Document: draft-cheshire-dnsext-multicastdns-06.txt
Internet-Draft
Category: Standards Track
Expires 10th February 2007

Stuart Cheshire
Marc Krochmal
Apple Computer, Inc.
10th August 2006

Multicast DNS

<draft-cheshire-dnsext-multicastdns-06.txt>

Status of this Memo

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 becoming essential.

Multicast DNS (mDNS) provides the ability to do 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.

Table of Contents

1. Introduction

When reading this document, familiarity with the concepts of Zero Configuration Networking [ZC] and automatic link-local addressing [RFC 2462] [RFC 3927] is helpful.

This document proposes no change to the structure of DNS messages, and no new operation codes, response codes, or resource record types. This document simply discusses what needs to happen if DNS clients start sending DNS queries to a multicast address, and how a collection of hosts can cooperate to collectively answer those queries in a useful manner.

There has been discussion of how much burden Multicast DNS might impose on a network. It should be remembered that whenever IPv4 hosts communicate, they broadcast ARP packets on the network on a regular basis, and this is not disastrous. The approximate amount of multicast traffic generated by hosts making conventional use of Multicast DNS is anticipated to be roughly the same order of magnitude as the amount of broadcast ARP traffic those hosts already generate.

New applications making new use of Multicast DNS capabilities for unconventional purposes may generate more traffic. If some of those new applications are "chatty", then work will be needed to help them become less chatty. When performing any analysis, it is important to make a distinction between the application behavior and the underlying protocol behavior. If a chatty application uses UDP, that doesn't mean that UDP is chatty, or that IP is chatty, or that Ethernet is chatty. What it means is that the application is chatty. The same applies to any future applications that may decide to layer increasing portions of their functionality over Multicast DNS.

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" [RFC 2119].

This document uses the term "host name" in the strict sense to mean a fully qualified domain name that has an 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.

In any place where there may be potential confusion between these two types of TTL, the term "IP TTL" is used to refer to the IP header TTL (hop limit), and the term "RR TTL" is used to refer to the Resource Record TTL (cache lifetime).

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."

This document uses the terms "shared" and "unique" when referring to resource record sets.

A "shared" resource record set is one where several Multicast DNS responders may have records with that 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 9.1 "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 or the answers will be perceived to be in conflict with each other.

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

This document proposes that the DNS top-level domain ".local." be designated 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, 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).

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 in a DHCP option, or via any other valid 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 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 in detail, instead concentrating on the issue of resolving local names using DNS packets sent to a multicast address.

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, for example, "cheshire.apple.com." For those of us who have this luxury, this works very well. However, the majority of home customers do not have easy access to any portion of the global DNS namespace within which they have the authority to create names as they wish. 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 "cheshire.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 favour. If this happens, the computer (or its human user) SHOULD cease using the name, and may choose to 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.

The point made in the previous paragraph is very important and bears repeating. It is easy for those of us in the IETF community who run our own name servers at home to forget that the majority of computer users do not run their own name server and have no easy way to create their own host names. When these users wish to transfer files between two laptop computers, they are frequently reduced to typing in dotted-decimal IP addresses because they simply have no other way for one host to refer to the other by name. This is a sorry state of affairs. What is worse, most users don't even bother trying to use dotted-decimal IP addresses. Most users still move data between machines by burning it onto CD-R, copying it onto a USB "keychain" flash drive, or similar removable media.

In a world of gigabit Ethernet and ubiquitous wireless networking it is a sad indictment of the networking community that most users still prefer sneakernet.

Allowing ad-hoc allocation of single-label names in a single flat ".local." namespace may seem to invite chaos. However, operational experience with AppleTalk NBP names [NBP], which on any given link are also effectively single-label names in a flat namespace, shows that in practice name collisions happen extremely rarely and are not a problem. Groups of computer users from disparate organizations bring Macintosh laptop computers to events such as IETF Meetings, the Mac Hack conference, the Apple World Wide Developer Conference, etc., and complaints at these events about users suffering conflicts and being forced to rename their machines have never been an issue.

This document advocates a single flat namespace for dot-local host names, (i.e. the names of DNS address records), but other DNS record types (such as those used by DNS Service Discovery [DNS-SD]) may contain as many labels as appropriate for the desired usage, subject to the 255-byte name length limit specified below in Section 3.3 "Maximum Multicast DNS Name Length".

