dnssd
Internet-Draft
Intended status: Standards Track
Expires: 24 October 2026

T. Lemon
Apple Inc.
E. Dijk
IoTconsultancy.nl

22 April 2026

# Multicast DNS conflict resolution using the Time Since Received (TSR) EDNS option
## draft-ietf-dnssd-tsr-02

## Abstract

This document specifies a new conflict resolution mechanism for DNS, for use in cases where the advertisement is being proxied, rather than advertised directly, e.g. when using a combined DNS-SD advertising proxy and SRP registrar. A new EDNS option is defined that communicates the time at which the set of resource records on a particular DNS owner name was most recently updated.

## About This Document

This note is to be removed before publishing as an RFC.

The latest revision of this draft can be found at https://datatracker.ietf.org/doc/html/draft-ietf-dnssd-tsr. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-ietf-dnssd-tsr/.

Discussion of this document takes place on the WG Working Group mailing list (mailto:dnssd@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/dnssd/. Subscribe at https://www.ietf.org/mailman/listinfo/dnssd/.

Source for this draft and an issue tracker can be found at https://github.com/dnssd-wg/draft-ietf-dnssd-tsr.

## Status of This Memo

## Copyright Notice



## Table of Contents

## 1. Introduction

Unlike the Domain Name System [RFC1034], with its authority servers and delegation of authority, Multicast DNS has no single source of authority. Because of this, mDNS has a mechanism, conflict resolution (Section 9 of [RFC6762]) for detecting and fixing conflicts in mDNS advertisements.

The current goal of mDNS conflict resolution is to prevent a newly advertised service from taking the place of an existing service with the same name that is already being advertised. This goal, however, assumes that the entity advertising an mDNS service is in fact authoritative for that service. For mDNS proxies, such as an advertising proxy [I-D.ietf-dnssd-advertising-proxy], this is not the case: the source of truth for the service being advertised is an DNSSD Service Registration Protocol (SRP) [RFC9665] requester.

On a link with more than one SRP registrar, an SRP requester may register with one SRP registrar, and then subsequently update its registration on a different SRP registrar. Both SRP registrars may be acting as advertising proxies. If so, the original SRP registrar may still be advertising the old SRP registration using mDNS. If the information in the new SRP registration is identical to that in the old registration, this is often not a problem. However if some information has changed (e.g., a new IP address has been added, or a TXT record updated), then the new registration will be seen to be in conflict with the old registration. In addition, the method used in mDNS to detect conflicts can sometimes produce apparent conflicts where no actual conflict exists because of the way records in mDNS packets are marshalled.

In the case of such an apparent conflict, the current behavior of mDNS is for the older (stale) registration to win, and the newer (current) information to be discarded. This behavior, which is entirely correct for services that are advertising on their own behalf, is exactly wrong when a service advertisement is being proxied.

### 1.1. Current Behavior

When a new service is to be advertised, the registrant (the entity requesting the registration) typically registers the service with a central mDNS registrar on the host on which it is running. This mDNS registrar locally stores services that have already been registered, and may detect a conflict with one of those registered services. This can be true whether the conflicting service entry is data for which the mDNS registrar is authoritative (stored in its local registration database), or an entry it has received via mDNS (stored in its cache).

In the case of such a conflict, no network transaction is required: the mDNS registrar detects it locally. It addresses the conflict in one of two ways. The first alternative is that the mDNS registrar will report the conflict to the registrant as an error, which it must fix. Alternatively, if the registrant has indicated that the mDNS registrar should automatically choose a new name for the service in case of conflict, the mDNS registrar does so automatically, without necessarily notifying the registrant.

Once any locally-detectable conflicts have been resolved, the mDNS registrar probes (see Section 8.1 of [RFC6762]) the local network to see if any other host has already registered a service name that conflicts with the proposed new service name. If such a service name is present on the network, the mDNS registrar follows the same process previously described: either report the error to the mDNS registrant or automatically choose a new name.

The effect of this approach is that generally whichever registrant first registers a service under a particular name wins. If another registrant comes along later and registers the same service name with conflicting service information, the newcomer’ s information is rejected.

### 1.2. Problem Statement

The current behavior works well for registrants registering services on their own behalf. However, for example in the case of an SRP registrar, it works poorly: an SRP registrar acting as an advertising proxy publishes the contents of its DNS zone using mDNS. The sources of truth for this information are the SRP requesters, not the SRP registrar itself. The SRP registrar in this case acts as a proxy for the SRP requesters.

In the case of an advertising proxy publishing records originating from an SRP Update, the most recent information is correct; the older information is simply stale, and not competing. When the SRP requester is able to continue registering with the same SRP registrar, this works well: stale data is automatically removed and replaced with current data. However, if more than one SRP registrar is available, the SRP requester may wind up sending its SRP Updates to a different SRP registrar. This can happen as a result of a network partition event, or in cases where the SRP registrar is contacted using an anycast address.

