some members of a unique RRSet, it MUST send the entire RRSet, preferably in a single packet, or if the entire RRSet will not fit in a single packet, in a quick burst of packets sent as close together as possible. The host MUST set the cache-flush bit on all members of the unique RRSet.

Another reason for waiting one second before deleting stale records from the cache is to accommodate bridged networks. For example, a host's address record announcement on a wireless interface may be bridged onto a wired Ethernet, and cause that same host's Ethernet address records to be flushed from peer caches. The one-second delay gives the host the chance to see its own announcement arrive on the wired Ethernet, and immediately re-announce its Ethernet interface's address records so that both sets remain valid and live in peer caches.

These rules, about when to set the cache-flush bit and about sending the entire rrset, apply regardless of *why* the response packet is being generated. They apply to startup announcements as described in Section 8.3 "Announcing", and to responses generated as a result of receiving query packets.

The "cache-flush" bit is only set in records in the Resource Record Sections of Multicast DNS responses sent to UDP port 5353.

The "cache-flush" bit MUST NOT be set in any resource records in a response packet sent in legacy unicast responses to UDP ports other than 5353.

The "cache-flush" bit MUST NOT be set in any resource records in the Known-Answer list of any query packet.

The "cache-flush" bit MUST NOT ever be set in any shared resource record. To do so would cause all the other shared versions of this resource record with different rdata from different Responders to be immediately deleted from all the caches on the network.

The "cache-flush" bit does *not* apply to questions listed in the Question Section of a Multicast DNS packet. The top bit of the rrclass field in questions is used for an entirely different purpose (see Section 5.4, "Questions Requesting Unicast Responses").

Note that the "cache-flush" bit is NOT part of the resource record class. The "cache-flush" bit is the most significant bit of the second 16-bit word of a resource record in a Resource Record Section of an mDNS packet (the field conventionally referred to as the rrclass field), and the actual resource record class is the least-significant fifteen bits of this field. There is no mDNS resource record class 0x8001. The value 0x8001 in the rrclass field of a resource record in an mDNS response packet indicates a resource record with class 1, with the "cache-flush" bit set. When receiving a resource record with the "cache-flush" bit set, implementations should take care to mask off that bit before storing the resource record in memory, or otherwise ensure that it is given the correct

semantic interpretation.

The re-use of the top bit of the rrclass field only applies to conventional Resource Record types that are subject to caching, not to pseudo-RRs like OPT [RFC2671], TSIG [RFC2845], TKEY [RFC2930], SIG0 [RFC2931], etc., that pertain only to a particular transport level message and not to any actual DNS data. Since pseudo-RRs should never go into the mDNS cache, the concept of a "cache-flush" bit for these types is not applicable. In particular the rrclass field of an OPT records encodes the sender's UDP payload size, and should be interpreted as a 16-bit length value in the range 0-65535, not a one-bit flag and a 15-bit length.

10.3. Cache Flush on Topology change

If the hardware on a given host is able to indicate physical changes of connectivity, then when the hardware indicates such a change, the host should take this information into account in its mDNS cache management strategy. For example, a host may choose to immediately flush all cache records received on a particular interface when that cable is disconnected. Alternatively, a host may choose to adjust the remaining TTL on all those records to a few seconds so that if the cable is not reconnected quickly, those records will expire from the cache.

Likewise, when a host reboots, or wakes from sleep, or undergoes some other similar discontinuous state change, the cache management strategy should take that information into account.

10.4. Cache Flush on Failure Indication

Sometimes a cache record can be determined to be stale when a client attempts to use the rdata it contains, and finds that rdata to be incorrect.

For example, the rdata in an address record can be determined to be incorrect if attempts to contact that host fail, either because (for an IPv4 address on a local subnet) ARP requests for that address go unanswered, because (for an IPv6 address with an on-link prefix) ND requests for that address go unanswered, or because (for an address on a remote network) a router returns an ICMP "Host Unreachable" error.

The rdata in an SRV record can be determined to be incorrect if attempts to communicate with the indicated service at the host and port number indicated are not successful.

The rdata in a DNS-SD PTR record can be determined to be incorrect if

attempts to look up the SRV record it references are not successful.

In any such case, the software implementing the mDNS resource record cache should provide a mechanism so that clients detecting stale rdata can inform the cache.

When the cache receives this hint that it should reconfirm some record, it MUST issue two or more queries for the resource record in question. If no response is received within ten seconds, then, even though its TTL may indicate that it is not yet due to expire, that record SHOULD be promptly flushed from the cache.

The end result of this is that if a printer suffers a sudden power failure or other abrupt disconnection from the network, its name may continue to appear in DNS-SD browser lists displayed on users' screens. Eventually that entry will expire from the cache naturally, but if a user tries to access the printer before that happens, the failure to successfully contact the printer will trigger the more hasty demise of its cache entries. This is a sensible trade-off between good user-experience and good network efficiency. If we were to insist that printers should disappear from the printer list within 30 seconds of becoming unavailable, for all failure modes, the only way to achieve this would be for the client to poll the printer at least every 30 seconds, or for the printer to announce its presence at least every 30 seconds, both of which would be an unreasonable burden on most networks.

10.5. Passive Observation of Failures (POOF)

A host observes the multicast queries issued by the other hosts on the network. One of the major benefits of also sending responses using multicast is that it allows all hosts to see the responses (or lack thereof) to those queries.

If a host sees queries, for which a record in its cache would be expected to be given as an answer in a multicast response, but no such answer is seen, then the host may take this as an indication that the record may no longer be valid.

After seeing two or more of these queries, and seeing no multicast response containing the expected answer within ten seconds, then even though its TTL may indicate that it is not yet due to expire, that record SHOULD be flushed from the cache. The host SHOULD NOT perform its own queries to re-confirm that the record is truly gone. If every host on a large network were to do this, it would cause a lot of unnecessary multicast traffic. If host A sends multicast queries that remain unanswered, then there is no reason to suppose that host B or any other host is likely to be any more successful.

The previous section, "Cache Flush on Failure Indication", describes a situation where a user trying to print discovers that the printer is no longer available. By implementing the passive observation described here, when one user fails to contact the printer, all hosts on the network observe that failure and update their caches accordingly.

11. Source Address Check

All Multicast DNS responses (including responses sent via unicast) SHOULD be sent with IP TTL set to 255. This is recommended to provide backwards-compatibility with older Multicast DNS Queriers (implementing draft-cheshire-dnsext-multicastdns-04.txt, published February 2004) that check the IP TTL on reception to determine whether the packet originated on the local link. These older Queriers discard all packets with TTLs other than 255.

A host sending Multicast DNS queries to a link-local destination address (including the 224.0.0.251 and FF02::FB link-local multicast addresses) MUST only accept responses to that query that originate from the local link, and silently discard any other response packets. Without this check, it could be possible for remote rogue hosts to send spoof answer packets (perhaps unicast to the victim host) which the receiving machine could misinterpret as having originated on the local link.

The test for whether a response originated on the local link is done in two ways:

* All responses received with a destination address in the IP header which is the link-local multicast address 224.0.0.251 or FF02::FB are necessarily deemed to have originated on the local link, regardless of source IP address. This is essential to allow devices to work correctly and reliably in unusual configurations, such as multiple logical IP subnets overlayed on a single link, or in cases of severe misconfiguration, where devices are physically connected to the same link, but are currently misconfigured with completely unrelated IP addresses and subnet masks.