Enforcing uniqueness of host names (i.e. the names of DNS address records mapping names to IP addresses) 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.

3.1 Governing Standards Body

Note that this use of the ".local." suffix falls under IETF/IANA jurisdiction, not ICANN jurisdiction. DNS is an IETF network protocol, governed by protocol rules defined by the IETF. These IETF protocol rules dictate character set, maximum name length, packet format, etc. ICANN determines additional rules that apply when the IETF's DNS protocol is used on the public Internet. In contrast, private uses of the DNS protocol on isolated private networks are not governed by ICANN. Since this proposed change is a change to the core DNS protocol rules, it affects everyone, not just those machines using the ICANN-governed Internet. Hence this change falls into the category of an IETF protocol rule, not an ICANN usage rule.

This allocation of responsibility is formally established in "Memorandum of Understanding Concerning the Technical Work of the Internet Assigned Numbers Authority" [RFC 2860]. Exception (a) of clause 4.3 states that the IETF has the authority to instruct IANA to reserve pseudo-TLDs as required for protocol design purposes. For example, "Reserved Top Level DNS Names" [RFC 2606] defines the following pseudo-TLDs:

```
.test
.example
.invalid
.localhost
```

3.2 Private DNS Namespaces

Note also that 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. There are also implementations for Linux and other platforms [dotlocal]. Operators setting up private internal networks ("intranets") are advised that their lives may be easier if they avoid using the suffix ".local." in names in their private internal DNS server. Alternative possibilities include:

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

Another alternative naming scheme, advocated by Professor D. J. Bernstein, is to use a numerical suffix, such as ".6." [djbdl].

3.3 Maximum Multicast DNS Name Length

RFC 1034 says:

"the total number of octets that represent a domain name (i.e., the sum of all label octets and label lengths) is limited to 255."

This text implies that the final root label at the end of every name is included in this count (a name can't be represented without it), but the text does not explicitly state that. Implementations of Multicast DNS MUST include the label length byte of the final root label at the end of every name when enforcing the rule that no name may be longer than 255 bytes. For example, the length of the name "apple.com." is considered to be 11, which is the number of bytes it takes to represent that name in a packet without using name compression:

```
  ------------------------------------------------------
  | 0x05 | a | p | p | l | e | 0x03 | c | o | m | 0x00 |
  ------------------------------------------------------
```

4. 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 clients that check the IP TTL on reception to determine whether the packet originated on the local link. These older clients 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 link-local multicast address) 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 sent to the link-local multicast address 224.0.0.251 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 sent to a unicast destination address, the source IP address in the packet is checked to see if it is an address on a local subnet. An 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.

Since queriers will ignore responses apparently originating outside the local subnet, responders 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, then even if the query indicates that a unicast response is preferred (see Section 6.5, "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.

5. 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 mDNS multicast address 224.0.0.251. 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.

6. Querying

There are three kinds of Multicast DNS Queries, one-shot queries of the kind made by today's conventional DNS clients, one-shot queries accumulating multiple responses made by multicast-aware DNS clients, and continuous ongoing Multicast DNS Queries used by IP network browser software.

A Multicast DNS Responder that is offering records that are intended to be unique on the local link 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.

6.1 One-Shot Multicast DNS Queries

An unsophisticated DNS client may simply send its DNS queries blindly to 224.0.0.251:5353, without necessarily even being aware 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 12 "Special Characteristics of Multicast DNS Domains") the query is sent to 224.0.0.251:5353 instead of the configured unicast DNS server address that would otherwise be used. Typically the timeout would also be shortened to two or three seconds, but it's possible to make a minimal mDNS client with no other changes apart from these.

A simple DNS client 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://cheshire.local." into their Web browser and gets to see the page they were hoping for, then the protocol has met the user's needs in this case.

While an unsophisticated DNS client like this is perfectly adequate for simple hostname lookup, it may not get ideal behavior in other cases. Additional refinements that may be adopted by more sophisticated clients are described below.

6.2 One-Shot Queries, Accumulating Multiple Responses

A more sophisticated DNS client should understand that Multicast DNS is not exactly the same as unicast DNS, and should modify its behavior in some simple ways.