When the SRP requester sends its SRP Update to a different SRP registrar, the behavior of the mDNS conflict resolution approach without TSR is that the SRP requester’s service will be given a new name, and both the old (stale) service advertisement (A) and the new (more recent) service advertisement (A’ ) will be discoverable as separate services.

This creates a new burden on consumers of such services: they need to parse through the whole list of services of their type, using metadata from the TXT record in the service instance data, if possible, to determine that service A and service A’ are the same service. If no such information is present in the TXT record, the only way to determine that one of these two advertised services is stale is to attempt to use the advertised service, which may no longer be reachable if, for example, the change that produced the conflict was an IP address change. When the SRP lease for the stale service expires, that service’s mDNS advertisement will be removed, and the service will no longer be discoverable under the original name, even if its IP address hasn’t changed.

This document proposes an enhancement to the current conflict resolution algorithm for mDNS, which allows an mDNS proxy to report the time at which it received the registration for DNS records it is advertising, and the source from which these records were received. This is done using a new Time Since Received (TSR) EDNS(0) ([RFC6891]) option, of which there must be exactly one per name being advertised by the mDNS proxy.

### 1.3. Conventions, Terms and Definitions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

*absolute time* a node-local timestamp that is derived from a local monotonic clock. This clock is not required to be synchronized with any other clock, nor is it an absolute time in the sense of UTC. The clock may restart upon boot of the node but not during regular operation.

*authoritative data* DNS records stored in the local registration database, for which an mDNS registrar is authoritative (as defined by [RFC6762]).

*cache* each mDNS registrar maintains a cache of DNS records received from hosts on the local network, as described in [RFC6762]. Information in the cache represents the mDNS registrar’s current understanding of what records are advertised on the network. Authoritative data may be present in the cache, but the presence of data in the cache does not in itself indicate that the mDNS registrar is authoritative for that data.

*local registration database* a database of DNS records maintained by the mDNS registrar. Records in this database are registered locally with the mDNS registrar. The mDNS registrar is authoritative for all DNS records in this database. When the mDNS registrar receives queries that match records in this database, the mDNS registrar can respond to these queries with the matching records.

*mDNS registrar* an mDNS [RFC6762] implementation on a host that accepts local requests for querying, advertising and registering DNS records from one or more requesters and/or registrants. This could for example be an mDNS daemon process running in an operating system, accepting API calls from local processes to register or update DNS records for that process. It stores the locally-registered records in its local registration database.

*mDNS registrant* an entity or software process that is attempting to register records for advertisement to a (local) mDNS registrar.

*mDNS requester* an entity or software process that issues mDNS

queries to a local mDNS registrar, via an API call. The mDNS registrar is responsible for executing these queries and notifying the mDNS requester about the answer(s).

*mDNS proxy* a host that runs an mDNS registrar and at least one mDNS registrant acting as a proxy. That is, it needs to advertise mDNS records on behalf of one or more entities not located on the host itself. The advertising proxy [I-D.ietf-dnssd-advertising-proxy] is an example of an mDNS proxy.

*SRP requester* a client that uses the Service Registration Protocol (SRP) [RFC9665] to send an SRP Update to an SRP registrar.

*SRP registrar* a server that accepts SRP Updates sent by SRP requesters using the SRP [RFC9665]. DNS records registered via SRP to an SRP registrar may then be advertised by mDNS using an advertising proxy [I-D.ietf-dnssd-advertising-proxy] located on the same host. In that case, the SRP registrar process acts as an mDNS registrant towards its local mDNS registrar process.

*TSR data* locally stored data, associated with a single DNS owner name, that keeps track of the absolute time when a set of resource records were last updated and includes a key checksum to identify the owner of these records.

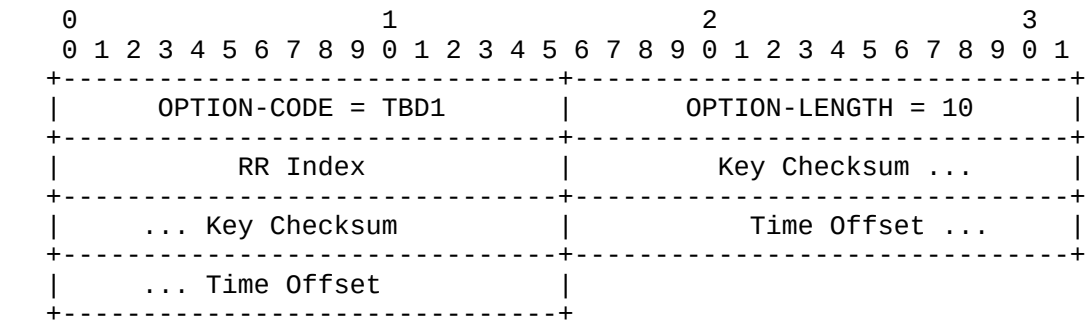
*TSR option* an EDNS0 Time Since Received (TSR) option as defined by this specification.

*TSR time* the (node-local) absolute time indicated by a particular TSR data.