* For responses received with a unicast destination address in the IP header, the source IP address in the packet is checked to see if it is an address on a local subnet. An IPv4 source address is determined to be on a local subnet if, for (one of) the address(es) configured on the interface receiving the packet, (I & M) == (P & M), where I and M are the interface address and subnet mask respectively, P is the source IP address from the packet, '&' represents the bitwise logical 'and' operation, and '==' represents

a bitwise equality test. An IPv6 source address is determined to be on the local link if, for any of the on-link IPv6 prefixes on the interface receiving the packet (learned via IPv6 router advertisements or otherwise configured on the host), the first 'n' bits of the IPv6 source address match the first 'n' bits of the prefix address, where 'n' is the length of the prefix being considered.

Since queriers will ignore responses apparently originating outside the local subnet, a Responder SHOULD avoid generating responses that it can reasonably predict will be ignored. This applies particularly in the case of overlayed subnets. If a Responder receives a query addressed to the link-local multicast address 224.0.0.251, from a source address not apparently on the same subnet as the Responder (or, in the case of IPv6, from a source IPv6 address for which the Responder does not have any address with the same prefix on that interface) then even if the query indicates that a unicast response is preferred (see Section 5.4, "Questions Requesting Unicast Responses"), the Responder SHOULD elect to respond by multicast anyway, since it can reasonably predict that a unicast response with an apparently non-local source address will probably be ignored.

12. Special Characteristics of Multicast DNS Domains

Unlike conventional DNS names, names that end in ".local." have only local significance. The same is true of names within the IPv4 Link-Local reverse mapping domain "254.169.in-addr.arpa." and the IPv6 Link-Local reverse mapping domains "8.e.f.ip6.arpa.", "9.e.f.ip6.arpa.", "a.e.f.ip6.arpa.", and "b.e.f.ip6.arpa."

These names function primarily as protocol identifiers, rather than as user-visible identifiers. Even though they may occasionally be visible to end users, that is not their primary purpose. As such these names should be treated as opaque identifiers. In particular, the string "local" should not be translated or localized into different languages, much as the name "localhost" is not translated or localized into different languages.

Conventional Unicast DNS seeks to provide a single unified namespace, where a given DNS query yields the same answer no matter where on the planet it is performed or to which recursive DNS server the query is sent. In contrast, each IP link has its own private ".local.", "254.169.in-addr.arpa." and IPv6 Link-Local reverse mapping namespaces, and the answer to any query for a name within those domains depends on where that query is asked. (This characteristic is not unique to Multicast DNS. Although the original concept of DNS was a single global namespace, in recent years split views, firewalls,

intranets, and the like have increasingly meant that the answer to a given DNS query has become dependent on the location of the querier.)

The IPv4 name server address for a Multicast DNS Domain is 224.0.0.251. The IPv6 name server address for a Multicast DNS Domain is FF02::FB. These are multicast addresses; therefore they identify not a single host but a collection of hosts, working in cooperation to maintain some reasonable facsimile of a competently managed DNS zone. Conceptually a Multicast DNS Domain is a single DNS zone, however its server is implemented as a distributed process running on a cluster of loosely cooperating CPUs rather than as a single process running on a single CPU.

Multicast DNS Domains are not delegated from their parent domain via use of NS (Name Server) records, and there is also no concept of delegation of subdomains within a Multicast DNS Domain. Just because a particular host on the network may answer queries for a particular record type with the name "example.local." does not imply anything about whether that host will answer for the name "child.example.local.", or indeed for other record types with the name "example.local."

There are no NS records anywhere in Multicast DNS Domains. Instead, the Multicast DNS Domains are reserved by IANA and there is effectively an implicit delegation of all Multicast DNS Domains to the 224.0.0.251:5353 and [FF02::FB]:5353 multicast groups, by virtue of client software implementing the protocol rules specified in this document.

Multicast DNS Zones have no SOA (Start of Authority) record. A conventional DNS zone's SOA record contains information such as the email address of the zone administrator and the monotonically increasing serial number of the last zone modification. There is no single human administrator for any given Multicast DNS Zone, so there is no email address. Because the hosts managing any given Multicast DNS Zone are only loosely coordinated, there is no readily available monotonically increasing serial number to determine whether or not the zone contents have changed. A host holding part of the shared zone could crash or be disconnected from the network at any time without informing the other hosts. There is no reliable way to provide a zone serial number that would, whenever such a crash or disconnection occurred, immediately change to indicate that the contents of the shared zone had changed.

Zone transfers are not possible for any Multicast DNS Zone.

13. Enabling and Disabling Multicast DNS

The option to fail-over to Multicast DNS for names not ending in ".local." SHOULD be a user-configured option, and SHOULD be disabled by default because of the possible security issues related to unintended local resolution of apparently global names. Enabling Multicast DNS for names not ending in ".local." may be appropriate on a secure isolated network, or on some future network were machines exclusively use DNSSEC for all DNS queries, and have Multicast DNS responders capable of generating the appropriate cryptographic DNSSEC signatures, thereby guarding against spoofing.

The option to lookup unqualified (relative) names by appending ".local." (or not) is controlled by whether ".local." appears (or not) in the client's DNS search list.

No special control is needed for enabling and disabling Multicast DNS for names explicitly ending with ".local." as entered by the user. The user doesn't need a way to disable Multicast DNS for names ending with ".local.", because if the user doesn't want to use Multicast DNS, they can achieve this by simply not using those names. If a user *does* enter a name ending in ".local.", then we can safely assume the user's intention was probably that it should work. Having user configuration options that can be (intentionally or unintentionally) set so that local names don't work is just one more way of frustrating the user's ability to perform the tasks they want, perpetuating the view that, "IP networking is too complicated to configure and too hard to use."

14. Considerations for Multiple Interfaces

A host SHOULD defend its dot-local host name on all active interfaces on which it is answering Multicast DNS queries.

In the event of a name conflict on *any* interface, a host should configure a new host name, if it wishes to maintain uniqueness of its host name.

A host may choose to use the same name for all of its address records on all interfaces, or it may choose to manage its Multicast DNS host name(s) independently on each interface, potentially answering to different names on different interfaces.

When answering a Multicast DNS query, a multi-homed host with a link-local address (or addresses) SHOULD take care to ensure that any address going out in a Multicast DNS response is valid for use on the interface on which the response is going out.

Just as the same link-local IP address may validly be in use simultaneously on different links by different hosts, the same link-local host name may validly be in use simultaneously on different links, and this is not an error. A multi-homed host with connections to two different links may be able to communicate with two different hosts that are validly using the same name. While this kind of name duplication should be rare, it means that a host that wants to fully support this case needs network programming APIs that allow applications to specify on what interface to perform a link-local Multicast DNS query, and to discover on what interface a Multicast DNS response was received.

There is one other special precaution that multi-homed hosts need to take. It's common with today's laptop computers to have an Ethernet connection and an 802.11 [IEEE.802.11] wireless connection active at the same time. What the software on the laptop computer can't easily tell is whether the wireless connection is in fact bridged onto the same network segment as its Ethernet connection. If the two networks are bridged together, then packets the host sends on one interface will arrive on the other interface a few milliseconds later, and care must be taken to ensure that this bridging does not cause problems:

When the host announces its host name (i.e. its address records) on its wireless interface, those announcement records are sent with the cache-flush bit set, so when they arrive on the Ethernet segment, they will cause all the peers on the Ethernet to flush the host's Ethernet address records from their caches. The mDNS protocol has a safeguard to protect against this situation: when records are received with the cache-flush bit set, other records are not deleted from peer caches immediately, but are marked for deletion in one second. When the host sees its own wireless address records arrive on its Ethernet interface, with the cache-flush bit set, this one-second grace period gives the host time to respond and re-announce its Ethernet address records, to reinstate those records in peer caches before they are deleted.