As described above, there are some cases, such as looking up the address associated with a unique host name, where a single response

is sufficient, and moreover may be all that is expected. However,
there are other DNS queries where more than one response is

possible, and for these queries a more sophisticated Multicast DNS client should include the ability to wait for an appropriate period of time to collect multiple responses.

A naive DNS client retransmits its query only so long as it has received no response. A more sophisticated Multicast DNS client is aware that having received one response is not necessarily an indication that it might not receive others, and has the ability to retransmit its query an appropriate number of times at appropriate intervals until it is satisfied with the collection of responses it has gathered.

A more sophisticated Multicast DNS client that is retransmitting a query for which it has already received some responses, MUST implement Known Answer Suppression, as described below in Section 7.1 "Known Answer Suppression". This 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. In addition, when retransmitting queries, the interval between the first two queries SHOULD be one second, and the intervals between subsequent queries SHOULD double.

6.3 Continuous Multicast DNS Querying

In One-Shot Queries, with either a single or multiple responses, the underlying assumption is that the transaction begins when the application issues a query, and ends when all the desired responses have been received. There is another type of operation which is more akin to continuous monitoring.

iTunes users are accustomed to seeing a list of shared network music libraries in the sidebar of the iTunes window. There is no "refresh" button for the user to click because the list is always accurate, always reflecting the currently available libraries. When a new library becomes available it promptly appears in the list, and when a library becomes unavailable it promptly disappears. It is vitally important that this responsive user interface be achieved without naive polling that would place an unreasonable burden on the network.

Therefore, when retransmitting mDNS queries to implement this kind of continuous monitoring, the interval between the first two queries SHOULD be one second, the intervals between the subsequent queries SHOULD double, and the querier MUST implement Known Answer Suppression, as described below in Section 7.1. 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 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 re-query for records 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 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.

6.4 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.

6.5 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 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 6.3 "Continuous Multicast DNS Querying") 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 (see "Known Answer Suppression" below) 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 "Benefits of Multicast Responses" below). 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 issued by 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. 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 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.

Except when defending a unique name against a probe from another host unicast replies are subject to all the same packet generation rules as multicast replies, including the cache flush bit (see Section 11.3, "Announcements to Flush Outdated Cache Entries") and randomized delays to reduce network collisions (see Section 8, "Responding").

6.6 Delaying Initial Query

If a query is issued for which there already exist one or more records in the local cache, and those record(s) were received with the cache flush bit set (see Section 11.3, "Announcements to Flush Outdated Cache Entries"), indicating that they form a unique RRSet, then the host SHOULD delay its initial query by imposing a random delay from 500-1000ms. This is to avoid the situation where a group of hosts are synchronized by some external event and all perform the same query simultaneously. This means that when the first host (selected randomly by this algorithm) transmits its query, all the other hosts that were about to transmit the same query can suppress their superfluous queries, as described in "Duplicate Question Suppression" below.

6.7 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 6.5 "Questions Requesting Unicast Responses". 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, 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.

7. Duplicate Suppression

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 message with those answers.

A Multicast DNS Responder SHOULD 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 SHOULD delete that answer from the list of answers it is planning to give, provided that no other host on the network is also waiting to receive the same answer record.

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 wasting 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 send a query, and it sees another host on the network send a 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 clients 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. When multiple responders on the network have the same data, there is no need for all of them to respond.

This feature is particularly useful when multiple Sleep Proxy Servers are deployed (see Section 17, "Multicast DNS and Power Management").

In the future it is possible that every general-purpose OS (Mac, Windows, Linux, etc.) will implement Sleep Proxy Service as a matter of course. In this case there could be a large number of Sleep Proxy Servers on any given network, which is good for reliability and fault-tolerance, but would be bad for the network if every Sleep Proxy Server were to answer every query.

8. Responding

When a Multicast DNS Responder constructs and sends a Multicast DNS response packet, the Answer Section 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 Answer Section 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"), and the record rrclass must match the question qclass (unless the qclass is "ANY").

A Multicast DNS Responder MUST only respond when it has a positive non-null response to send. Error responses must never be sent. The non-existence of any name in a Multicast DNS Domain is ascertained by the failure of any machine to respond to the Multicast DNS query, not by NXDOMAIN errors.