As described, this solves one problem, but creates another, because when those Ethernet announcement records arrive back on the wireless interface, the host would again respond defensively to reinstate its wireless records, and this process would continue forever, continuously flooding the network with traffic. The mDNS protocol has a second safeguard, to solve this problem: the cache-flush bit does not apply to records received very recently, within the last second. This means that when the host sees its own Ethernet address records arrive on its wireless interface, with the cache-flush bit set, it knows there's no need to re-announce its wireless address records again because it already sent them less than a second ago, and this makes them immune from deletion from peer caches. (See Section 10.2.)

15. Considerations for Multiple Responders on the Same Machine

It is possible to have more than one Multicast DNS Responder and/or Querier implementation coexist on the same machine, but there are some known issues.

15.1. Receiving Unicast Responses

In most operating systems, incoming *multicast* packets can be delivered to *all* open sockets bound to the right port number, provided that the clients take the appropriate steps to allow this. For this reason, all Multicast DNS implementations SHOULD use the SO_REUSEPORT and/or SO_REUSEADDR options (or equivalent as appropriate for the operating system in question) so they will all be able to bind to UDP port 5353 and receive incoming multicast packets addressed to that port. However, unlike multicast packets, incoming unicast UDP packets are typically delivered only to the first socket to bind to that port. This means that "QU" responses and other packets sent via unicast will be received only by the first Multicast DNS Responder and/or Querier on a system. This limitation can be partially mitigated if Multicast DNS implementations detect when they are not the first to bind to port 5353, and in that case they do not request "QU" responses. One way to detect if there is another Multicast DNS implementation already running is to attempt binding to port 5353 without using SO_REUSEPORT and/or SO_REUSEADDR, and if that fails it indicates that some other socket is already bound to this port.

15.2. Multi-Packet Known-Answer lists

When a Multicast DNS Querier issues a query with too many Known Answers to fit into a single packet, it divides the Known-Answer list into two or more packets. Multicast DNS Responders associate the initial truncated query with its continuation packets by examining the source IP address in each packet. Since two independent Multicast DNS Queriers running on the same machine will be sending packets with the same source IP address, from an outside perspective they appear to be a single entity. If both Queriers happened to send the same multi-packet query at the same time, with different Known-Answer lists, then they could each end up suppressing answers that the other needs.

15.3. Efficiency

If different clients on a machine were each to have their own separate independent Multicast DNS implementation, they would lose certain efficiency benefits. Apart from the unnecessary code duplication, memory usage, and CPU load, the clients wouldn't get the

benefit of a shared system-wide cache, and they would not be able to aggregate separate queries into single packets to reduce network traffic.

15.4. Recommendation

Because of these issues, this document encourages implementers to design systems with a single Multicast DNS implementation that provides Multicast DNS services shared by all clients on that machine, much as most operating systems today have a single TCP implementation, which is shared between all clients on that machine. Due to engineering constraints, there may be situations where embedding a "user level" Multicast DNS implementation in the client application software is the most expedient solution, and while this will usually work in practice, implementers should be aware of the issues outlined in this section.

16. Multicast DNS Character Set

Historically, unicast DNS has been plagued by the lack of any support for non-US characters. Indeed, conventional DNS is usually limited to just letters, digits and hyphens, not even allowing spaces or other punctuation. Attempts to remedy this for unicast DNS have been badly constrained by the perceived need to accommodate old buggy legacy DNS implementations. In reality, the DNS specification itself actually imposes no limits on what characters may be used in names, and good DNS implementations handle any arbitrary eight-bit data without trouble. "Clarifications to the DNS Specification" [RFC2181] directly discusses the subject of allowable character set in Section 11 ("Name syntax"), and explicitly states that DNS names may contain arbitrary eight-bit data. However, the old rules for ARPANET host names back in the 1980s required host names to be just letters, digits, and hyphens [RFC1034], and since the predominant use of DNS is to store host address records, many have assumed that the DNS protocol itself suffers from the same limitation. It might be accurate to say that there could be hypothetical bad implementations that do not handle eight-bit data correctly, but it would not be accurate to say that the protocol doesn't allow names containing eight-bit data.

Multicast DNS is a new protocol and doesn't (yet) have old buggy legacy implementations to constrain the design choices. Accordingly, it adopts the simple obvious elegant solution: all names in Multicast DNS MUST be encoded as precomposed UTF-8 [RFC3629] "Net-Unicode" [RFC5198] text.

Some users of 16-bit Unicode have taken to stuffing a "zero-width non-breaking space" character (U+FEFF) at the start of each UTF-16

file, as a hint to identify whether the data is big-endian or little-endian, and calling it a "Byte Order Mark" (BOM). Since there is only one possible byte order for UTF-8 data, a BOM is neither necessary nor permitted. Multicast DNS names MUST NOT contain a "Byte Order Mark". Any occurrence of the Unicode character U+FEFF at the start or anywhere else in a Multicast DNS name MUST be interpreted as being an actual intended part of the name, representing (just as for any other legal unicode value) an actual literal instance of that character (in this case a zero-width non-breaking space character).

For names that are restricted to US-ASCII letters, digits and hyphens, the UTF-8 encoding is identical to the US-ASCII encoding, so this is entirely compatible with existing host names. For characters outside the US-ASCII range, UTF-8 encoding is used.

Multicast DNS implementations MUST NOT use any other encodings apart from precomposed UTF-8 (US-ASCII being considered a compatible subset of UTF-8). The reasons for selecting UTF-8 instead of Punycode [RFC3492] are discussed further in Appendix F.

The simple rules for case-insensitivity in Unicast DNS [RFC1034] [RFC1035] also apply in Multicast DNS; that is to say, in name comparisons, the lower-case letters "a" to "z" (0x61 to 0x7A) match their upper-case equivalents "A" to "Z" (0x41 to 0x5A). Hence, if a Querier issues a query for an address record with the name "myprinter.local.", then a Responder having an address record with the name "MyPrinter.local." should issue a response. No other automatic equivalences should be assumed. In particular all UTF-8 multi-byte characters (codes 0x80 and higher) are compared by simple binary comparison of the raw byte values. Accented characters are *not* defined to be automatically equivalent to their unaccented counterparts. Where automatic equivalences are desired, this may be achieved through the use of programmatically-generated CNAME records. For example, if a Responder has an address record for an accented name Y, and a Querier issues a query for a name X, where X is the same as Y with all the accents removed, then the Responder may issue a response containing two resource records: A CNAME record "X CNAME Y", asserting that the requested name X (unaccented) is an alias for the true (accented) name Y, followed by the address record for Y.

17. Multicast DNS Message Size

The 1987 DNS specification [RFC1035] restricts DNS Messages carried by UDP to no more than 512 bytes (not counting the IP or UDP headers). For UDP packets carried over the wide-area Internet in 1987, this was appropriate. For link-local multicast packets on today's networks, there is no reason to retain this restriction. Given that the packets are by definition link-local, there are no Path MTU issues to consider.

Multicast DNS Messages carried by UDP may be up to the IP MTU of the physical interface, less the space required for the IP header (20 bytes for IPv4; 40 bytes for IPv6) and the UDP header (8 bytes).

In the case of a single mDNS Resource Record which is too large to fit in a single MTU-sized multicast response packet, a Multicast DNS Responder SHOULD send the Resource Record alone, in a single IP datagram, using multiple IP fragments. Resource Records this large SHOULD be avoided, except in the very rare cases where they really are the appropriate solution to the problem at hand. Implementers should be aware that many simple devices do not re-assemble fragmented IP datagrams, so large Resource Records SHOULD NOT be used except in specialized cases where the implementer knows that all receivers implement reassembly, or where the large Resource Record contains optional data which is not essential for correct operation of the client.

A Multicast DNS packet larger than the interface MTU, which is sent using fragments, MUST NOT contain more than one Resource Record.

Even when fragmentation is used, a Multicast DNS packet, including IP and UDP headers, MUST NOT exceed 9000 bytes.

Note that 9000 bytes is also the maximum payload size of an Ethernet "Jumbo" packet [Jumbo]. However, in practice Ethernet "Jumbo" packets are not widely used, so it is advantageous to keep packets under 1500 bytes whenever possible. Even on hosts that normally handle Ethernet "Jumbo" packets and IP fragment reassembly, it is becoming more common for these hosts to implement power-saving modes where the main CPU goes to sleep and hands off packet reception tasks to a more limited processor in the network interface hardware, which may not support Ethernet "Jumbo" packets or IP fragment reassembly.

18. Multicast DNS Message Format

This section describes specific rules pertaining to the allowable values for the header fields of a Multicast DNS message, and other message format considerations.

18.1. ID (Query Identifier)

Multicast DNS implementations SHOULD listen for unsolicited responses issued by hosts booting up (or waking up from sleep or otherwise joining the network). Since these unsolicited responses may contain a useful answer to a question for which the Querier is currently awaiting an answer, Multicast DNS implementations SHOULD examine all received Multicast DNS response messages for useful answers, without regard to the contents of the ID field or the Question Section. In Multicast DNS, knowing which particular query message (if any) is responsible for eliciting a particular response message is less interesting than knowing whether the response message contains useful information.

Multicast DNS implementations MAY cache any or all Multicast DNS response messages they receive, for possible future use, provided of course that normal TTL aging is performed on these cached resource records.

In multicast query messages, the Query ID SHOULD be set to zero on transmission.

In multicast responses, including unsolicited multicast responses, the Query ID MUST be set to zero on transmission, and MUST be ignored on reception.

In legacy unicast response messages generated specifically in response to a particular (unicast or multicast) query, the Query ID MUST match the ID from the query message.

18.2. QR (Query/Response) Bit

In query messages the QR bit MUST be zero.
In response messages the QR bit MUST be one.

18.3. OPCODE

In both multicast query and multicast response messages, MUST be zero (only standard queries are currently supported over multicast).

18.4. AA (Authoritative Answer) Bit

In query messages, the Authoritative Answer bit MUST be zero on transmission, and MUST be ignored on reception.

In response messages for Multicast Domains, the Authoritative Answer bit MUST be set to one (not setting this bit would imply there's some other place where "better" information may be found) and MUST be ignored on reception.

18.5. TC (Truncated) Bit

In query messages, if the TC bit is set, it means that additional Known-Answer records may be following shortly. A Responder SHOULD record this fact, and wait for those additional Known-Answer records, before deciding whether to respond. If the TC bit is clear, it means that the querying host has no additional Known Answers.

In multicast response messages, the TC bit MUST be zero on transmission, and MUST be ignored on reception.

In legacy unicast response messages, the TC bit has the same meaning as in conventional unicast DNS: it means that the response was too large to fit in a single packet, so the Querier SHOULD re-issue its query using TCP in order to receive the larger response.

18.6. RD (Recursion Desired) Bit

In both multicast query and multicast response messages, the Recursion Desired bit SHOULD be zero on transmission, and MUST be ignored on reception.

18.7. RA (Recursion Available) Bit

In both multicast query and multicast response messages, the Recursion Available bit MUST be zero on transmission, and MUST be ignored on reception.

18.8. Z (Zero) Bit

In both query and response messages, the Zero bit MUST be zero on transmission, and MUST be ignored on reception.

18.9. AD (Authentic Data) Bit

In both multicast query and multicast response messages the Authentic Data bit [RFC2535] MUST be zero on transmission, and MUST be ignored on reception.

18.10. CD (Checking Disabled) Bit

In both multicast query and multicast response messages, the Checking Disabled bit [RFC2535] MUST be zero on transmission, and MUST be ignored on reception.

18.11. RCODE (Response Code)

In both multicast query and multicast response messages, the Response Code MUST be zero on transmission. Multicast DNS messages received with non-zero Response Codes MUST be silently ignored.

18.12. Repurposing of top bit of qclass in Question Section

In the Question Section of a Multicast DNS Query, the top bit of the qclass field is used to indicate that unicast responses are preferred for this particular question. (See Section 5.4.)

18.13. Repurposing of top bit of rrclass in Resource Record Sections

In the Resource Record Sections of a Multicast DNS Response, the top bit of the rrclass field is used to indicate that the record is a member of a unique RRSet, and the entire RRSet has been sent together (in the same packet, or in consecutive packets if there are too many records to fit in a single packet). (See Section 10.2.)

18.14. Name Compression

When generating Multicast DNS packets, implementations SHOULD use name compression wherever possible to compress the names of resource records, by replacing some or all of the resource record name with a compact two-byte reference to an appearance of that data somewhere earlier in the packet [RFC1035].

This applies not only to Multicast DNS Responses, but also to Queries. When a Query contains more than one question, successive questions in the same message often contain similar names, and consequently name compression SHOULD be used, to save bytes. In addition, Queries may also contain Known Answers in the Answer Section, or probe tie-breaking data in the Authority Section, and these names SHOULD similarly be compressed for network efficiency.

In addition to compressing the *names* of resource records, names that appear within the *rdata* of the following rrtypes SHOULD also be compressed in all Multicast DNS packets:

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

Until future IETF Standards Action specifying that names in the rdata of other types should be compressed, names that appear within the rdata of any type not listed above MUST NOT be compressed.

Implementations receiving Multicast DNS packets MUST correctly decode compressed names appearing in the Question Section, and compressed names of resource records appearing in other sections.

In addition, implementations MUST correctly decode compressed names appearing within the *rdata* of the rrtypes listed above. Where possible, implementations SHOULD also correctly decode compressed names appearing within the *rdata* of other rrtypes known to the implementers at the time of implementation, because such forward-thinking planning helps facilitate the deployment of future implementations that may have reason to compress those rrtypes. It is possible that no future IETF Standards Action will be created which mandates or permits the compression of rdata in new types, but having implementations designed such that they are capable of decompressing all known types known helps keep future options open.

One specific difference between Unicast DNS and Multicast DNS is that Unicast DNS does not allow name compression for the target host in an SRV record, because Unicast DNS implementations before the first SRV specification in 1996 [RFC2052] may not decode these compressed records properly. Since all Multicast DNS implementations were created after 1996, all Multicast DNS implementations are REQUIRED to decode compressed SRV records correctly.

In legacy unicast responses generated to answer legacy queries, name compression MUST NOT be performed on SRV records.

19. Summary of Differences Between Multicast DNS and Unicast DNS

Multicast DNS shares, as much as possible, the familiar APIs, naming syntax, resource record types, etc., of Unicast DNS. There are of course necessary differences by virtue of it using multicast, and by virtue of it operating in a community of cooperating peers, rather than a precisely defined hierarchy controlled by a strict chain of formal delegations from the root. These differences are summarized below:

Multicast DNS...
* uses multicast
* uses UDP port 5353 instead of port 53
* operates in well-defined parts of the DNS namespace
* uses UTF-8, and only UTF-8, to encode resource record names
* allows names up to 255 bytes plus a terminating zero byte
* allows name compression in rdata for SRV and other record types
* allows larger UDP packets
* allows more than one question in a query packet
* defines consistent results for qtype "ANY" and qclass "ANY" queries
* uses the Answer Section of a query to list Known Answers
* uses the TC bit in a query to indicate additional Known Answers
* uses the Authority Section of a query for probe tie-breaking
* ignores the Query ID field (except for generating legacy responses)
* doesn't require the question to be repeated in the response packet
* uses unsolicited responses to announce new records
* uses NSEC records to signal non-existence of records
* defines a "unicast response" bit in the rrclass of query questions
* defines a "cache-flush" bit in the rrclass of response answers
* uses DNS RR TTL 0 to indicate that a record has been deleted
* recommends AAAA records in the additional section when responding to rrtype "A" queries, and vice versa
* monitors queries to perform Duplicate Question Suppression
* monitors responses to perform Duplicate Answer Suppression...
* ... and Ongoing Conflict Detection
* ... and Opportunistic Caching

20. 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.