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 [IEEE802] and similar shared multiple access networks SHOULD have the capability of delaying its responses by up to 500ms, as determined by the rules 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 with a positive non-null 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 immediately, 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 9.1, "Probing").

Responding immediately 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 immediately without delay is *not* appropriate for things like looking up PTR records used for DNS 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".

Except when a unicast response has been explicitly requested via the "unicast response" bit, Multicast DNS Responses MUST be sent to UDP port 5353 (the well-known port assigned to mDNS) on the 224.0.0.251 multicast address (or its IPv6 equivalent FF02::FB). Operating in a Zeroconf environment requires constant vigilance. Just because a name has been previously verified unique does not mean 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.

Sending all responses by multicast also facilitates opportunistic caching by other hosts on the network.

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 client on the network should have seen the previous transmission and cached it. A client 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.

8.2 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. Any (non-defensive) answers generated in response to query packets containing more than one question SHOULD be randomly delayed in the range 20-120ms, or 400-500ms if the TC (truncated) bit is set, as described above. (Answers defending a name, in response to a probe for that name, are not subject to this delay rule and are still sent immediately.)

8.2 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.

8.3 Legacy Unicast Responses

If the source UDP port in a received Multicast DNS Query is not port 5353, this indicates that the client originating the query is a simple client that does not fully implement all of Multicast DNS. In this case, the Multicast DNS Responder MUST send a UDP response directly back to the client, 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.

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 11 "Resource Record TTL Values and Cache Coherency" to update and invalidate stale data. Were unicast responses sent to legacy clients to use the same high TTLs, these legacy clients, which do not implement these cache coherency mechanisms, could retain stale cached resource record data long after it is no longer valid.

Having sent this unicast response, if the Responder has not sent this record in any multicast response recently, it SHOULD schedule the record to be sent via multicast as well, to facilitate passive conflict detection. "Recently" in this context means "if the time since the record was last sent via multicast is less than one quarter of the record's TTL".

Note that while legacy queries usually contain exactly one question, they are permitted to contain multiple questions, and responders listening for multicast queries on 224.0.0.251:5353 MUST be prepared to handle this correctly, responding by generating a unicast response containing the list of question(s) they are answering in the Question Section, and the records answering those question(s) in the Answer Section.

9. 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 an Ethernet "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.

9.1 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 its address record which maps its unique host name to its unique IP address. All Probe Queries SHOULD be done using the desired resource record name and query type T_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, 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 HINFO 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 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 this 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 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 for unique records, 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 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 first two 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. At the present time, this document recommends that the third probe SHOULD be sent as a standard "QM" question, for backwards compatibility with the small number of old devices still in use that don't implement unicast responses.

If, at any time during probing, from the beginning of the initial random 0-250ms delay onward, any conflicting Multicast DNS responses are received, then the probing host MUST defer to the existing host, and MUST choose new names for some or all of its resource records as appropriate, to avoid conflict with pre-existing hosts on the network. In the case of a host probing using query type T_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 failures 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, with absolute certainty, that its unique resource record set name, rrtype and rrclass cannot already be in use by any other responder on the network, then it MAY 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.

9.2 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 [RFC 2136].

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 T_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 treats this exactly as if it had received a positive answer to its query, and concludes that it may not use the desired name.