A dual-stack (v4/v6) host can participate in both ".local." zones, and should register its name(s) and perform its lookups both using IPv4 and IPv6. This enables it to reach, and be reached by, both IPv4-only and IPv6-only hosts. In effect this acts like a multi-homed host, with one connection to the logical "IPv4 Ethernet segment", and a connection to the logical "IPv6 Ethernet segment". When such a host generates NSEC records, if it is using the same host name for its IPv4 addresses and its IPv6 addresses on that network interface, its NSEC records should indicate that the host name has both 'A' and AAAA records.

21. Security Considerations

The algorithm for detecting and resolving name conflicts is, by its very nature, an algorithm that assumes cooperating participants. Its purpose is to allow a group of hosts to arrive at a mutually disjoint set of host names and other DNS resource record names, in the absence of any central authority to coordinate this or mediate disputes. In the absence of any higher authority to resolve disputes, the only alternative is that the participants must work together cooperatively to arrive at a resolution.

In an environment where the participants are mutually antagonistic and unwilling to cooperate, other mechanisms are appropriate, like manually configured DNS.

In an environment where there is a group of cooperating participants, but clients cannot be sure that there are no antagonistic hosts on the same physical link, the cooperating participants need to use IPSEC signatures and/or DNSSEC [RFC4033] signatures so that they can distinguish mDNS messages from trusted participants (which they process as usual) from mDNS messages from untrusted participants (which they silently discard).

If DNS queries for *global* DNS names are sent to the mDNS multicast

address (during network outages which disrupt communication with the greater Internet) it is *especially* important to use DNSSEC, because the user may have the impression that he or she is communicating with some authentic host, when in fact he or she is really communicating with some local host that is merely masquerading as that name. This is less critical for names ending with ".local.", because the user should be aware that those names have only local significance and no global authority is implied.

Most computer users neglect to type the trailing dot at the end of a fully-qualified domain name, making it a relative domain name (e.g. "www.example.com"). In the event of network outage, attempts to positively resolve the name as entered will fail, resulting in application of the search list, including ".local.", if present. A malicious host could masquerade as "www.example.com." by answering the resulting Multicast DNS query for "www.example.com.local." To avoid this, a host MUST NOT append the search suffix ".local.", if present, to any relative (partially qualified) host name containing two or more labels. Appending ".local." to single-label relative host names is acceptable, since the user should have no expectation that a single-label host name will resolve as-is. However, users who have both "example.com" and "local" in their search lists should be aware that if they type "www" into their web browser, it may not be immediately clear to them whether the page that appears is "www.example.com" or "www.local".

Multicast DNS uses UDP port 5353. On operating systems where only privileged processes are allowed to use ports below 1024, no such privilege is required to use port 5353.

22. IANA Considerations

IANA has allocated the IPv4 link-local multicast address 224.0.0.251 for the use described in this document [mcast4].

IANA has allocated the IPv6 multicast address set FF0X::FB for the use described in this document [mcast6]. Only address FF02::FB (Link-Local Scope) is currently in use by deployed software, but it is possible that in future implementers may experiment with Multicast DNS using larger-scoped addresses, such as FF05::FB (Site-Local Scope) [RFC4291].

The re-use of the top bit of the rrclass field in the Question and Resource Record Sections means that Multicast DNS can only carry DNS records with classes in the range 0-32767. Classes in the range 32768 to 65535 are incompatible with Multicast DNS. IANA is requested to take note of this fact, and if IANA receives a request to allocate a DNS class value above 32767, IANA should make sure the requester is aware of this implication before proceeding. This does not mean that allocations of DNS class values above 32767 should not be allowed, only that they should not be allowed until the requester has indicated that they are aware of how this allocation will interact with Multicast DNS. However, to-date only three DNS classes have been assigned by IANA (1, 3 and 4), and only one (1, "Internet") is actually in widespread use, so this issue is likely to remain a purely theoretical one.

When this document is published, IANA should designate a list of domains which are deemed to have only link-local significance, as described in Section 12 of this document ("Special Characteristics of Multicast DNS Domains") [SUDN].

Specifically, the designated link-local domains are:

local.
254.169.in-addr.arpa.
8.e.f.ip6.arpa.
9.e.f.ip6.arpa.
a.e.f.ip6.arpa.
b.e.f.ip6.arpa.

23. Domain Name Reservation Considerations

The six domains listed above, and any names falling within those domains (e.g. "MyPrinter.local.", "34.12.254.169.in-addr.arpa.", "Ink-Jet._pdl-datastream._tcp.local.") are special [SUDN] in the following ways:

1. Users may use these names as they would other DNS names, entering them anywhere that they would otherwise enter a conventional DNS name, or a dotted decimal IPv4 address, or a literal IPv6 address.

   Since there is no central authority responsible for assigning dot-local names, and all devices on the local network are equally entitled to claim any dot-local name, users SHOULD be aware of this and SHOULD exercise appropriate caution. In an untrusted or unfamiliar network environment, users SHOULD be aware that using a name like "www.local" may not actually connect them to the web site they expected, and could easily connect them to a different web page, or even a fake or spoof of their intended web site, designed to trick them into revealing confidential information. As always with networking, end-to-end cryptographic security can be a useful tool. For example, when connecting with ssh, the ssh host key verification process will inform the user if it detects that the identity of the entity they are communicating with has changed since the last time they connected to that name.

2. Application software may use these names as they would other similar DNS names, and is not required to recognize the names and treat them specially. Due to the relative ease of spoofing dot-local names, end-to-end cryptographic security remains important when communicating across a local network, just as it is when communicating across the global Internet.

3. Name resolution APIs and libraries SHOULD recognize these names as special and SHOULD NOT send queries for these names to their configured (unicast) caching DNS server(s). This is to avoid unnecessary load on the root name servers and other name servers, caused by queries for which those name servers do not have useful non-negative answers to give, and will not ever have useful non-negative answers to give.

4. Caching DNS servers SHOULD recognize these names as special and SHOULD NOT attempt to look up NS records for them, or otherwise query authoritative DNS servers in an attempt to resolve these names. Instead, caching DNS servers SHOULD generate immediate NXDOMAIN responses for all such queries they may receive (from misbehaving name resolver libraries). This is to avoid unnecessary load on the root name servers and other name servers.

5. Authoritative DNS servers SHOULD NOT by default be configurable to answer queries for these names, and, like caching DNS servers, SHOULD generate immediate NXDOMAIN responses for all such queries they may receive. DNS server software MAY provide a configuration option to override this default, for testing purposes or other specialized uses.

6. DNS server operators SHOULD NOT attempt to configure authoritative DNS servers to act as authoritative for any of these names. Configuring an authoritative DNS server to act as authoritative for any of these names may not, in many cases, yield the expected result, since name resolver libraries and caching DNS servers SHOULD NOT send queries for those names (see 3 and 4 above), so such queries SHOULD be suppressed before they even reach the authoritative DNS server in question, and consequently it will not even get an opportunity to answer them.

7. DNS Registrars MUST NOT allow any of these names to be registered in the normal way to any person or entity. These names are reserved protocol identifiers with special meaning and fall outside the set of names available for allocation by registrars. Attempting to allocate one of these names as if it were a normal DNS domain name will probably not work as desired, for reasons 3, 4 and 6 above.

24. Acknowledgments

The concepts described in this document have been explored, developed and implemented with help from Ran Atkinson, Richard Brown, Freek Dijkstra, Ralph Droms, Erik Guttman, Pasi Sarolahti, Pekka Savola, Mark Townsley, Paul Vixie, Bill Woodcock, and others. Special thanks go to Bob Bradley, Josh Graessley, Scott Herscher, Rory McGuire, Roger Pantos and Kiren Sekar for their significant contributions.

25. References

25.1. Normative References

[mcast4] "IPv4 Multicast Address Space Registry", <http://www.iana.org/assignments/multicast-addresses/>.

[mcast6] "IPv6 Multicast Address Space Registry", <http://www.iana.org/assignments/ipv6-multicast-addresses/>.

[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.

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

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, November 2003.

[RFC4034] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "Resource Records for the DNS Security Extensions", RFC 4034, March 2005.

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

[SUDN] Cheshire, S. and M. Krochmal, "Special-Use Domain Names", draft-cheshire-dnsext-special-names-01 (work in progress), January 2011.

25.2. Informative References

[B4W] "Bonjour for Windows", <http://en.wikipedia.org/wiki/Bonjour_(software)>.

[DNS-SD] Cheshire, S. and M. Krochmal, "DNS-Based Service Discovery", draft-cheshire-dnsext-dns-sd-08 (work in progress), January 2011.

[IEEE.802.3] "Information technology - Telecommunications and information exchange between systems - Local and metropolitan area networks - Specific requirements - Part 3: Carrier Sense Multiple Access with Collision Detection (CMSA/CD) Access Method and Physical Layer

Specifications", IEEE Std 802.3-2008, December 2008, <http://standards.ieee.org/getieee802/802.3.html>.

[IEEE.802.11] "Information technology - Telecommunications and information exchange between systems - Local and metropolitan area networks - Specific requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications", IEEE Std 802.11-2007, June 2007, <http://standards.ieee.org/getieee802/802.11.html>.

[Jumbo] "Ethernet Jumbo Frames", November 2009, <http://www.ethernetalliance.org/files/static_page_files/EA-Ethernet Jumbo Frames v0 1.pdf>.

[NBP] Cheshire, S. and M. Krochmal, "Requirements for a Protocol to Replace AppleTalk NBP", draft-cheshire-dnsext-nbp-10 (work in progress), January 2011.

[RFC2052] Gulbrandsen, A. and P. Vixie, "A DNS RR for specifying the location of services (DNS SRV)", RFC 2052, October 1996.

[RFC2132] Alexander, S. and R. Droms, "DHCP Options and BOOTP Vendor Extensions", RFC 2132, 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.

[RFC2181] Elz, R. and R. Bush, "Clarifications to the DNS Specification", RFC 2181, July 1997.

[RFC2535] Eastlake, D., "Domain Name System Security Extensions", RFC 2535, March 1999.

[RFC2671] Vixie, P., "Extension Mechanisms for DNS (EDNS0)", RFC 2671, August 1999.

[RFC2845] Vixie, P., Gudmundsson, O., Eastlake, D., and B. Wellington, "Secret Key Transaction Authentication for DNS (TSIG)", RFC 2845, May 2000.

[RFC2930] Eastlake, D., "Secret Key Establishment for DNS (TKEY RR)", RFC 2930, September 2000.

[RFC2931] Eastlake, D., "DNS Request and Transaction Signatures (SIG(0)s)", RFC 2931, September 2000.

[RFC3492] Costello, A., "Punycode: A Bootstring encoding of Unicode for Internationalized Domain Names in Applications (IDNA)", RFC 3492, March 2003.

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

[RFC4033] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "DNS Security Introduction and Requirements", RFC 4033, March 2005.

[RFC4291] Hinden, R. and S. Deering, "IP Version 6 Addressing Architecture", RFC 4291, February 2006.

[RFC4795] Aboba, B., Thaler, D., and L. Esibov, "Link-local Multicast Name Resolution (LLMNR)", RFC 4795, January 2007.

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

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

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

[Zeroconf] Cheshire, S. and D. Steinberg, "Zero Configuration Networking: The Definitive Guide", O'Reilly Media, Inc. , ISBN 0-596-10100-7, December 2005.

Appendix A. Design Rationale for Choice of UDP Port Number

Arguments were made for and against using Multicast on UDP port 53, the standard unicast DNS port. Some of the arguments are given below. The arguments for using a different port were greater in number and more compelling so that was the option that was ultimately selected. The UDP port "5353" was selected for its mnemonic similarity to "53".

Arguments for using UDP port 53:

* This is "just DNS", so it should be the same port.

* There is less work to be done updating old resolver libraries to do simple mDNS queries. Only the destination address need be changed. In some cases, this can be achieved without any code changes, just by adding the address 224.0.0.251 to a configuration file.

Arguments for using a different port (UDP port 5353):

* This is not "just DNS". This is a DNS-like protocol, but different.

* Changing resolver library code to use a different port number is not hard. In some cases, this can be achieved without any code changes, just by adding the address 224.0.0.251:5353 to a configuration file.

* Using the same port number makes it hard to run an mDNS Responder and a conventional unicast DNS server on the same machine. If a conventional unicast DNS server wishes to implement mDNS as well, it can still do that, by opening two sockets. Having two different port numbers allows this flexibility.

* Some VPN software hijacks all outgoing traffic to port 53 and redirects it to a special DNS server set up to serve those VPN clients while they are connected to the corporate network. It is questionable whether this is the right thing to do, but it is common, and redirecting link-local multicast DNS packets to a remote server rarely produces any useful results. It does mean, for example, that a user of such VPN software becomes unable to access their local network printer sitting on their desk right next to their computer. Using a different UDP port helps avoid this particular problem.

* On many operating systems, unprivileged software may not send or receive packets on low-numbered ports. This means that any software sending or receiving mDNS packets on port 53 would have to run as "root", which is an undesirable security risk. Using a higher-numbered UDP port avoids this restriction.

Appendix B. Design Rationale for Not Using Hashed Multicast Addresses

Some discovery protocols use a range of multicast addresses, and determine the address to be used by a hash function of the name being sought. Queries are sent via multicast to the address as indicated by the hash function, and responses are returned to the querier via unicast. Particularly in IPv6, where multicast addresses are extremely plentiful, this approach is frequently advocated. For example, IPv6 Neighbor Discovery [RFC4861] sends Neighbor Solicitation messages to the "solicited-node multicast address", which is computed as a function of the solicited IPv6 address.

There are some disadvantages to using hashed multicast addresses like this in a service discovery protocol:

* When a host has a large number of records with different names, the host may have to join a large number of multicast groups. Each time a host joins or leaves a multicast group, this results in IGMP or MLD traffic on the network announcing this fact. Joining a large number of multicast groups can place undue burden on the Ethernet hardware, which typically supports a limited number of multicast addresses efficiently. When this number is exceeded, the Ethernet hardware may have to resort to receiving all multicasts and passing them up to the host networking code for filtering in software, thereby defeating much of the point of using a multicast address range in the first place. Finally, many IPv6 stacks have a fixed limit IPV6_MAX_MEMBERSHIPS, and the code simply fails with an error if a client attempts to exceed this limit. Common values for IPV6_MAX_MEMBERSHIPS are 20 or 31.

* Multiple questions cannot be placed in one packet if they don't all hash to the same multicast address.

* Duplicate Question Suppression doesn't work if queriers are not seeing each other's queries.

* Duplicate Answer Suppression doesn't work if Responders are not seeing each other's responses.

* Opportunistic Caching doesn't work.

* Ongoing Conflict Detection doesn't work.

Appendix C. Design Rationale for Maximum Multicast DNS Name Length

Multicast DNS domain names may be up to 255 bytes long, not counting the terminating zero byte at the end.