The determination of "lexicographically later" is performed by first comparing the record class, 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, 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:

```
cheshire.local. A 169.254.99.200
cheshire.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.

9.2.1 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 for TXT is 16) and the SRV record comes second (the rrtype 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.

9.3 Announcing

The second startup step is that the Multicast DNS Responder MUST send a gratuitous Multicast DNS Response containing, in the Answer Section, all of its 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 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 11.3 "Announcements to Flush Outdated Cache Entries".

The Multicast DNS Responder MUST send at least two gratuitous responses, one second apart. A Responder MAY send up to eight gratuitous Responses, provided that the interval between gratuitous responses doubles 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. It is not uncommon for protocol designers to encounter some problem which they decide to solve using regular periodic announcements, but this is generally not a wise protocol design choice. In the small scale periodic announcements may seem to remedy the short-term problem, but they do not scale well if the protocol becomes successful. If every host on the network implements the protocol -- if multiple applications on every host on the network are implementing the protocol -- then even a low periodic rate of just one announcement per minute per application per host can add up to multiple packets per second in total. While gigabit Ethernet may be able to carry a million packets per second, other network technologies cannot. For example, while IEEE 802.11g wireless has a nominal data rate of up to 54Mb/sec, multicasting just 100 packets per second can consume the entire available bandwidth, leaving nothing for anything else.

With the increasing popularity of hand-held devices, unnecessary continuous packet transmission can have bad implications for battery life. It's worth pointing out the precedent that TCP was also designed with this "no regular periodic idle packets" philosophy. Standard TCP sends packets only when it has data to send or acknowledge. If neither client nor server sends any bytes, then the

TCP code will send no packets, and a TCP connection can remain active in this state indefinitely, with no packets being exchanged for hours, days, weeks or months.

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

9.4 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.

In the case of shared records, a host MUST send a "goodbye" announcement with TTL zero (see Section 11.2 "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.

10. Conflict Resolution

A conflict occurs when a Multicast DNS Responder has a unique record for which it is 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 (gratuitous or otherwise) containing a conflicting resource record in the Answer Section, the Multicast DNS Responder MUST immediately reset its conflicted unique record to probing state, and go through the startup steps described above in Section 9. "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 is obtained.

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

o 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, appending the digit "2" to the name, or incrementing a number at the end of the name if one is already present.

o Probe again, and repeat until a unique name is found.

o Record this newly chosen name in persistent storage so that the device will use the same name the next time it is power-cycled.

o 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 iTunes music server on the network. Your music has been renamed to "Bob's Music (G4 Cube)". If you want to change this name, use [describe appropriate menu item or preference dialog].

o If after one minute of probing the Multicast DNS Responder has been unable to find any unused name, it should display a message to the user or operator informing them 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.

How the user or operator is informed 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 other 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.

The examples in this section focus on address records (i.e. host names), but the same considerations apply to all resource records where uniqueness (or maintenance of some other defined constraint) is desired.

11. 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 contained within the resource record's rdata (e.g. SRV, reverse mapping PTR record, etc.) is 120 seconds.

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

A client 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 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.

11.1 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:

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

   o 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 10 "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. Clients 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.

11.2 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 a gratuitous announcement 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.

Clients 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 11.1 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.

11.3 Announcements to Flush Outdated Cache Entries

Whenever a host has a resource record with potentially new data (e.g. after rebooting, waking from sleep, connecting to a new network link, changing IP address, etc.), the host MUST send a series of gratuitous announcements to update cache entries in its neighbor hosts. In these gratuitous announcements, if the record is one that is intended to be 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 the Answer Section of the DNS Response, it means, "This is an assertion that this information is true." When a resource record appears in the Answer 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 valid".

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 SHOULD 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 SHOULD set the cache flush bit on all members of the unique RRSet. In the event that for some reason the host chooses not to send the entire unique RRSet in a single packet or a rapid packet burst, it MUST NOT set the cache flush bit on any of those records.

The 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 apply regardless of *why* the response packet is being
generated. They apply to startup announcements as described in

Section 9.3 "Announcing", and to responses generated as a result of receiving query packets.

The "cache flush" bit is only set in records in the Answer Section 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 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 6.5, "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 the Answer 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.

11.4 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.

11.5 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 ARP/ND requests for that address go unanswered (for an address on a local subnet) or because a router returns an ICMP "Host Unreachable" error (for an address on a remote subnet).

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 in a reasonable amount of time, 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.

11.6 Passive Observation of Failures

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 a reasonable amount of time, then even though its TTL may indicate that it is not yet due to expire, that record MAY 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.

12. Special Characteristics of Multicast DNS Domains

Unlike conventional DNS names, names that end in ".local." or "254.169.in-addr.arpa." have only local significance. The same is true of names within the IPv6 Link-Local reverse mapping domains.

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.)

Multicast DNS Domains are not delegated from their parent domain via use of NS records. There are no NS records anywhere in Multicast DNS Domains. Instead, all Multicast DNS Domains are delegated to the IP addresses 224.0.0.251 and FF02::FB by virtue of the individual organizations producing DNS client software deciding how to handle those names. It would be extremely valuable for the industry if this special handling were ratified and recorded by IANA, since otherwise the special handling provided by each vendor is likely to be inconsistent.

The IPv4 name server for a Multicast DNS Domain is 224.0.0.251. The IPv6 name server 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.

No delegation is performed within Multicast DNS Domains. Because the cluster of loosely coordinated CPUs is cooperating to administer a single zone, delegation is neither necessary nor desirable. 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."

Multicast DNS Zones have no SOA 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. Multicast DNS for Service Discovery

This document does not describe using Multicast DNS for network browsing or service discovery. However, the mechanisms this document describes are compatible with (and support) the browsing and service discovery mechanisms proposed in "DNS-Based Service Discovery" [DNS-SD].

14. 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.

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." This in turn perpetuates the continued use of protocols like AppleTalk. If we want to retire AppleTalk, NetBIOS, etc., we need to offer users equivalent IP functionality that they can rely on to, "always work, like AppleTalk." A little Multicast DNS traffic may be a burden on the network, but it is an insignificant burden compared to continued widespread use of AppleTalk.

15. Considerations for Multiple Interfaces

A host SHOULD defend its host name (FQDN) 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 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.

16. 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.

16.1 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, 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.

16.2 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.

16.3 Efficiency

If different clients on a machine were to each 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.

16.4 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. Due to engineering constraints, there may be situations where embedding a Multicast DNS implementation in the client is the most expedient solution, and while this will work in practice, implementers should be aware of the issues outlined in this section.

17. Multicast DNS and Power Management

Many modern network devices have the ability to go into a low-power mode where only a small part of the Ethernet hardware remains powered, and the device can be woken up by sending a specially formatted Ethernet frame which the device's power-management hardware recognizes.

To make use of this in conjunction with Multicast DNS, we propose a network power management service called Sleep Proxy Service. A device that wishes to enter low-power mode first uses DNS-SD to determine if Sleep Proxy Service is available on the local network. In some networks there may be more than one piece of hardware implementing Sleep Proxy Service, for fault-tolerance reasons.

If the device finds the network has Sleep Proxy Service, the device transmits two or more gratuitous mDNS announcements setting the TTL of its relevant resource records to zero, to delete them from neighboring caches. The relevant resource records include address records and SRV records, and other resource records as may apply to a particular device. The device then communicates all of its remaining active records, plus the names, rrtypes and rrclasses of the deleted records, to the Sleep Proxy Service(s), along with a copy of the specific "magic packet" required to wake the device up.

When a Sleep Proxy Service sees an mDNS query for one of the device's active records (e.g. a DNS-SD PTR record), it answers on behalf of the device without waking it up. When a Sleep Proxy Service sees an mDNS query for one of the device's deleted resource records, it deduces that some client on the network needs to make an active connection to the device, and sends the specified "magic packet" to wake the device up. The device then wakes up, reactivates its deleted resource records, and re-announces them to the network. The client waiting to connect sees the announcements, learns the current IP address and port number of the desired service on the device, and proceeds to connect to it.

The connecting client does not need to be aware of how Sleep Proxy Service works. Only devices that implement low power mode and wish to make use of Sleep Proxy Service need to be aware of how that protocol works.

The reason that a device using a Sleep Proxy Service should send more than one goodbye packet is to ensure deletion of the resource records from all peer caches. If resource records were to inadvertently remain in some peer caches, then those peers may not issue any query packets for those records when attempting to access the sleeping device, so the Sleep Proxy Service would not receive any queries for the device's SRV and/or address records, and the necessary wake-up message would not be triggered.