"Domain Names - Implementation and Specification" [RFC1035] says:

> Various objects and parameters in the DNS have size limits. They are listed below. Some could be easily changed, others are more fundamental.
>
> labels 63 octets or less
>
> names 255 octets or less
>
> ...
>
> the total length of a domain name (i.e., label octets and label length octets) is restricted to 255 octets or less.

This text does not state whether this 255-byte limit includes the terminating zero at the end of every name.

Several factors lead us to conclude that the 255-byte limit does *not* include the terminating zero:

- It is common in software engineering to have size limits that are a power of two, or a multiple of a power of two, for efficiency. For example, an integer on a modern processor is typically 2, 4, or 8 bytes, not 3 or 5 bytes. The number 255 is not a power of two, nor is it to most people a particularly noteworthy number. It is noteworthy to computer scientists for only one reason -- because it is exactly one *less* than a power of two. When a size limit is exactly one less than a power of two, that suggests strongly that the one extra byte is being reserved for some specific reason -- in this case reserved perhaps to leave room for a terminating zero at the end.

- In the case of DNS label lengths, the stated limit is 63 bytes. As with the total name length, this limit is exactly one less than a power of two. This label length limit also excludes the label length byte at the start of every label. Including that extra byte, a 63-byte label takes 64 bytes of space in memory or in a DNS packet.

- It is common in software engineering for the semantic "length" of an object to be one less than the number of bytes it takes to store that object. For example, in C, strlen("foo") is 3, but

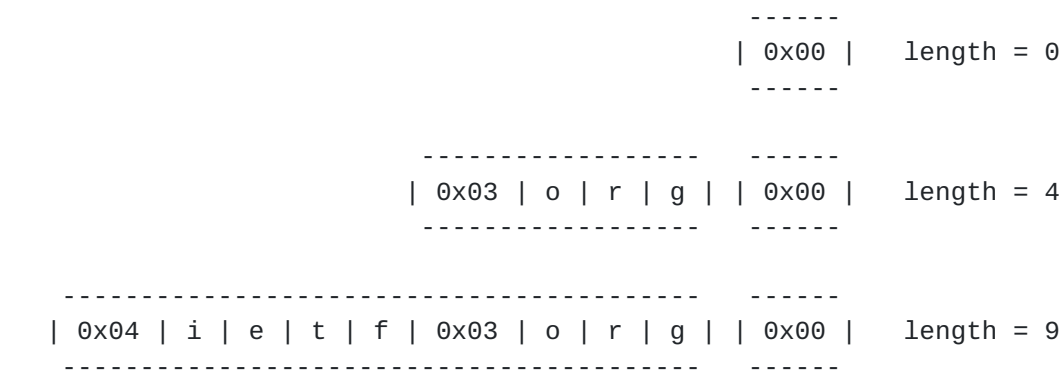
sizeof("foo") (which includes the terminating zero byte at the end) is 4.

o The text describing the total length of a domain name mentions explicitly that label length and data octets are included, but does not mention the terminating zero at the end. The zero byte at the end of a domain name is not a label length. Indeed, the value zero is chosen as the terminating marker precisely because it is not a legal length byte value -- DNS prohibits empty labels. For example, a name like "bad..name." is not a valid domain name because it contains a zero-length label in the middle, which cannot be expressed in a DNS packet, because software parsing the packet would misinterpret a zero label-length byte as being a zero "end of name" marker instead.

Finally, "Clarifications to the DNS Specification" [RFC2181] offers additional confirmation that in the context of DNS specifications the stated "length" of a domain name does not include the terminating zero byte at the end. That document refers to the root name, which is typically written as "." and is represented in a DNS packet by a single lone zero byte (i.e. zero bytes of data plus a terminating zero), as the "zero length full name":

The zero length full name is defined as representing the root of the DNS tree, and is typically written and displayed as ".".

This wording supports the interpretation that, in a DNS context, when talking about lengths of names, the terminating zero byte at the end is not counted. If the root name (".") is considered to be zero length, then to be consistent, the length (for example) of "org" has to be 4 and the length of "ietf.org" has to be 9, as shown below:



This means that the maximum length of a domain name, as represented in a Multicast DNS packet, up to but not including the final terminating zero, must not exceed 255 bytes.

However, many unicast DNS implementers have read these RFCs differently, and argue that the 255-byte limit does include the terminating zero, and that the "Clarifications to the DNS Specification" [RFC2181] statement that "." is the "zero length full name" was simply a mistake.

Hence, implementers should be aware that other unicast DNS implementations may limit the maximum domain name to 254 bytes plus a terminating zero, depending on how that implementer interpreted the DNS specifications.

Compliant Multicast DNS implementations MUST support names up to 255 bytes plus a terminating zero, i.e. 256 bytes total.

Appendix D. Benefits of Multicast Responses

Some people have argued that sending responses via multicast is inefficient on the network. In fact using multicast responses can result in a net lowering of overall multicast traffic for a variety of reasons, and provides other benefits too:

* Opportunistic Caching. One multicast response can update the caches on all machines on the network. If another machine later wants to issue the same query, it already has the answer in its cache, so it may not need to even transmit that multicast query on the network at all.

* Duplicate Query Suppression. When more than one machine has the same ongoing long-lived query running, every machine does not have to transmit its own independent query. When one machine transmits a query, all the other hosts see the answers, so they can suppress their own queries.

* Passive Observation Of Failures (POOF). When a host sees a multicast query, but does not see the corresponding multicast response, it can use this information to promptly delete stale data from its cache. To achieve the same level of user-interface quality and responsiveness without multicast responses would require lower cache lifetimes and more frequent network polling, resulting in a higher packet rate.

* Passive Conflict Detection. Just because a name has been previously verified unique does not guarantee it will continue to be so indefinitely. By allowing all Multicast DNS Responders to constantly monitor their peers' responses, conflicts arising out of network topology changes can be promptly detected and resolved. If responses were not sent via multicast, some other conflict detection mechanism would be needed, imposing its own additional burden on the network.

* Use on devices with constrained memory resources: When using delayed responses to reduce network collisions, Responders need to maintain a list recording to whom each answer should be sent. The option of multicast responses allows Responders with limited storage, which cannot store an arbitrarily long list of response addresses, to choose to fail-over to a single multicast response in place of multiple unicast responses, when appropriate.

* Overlayed Subnets. In the case of overlayed subnets, multicast responses allow a receiver to know with certainty that a response originated on the local link, even when its source address may apparently suggest otherwise.

* Robustness in the face of misconfiguration: Link-local multicast transcends virtually every conceivable network misconfiguration. Even if you have a collection of devices where every device's IP address, subnet mask, default gateway, and DNS server address are all wrong, packets sent by any of those devices addressed to a link-local multicast destination address will still be delivered to all peers on the local link. This can be extremely helpful when diagnosing and rectifying network problems, since it facilitates a direct communication channel between client and server that works without reliance on ARP, IP routing tables, etc. Being able to discover what IP address a device has (or thinks it has) is frequently a very valuable first step in diagnosing why it is unable to communicate on the local network.

Appendix E. Design Rationale for Encoding Negative Responses

Alternative methods of asserting nonexistence were considered, such as using an NXDOMAIN response, or emitting a resource record with zero-length rdata.

Using an NXDOMAIN response does not work well with Multicast DNS. A Unicast DNS NXDOMAIN response applies to the entire packet, but for efficiency Multicast DNS allows (and encourages) multiple responses in a single packet. If the error code in the header were NXDOMAIN, it would not be clear to which name(s) that error code applied.

Asserting nonexistence by emitting a resource record with zero-length rdata would mean that there would be no way to differentiate between a record that doesn't exist, and a record that does exist, with zero-length rdata. By analogy, most file systems today allow empty files, so a file that exists with zero bytes of data is not considered equivalent to a filename that does not exist.