The full specification of mDNS / DNS-SD Sleep Proxy Service is described in another document [not yet published].

18. Multicast DNS Character Set

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, with no spaces or other punctuation. Attempts to remedy this for unicast DNS have been badly constrained by the need to accommodate old buggy legacy DNS implementations. In reality, the DNS specification actually imposes no limits on what characters may be used in names, and good DNS implementations handle any arbitrary eight-bit data without trouble. However, the old rules for ARPANET host names back in the 1980s required names to be just letters, digits, and hyphens [RFC 1034], 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 would be more accurate to say that certain bad implementations may not handle eight-bit data correctly, not that the protocol doesn't support it.

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 are encoded using precomposed UTF-8 [RFC 3629]. The characters SHOULD conform to Unicode Normalization Form C (NFC) [UAX15]: Use precomposed characters instead of combining sequences where possible, e.g. use U+00C4 ("Latin capital letter A with diaeresis") instead of U+0041 U+0308 ("Latin capital letter A", "combining diaeresis").

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 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).

This point bears repeating: After many years of debate, as a

result of the need to accommodate certain DNS implementations that
apparently couldn't handle any character that's not a letter, digit
or hyphen (and apparently never will be updated to remedy this

limitation) the unicast DNS community settled on an extremely baroque encoding called "Punycode" [RFC 3492]. 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" 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.

The simple rules for case-insensitivity in Unicast DNS 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 client issues a query for an address record with the name "cheshire.local", then a responder having an address record with the name "Cheshire.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 client 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.

19. Multicast DNS Message Size

RFC 1035 restricts DNS Messages carried by UDP to no more than 512 bytes (not counting the IP or UDP headers) [RFC 1035]. 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, sent 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.

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.

20. Multicast DNS Message Format

This section describes specific restrictions on the allowable values for the header fields of a Multicast DNS message.

20.1 ID (Query Identifier)

Multicast DNS clients SHOULD listen for gratuitous responses issued by hosts booting up (or waking up from sleep or otherwise joining the network). Since these gratuitous responses may contain a useful answer to a question for which the client is currently awaiting an answer, Multicast DNS clients 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 clients 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 gratuitous multicast responses, the Query ID MUST be set to zero on transmission, and MUST be ignored on reception.

In 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.

20.2 QR (Query/Response) Bit

In query messages, MUST be zero.
In response messages, MUST be one.

20.3 OPCODE

In both multicast query and multicast response messages, MUST be zero (only standard queries are currently supported over multicast, unless other queries are allowed by future IETF Standards Action).

20.4 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 implies there's some other place where "better" information may be found) and MUST be ignored on reception.

20.5 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 MAY choose to 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 client SHOULD re-issue its query using TCP in order to receive the larger response.

20.6 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.

20.7 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.

20.8 Z (Zero) Bit

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

20.9 AD (Authentic Data) Bit [RFC 2535]

In query messages the Authentic Data bit MUST be zero on transmission, and MUST be ignored on reception.

In response messages, the Authentic Data bit MAY be set. Resolvers receiving response messages with the AD bit set MUST NOT trust the AD bit unless they trust the source of the message and either have a secure path to it or use DNS transaction security.

20.10 CD (Checking Disabled) Bit [RFC 2535]

In query messages, a resolver willing to do cryptography SHOULD set the Checking Disabled bit to permit it to impose its own policies.

In response messages, the Checking Disabled bit MUST be zero on transmission, and MUST be ignored on reception.

20.11 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.

20.12 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.

20.13 Repurposing of top bit of rrclass in Answer Section

In the Answer Section 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).

21. Choice of UDP Port Number

Arguments were made for and against using Multicast on UDP port 53. The final decision was to use UDP port 5353. Some of the arguments for and against are given below.

21.1 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 clients 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.

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

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

* Changing client code to use a different port number is not hard.

* 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 is important to allow 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 the user becomes unable to access their local network printer sitting on their desk right next to their computer. Using a different UDP port eliminates this particular problem.

* On many operating systems, unprivileged clients may not send or receive packets on low-numbered ports. This means that any client 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 eliminates this particular problem.

22. Summary of Differences Between Multicast DNS and Unicast DNS

The value of Multicast DNS is that it 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 authoritarian hierarchy controlled by a strict chain of formal delegations from the top. These differences are listed 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