A benefit of asserting nonexistence through NSEC records instead of through NXDOMAIN responses is that NSEC records can be added to the Additional Section of a DNS Response to offer additional information beyond what the Querier explicitly requested. For example, in a response to an SRV query, a Responder should include 'A' record(s) giving its IPv4 addresses in the Additional Section, and an NSEC record indicating which other types it does or does not have for this name. If the Responder is running on a host that does not support IPv6 (or does support IPv6 but currently has no IPv6 address on that interface) then this NSEC record in the Additional Section will indicate this absence of AAAA records. In effect, the Responder is saying, "Here's my SRV record, and here are my IPv4 addresses, and no, I don't have any IPv6 addresses, so don't waste your time asking." Without this information in the Additional Section it would take the Querier an additional round-trip to perform an additional Query to ascertain that the target host has no AAAA records. (Arguably Unicast DNS could also benefit from this ability to express nonexistence in the Additional Section, but that is outside the scope of this document.)

Appendix F. Use of UTF-8

After many years of debate, as a result of the perceived need to accommodate certain DNS implementations that apparently couldn't handle any character that's not a letter, digit or hyphen (and apparently never would be updated to remedy this limitation) the unicast DNS community settled on an extremely baroque encoding called "Punycode" [RFC3492]. Punycode is a remarkably ingenious encoding solution, but it is complicated, hard to understand, and hard to implement, using sophisticated techniques including insertion unsort coding, generalized variable-length integers, and bias adaptation. The resulting encoding is remarkably compact given the constraints, but it's still not as good as simple straightforward UTF-8, and it's hard even to predict whether a given input string will encode to a Punycode string that fits within DNS's 63-byte limit, except by simply trying the encoding and seeing whether it fits. Indeed, the encoded size depends not only on the input characters, but on the order they appear, so the same set of characters may or may not encode to a legal Punycode string that fits within DNS's 63-byte limit, depending on the order the characters appear. This is extremely hard to present in a user interface that explains to users why one name is allowed, but another name containing the exact same characters is not. Neither Punycode nor any other of the "ASCII-Compatible Encodings" [RFC5890] proposed for Unicast DNS may be used in Multicast DNS packets. Any text being represented internally in some other representation must be converted to canonical precomposed UTF-8 before being placed in any Multicast DNS packet.

Appendix G. Private DNS Namespaces

The special treatment of names ending in ".local." has been implemented in Macintosh computers since the days of Mac OS 9, and continues today in Mac OS X and iOS. There are also implementations for Microsoft Windows [B4W], Linux, and other platforms.

Some network operators setting up private internal networks ("intranets") have used unregistered top-level domains, and some may have used the ".local" top-level domain. Using ".local" as a private top-level domain conflicts with Multicast DNS and may cause problems for users. Clients can be configured to send both Multicast and Unicast DNS queries in parallel for these names, and this does allow names to be looked up both ways, but this results in additional network traffic and additional delays in name resolution, as well as potentially creating user confusion when it is not clear whether any given result was received via link-local multicast from a peer on the same link, or from the configured unicast name server. Because of this, we recommend against using ".local" as a private unicast DNS top-level domain. We do not recommend use of unregistered top-level domains at all, but should network operators decide to do this, the following top-level domains have been used on private internal networks without the problems caused by trying to re-use ".local" for this purpose:

.intranet
.internal
.private
.corp
.home
.lan

Appendix H. Deployment History

In July 1997, in an email to the net-thinkers@thumper.vmeng.com mailing list, Stuart Cheshire first proposed the idea of running AppleTalk Name Binding Protocol [NBP] over IP. As a result of this and related IETF discussions, the IETF Zeroconf Working Group was chartered September 1999. After various working group discussions and other informal IETF discussions, several Internet Drafts were written, which were loosely-related to the general themes of DNS and multicast, but did not address the service discovery aspect of NBP.

In April 2000 Stuart Cheshire registered IPv4 multicast address 224.0.0.251 with IANA [mcast4] and began writing code to test and develop the idea of performing NBP-like service discovery using Multicast DNS, which was documented in a group of three Internet Drafts:

- "draft-cheshire-dnsext-nbp-00.txt", was an overview explaining AppleTalk Name Binding Protocol, because many in the IETF community had little first-hand experience using AppleTalk, and confusion in the IETF community about what AppleTalk NBP did was causing confusion about what would be required in an IP-based replacement.

- "draft-cheshire-dnsext-nias-00.txt" ("Named Instances of Abstract Services") proposed a way to perform NBP-like service discovery using DNS-compatible names and record types.

- "draft-cheshire-dnsext-multicastdns-00.txt" proposed a way to transport those DNS-compatible queries and responses using IP multicast, for Zero Configuration environments where no conventional unicast DNS server was available.

In 2001 an update to Mac OS 9 added resolver library support for host name lookup using Multicast DNS. If the user typed a name such as "MyPrinter.local." into any piece of networking software that used the standard Mac OS 9 name lookup APIs, then those name lookup APIs would recognize the name as a dot-local name and query for it by sending simple one-shot Multicast DNS Queries to 224.0.0.251:5353. This enabled the user to, for example, enter the name "MyPrinter.local." into their web browser in order to view a printer's status and configuration web page, or enter the name "MyPrinter.local." into the printer setup utility to create a print queue for printing documents on that printer.

Multicast DNS Responder software, with full service discovery, first began shipping to end users in volume with the launch of Mac OS X 10.2 "Jaguar" in August 2002, and network printer makers (who had

historically supported AppleTalk in their network printers, and were receptive to IP-based technologies that could offer them similar ease-of-use) started adopting Multicast DNS shortly thereafter.

In September 2002 Apple released the source code for the mDNSResponder daemon as Open Source under Apple's standard Apple Public Source License (APSL).

Multicast DNS Responder software became available for Microsoft Windows users in June 2004 with the launch of Apple's "Rendezvous for Windows" (now "Bonjour for Windows"), both in executable form (a downloadable installer for end users) and as Open Source (one of the supported platforms within Apple's body of cross-platform code in the publicly-accessible mDNSResponder CVS source code repository) [B4W].

In August 2006, Apple re-licensed the cross-platform mDNSResponder source code under the Apache License, Version 2.0.

In January 2007, the IETF published "Link-Local Multicast Name Resolution", which is substantially similar to Multicast DNS, but incompatible in some small but important ways. In particular, the LLMNR design explicitly excluded support for service discovery [RFC4795], which made it an unsuitable candidate for a protocol to replace AppleTalk NBP [NBP].

In addition to desktop and laptop computers running Mac OS X and Microsoft Windows, Multicast DNS is now implemented in a wide range of hardware devices, such as Apple's "AirPort" wireless base stations, iPhone and iPad, and in home gateways from other vendors, network printers, network cameras, TiVo DVRs, etc.

The Open Source community has produced many independent implementations of Multicast DNS, some in C like Apple's mDNSResponder daemon, and others in a variety of different languages including Java, Python, Perl, and C#/Mono.

While the original focus of Multicast DNS and DNS-based Service Discovery was for Zero Configuration environments without a conventional unicast DNS server, DNS-based Service Discovery also works using unicast DNS servers, using DNS Update [RFC2136] to create service discovery records and standard DNS queries to query for them. Apple's Back to My Mac service, launched with Mac OS X 10.5 "Leopard" in October 2007, uses DNS-based Service Discovery over unicast DNS.

Authors' Addresses

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

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

Marc Krochmal
Apple Inc.
1 Infinite Loop
Cupertino, California 95014
USA

Phone: +1 408 974 4368
Email: marc@apple.com