* defines a clear limit on the maximum legal domain name (255 bytes)
* allows larger UDP packets
* allows more than one question in a query packet
* 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 gratuitous responses to announce new records to the peer group
* defines a "unicast response" bit in the rrclass of query questions
* defines a "cache flush" bit in the rrclass of response answers
* uses DNS TTL 0 to indicate that a record has been deleted
* monitors queries to perform Duplicate Question Suppression
* monitors responses to perform Duplicate Answer Suppression...
* ... and Ongoing Conflict Detection
* ... and Opportunistic Caching

23. Benefits of Multicast Responses

Some people have argued that sending responses via multicast is inefficient on the network. In fact using multicast responses results in a net lowering of overall multicast traffic, for a variety of reasons, in addition to other benefits.

* One multicast response can update the cache 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.

* 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.

* 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 significantly higher packet rate.

* Multicast responses allow passive conflict detection. Without this ability, some other conflict detection mechanism would be needed, imposing its own additional burden on the network.

* When using delayed responses to reduce network collisions, clients need to maintain a list recording to whom each answer should be sent. The option of multicast responses allows clients 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.

* 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.

* 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.

24. 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".

24.1 IPv6 Multicast Addresses by Hashing

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.

There are some problems with this:

* When a host has a large number of records with different names, the host may have to join a large number of multicast groups. This 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 software for filtering, thereby defeating the point of using a multicast address range in the first place.

* 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.

25. 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 administered DNS.

In an environment where there is a group of cooperating participants, but there may be other antagonistic participants on the same physical link, the cooperating participants need to use IPSEC signatures and/or DNSSEC [RFC 2535] 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).

When 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.

26. IANA Considerations

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

IANA has allocated the IPv6 multicast address set FF0X::FB for the use described in this document. 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).

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").

The re-use of the top bit of the rrclass field in the Question and Answer 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. However, since 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, this limitation is likely to remain a purely theoretical one.

No other IANA services are required by this document.

27. Acknowledgments

The concepts described in this document have been explored, developed and implemented with help from Freek Dijkstra, Erik Guttman, Paul Vixie, Bill Woodcock, and others.

Special thanks go to Bob Bradley, Josh Graessley, Scott Herscher, Roger Pantos and Kiren Sekar for their significant contributions.

28. Deployment History

Multicast DNS client software first became available to the public in Mac OS 9 in 2001. Multicast DNS Responder software first began shipping to end users in large volumes (i.e. millions) with the launch of Mac OS X 10.2 Jaguar in August 2002, and became available for Microsoft Windows users with the launch of Apple's "Rendezvous for Windows" (now "Bonjour for Windows") in June 2004.

Apple released the source code for the mDNSResponder daemon as Open Source in September 2002, first under Apple's standard Apple Public Source License, and then later, in August 2006, under the Apache

   License, Version 2.0.

In addition to desktop and laptop computers running Mac OS X and Microsoft Windows, Multicast DNS is implemented in a wide range of hardware devices, such as Apple's "AirPort Extreme" and "AirPort Express" wireless base stations, 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.

29. Copyright Notice

30. Normative References

[RFC 1034] Mockapetris, P., "Domain Names - Concepts and Facilities", STD 13, RFC 1034, November 1987.

[RFC 1035] Mockapetris, P., "Domain Names - Implementation and Specifications", STD 13, RFC 1035, November 1987.

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

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

[UAX15] "Unicode Normalization Forms" http://www.unicode.org/reports/tr15/

31. Informative References

[dotlocal] <http://www.dotlocal.org/>

[djbdl] <http://cr.yp.to/djbdns/dot-local.html>

[DNS-SD] Cheshire, S., and M. Krochmal, "DNS-Based Service Discovery", Internet-Draft (work in progress), draft-cheshire-dnsext-dns-sd-04.txt, August 2006.

[IEEE802] IEEE Standards for Local and Metropolitan Area Networks: Overview and Architecture. Institute of Electrical and Electronic Engineers, IEEE Standard 802, 1990.

[NBP] Cheshire, S., and M. Krochmal, "Requirements for a Protocol to Replace AppleTalk NBP", Internet-Draft (work in progress), draft-cheshire-dnsext-nbp-05.txt, August 2006.

[RFC 2136] Vixie, P., et al., "Dynamic Updates in the Domain Name System (DNS UPDATE)", RFC 2136, April 1997.

[RFC 2462] S. Thomson and T. Narten, "IPv6 Stateless Address Autoconfiguration", RFC 2462, December 1998.

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

[RFC 2606] Eastlake, D., and A. Panitz, "Reserved Top Level DNS Names", RFC 2606, June 1999.

[RFC 2860] Carpenter, B., Baker, F. and M. Roberts, "Memorandum of Understanding Concerning the Technical Work of the Internet Assigned Numbers Authority", RFC 2860, June 2000.

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

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

[ZC] Williams, A., "Requirements for Automatic Configuration of IP Hosts", Internet-Draft (work in progress), draft-ietf-zeroconf-reqts-12.txt, September 2002.

32. Authors' Addresses

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

Phone: +1 408 974 3207
EMail: rfc [at] stuartcheshire [dot] org

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

Phone: +1 408 974 4368
EMail: marc [at] apple [dot] com