## 2. Time Since Received EDNS Option

Each Time Since Received (TSR) EDNS option is applicable to exactly one DNS owner name. So all the records for that owner name that appear in the answer, authority and/or additional sections of an mDNS message would be covered by a single TSR option.

The TSR EDNS option has the following format:




The TSR EDNS option includes three fields in the OPTION-DATA ([RFC6891]): 'RR Index' (two byte integer in network byte order), 'Key Checksum' (four bytes), and 'Time Offset' (four bytes).

The 'RR Index' is the number of the RR in the mDNS packet. Question RRs are not counted. So if the message includes two answer RRs, one authority RR and two additional RRs, an index of 0 would refer to the first answer, an index of 1 to the second answer, and index of 2 to the single authority record, and so on. Questions are excluded because they have no data associated with them, and so it makes no sense for them to have TSR options associated with them.

If there is more than one record in the mDNS Message with the same owner name, only one TSR option is emitted for that name, and it applies to every RR in the mDNS Message with that owner name. It is not possible in the SRP protocol for two updates at two different times to contain records that apply to the same name: in such a situation, the second update completely replaces the first, so all data in the first update is then rendered stale.

The second field, 'Key Checksum', is a simple 32-bit checksum of the public key that the owner of the data (for example the SRP requester) used to authenticate itself. The key checksum is computed by treating the key as a series of 32-bit unsigned integers in network byte order, and adding these integers together to produce a 32-bit unsigned checksum. Overflow is not considered. This checksum need not be cryptographically secure: mDNS messages are not authenticated, so an attacker on the local link can always cause problems with mDNS by providing spurious responses. The purpose of the checksum is simply to notice whether, for a specific owner name, two different authoritative sources have provided information.

The 'Time Offset' field contains the difference, in seconds, between the the time at which the TSR option is being generated and the time of receipt of resource records for that owner name.

The time offset is represented in the mDNS message as a time in seconds relative to the time when the mDNS message is sent. If this difference is greater than seven days (7 * 24 * 60 * 60), the mDNS registrar MUST use a value of seven days rather than the larger value. The relative time value in the TSR option is converted to an absolute (local) time when stored in a cache or a local registration database on an mDNS registrar as TSR data, and is converted back to a relative time whenever an mDNS message with a TSR option is generated from local data.

# 3. mDNS Registrar Behavior

## 3.1. Validating requested local RR registrations that include a TSR option

When a local mDNS registrant asks an mDNS registrar to register one or more records on an owner name, and provides TSR data for that name, the mDNS registrar first checks that none of the records are marked shared. If any record is marked shared, the mDNS registrar MUST respond with an error indicating that the registration is invalid.

It then checks to see if there are any records either in cache or in its local registration database on that owner name. If no such data exists, the mDNS registrar stores the record(s) in this registration in its local registration database and puts them in the probing state.

When such data exists, the registrar MUST check to see if it has TSR data for that owner name. If it does not, or if there is TSR data on that name but the key checksum does not match, the registrar MUST treat this registration as a conflict and return an appropriate error to the registrant.

If such data exists and the key checksums match, there are three possibilities based on the known TSR time (from the existing data) and the proposed TSR time (from the TSR data in the registration request):

*Known TSR time is more recent* In this case, the registrar MUST treat the requested data as stale, and return an indication to the registrant that its registration is stale. This indication must be distinct from the "appropriate error" indication of conflict that was defined above.

*Both TSR times are the same* In this case, cached data (if any) on

the owner name is discarded first. Then, the requested records are added to the local registration database, updating existing records (if any). Since the requested records are considered identical to the known records, based on TSR time, no probing or announcing is performed for these records.

*Proposed TSR time is more recent* In this case, cached data (if any) on the owner name is discarded first. The registrant for any existing locally-registered data is notified that the data they had previously registered is stale, and the stale data is removed from the local registration database. The requested records are added to the local registration database and put in the probing state, and the TSR data is replaced by the proposed TSR data.

It is in principle possible for two different mDNS registrants to ask the same mDNS registrar to publish different RRs on the same name, some of which are shared and some of which are unique (see Section 2 of [RFC6762]). If an mDNS registrant tries to register an RR on a name for which the registrar already has data, cached or authoritative, on the same name, whether of the same RR type or a different RR type, for which there is no TSR data, or for which the key checksum in the TSR data being registered does not match what is already known, the registrar MUST treat this as an immediate conflict, and MUST NOT add the data to its local registration database and MUST NOT probe.

For the third case, as with any local mDNS registration, the mDNS registrar treats all of the records in the registration request as tentative (that is, they are put in the probing state) until they have been probed and no conflicting answers from other mDNS hosts have been received.

### 3.2. Probing resource records on names for which TSR data has been proposed

Section 8.1 of [RFC6762] describes how an mDNS registrar probes to ensure that there is no conflicting data for records in the probing state. The behavior for records that are in the probing state with owner names to which no TSR data applies remains unchanged. When there is TSR data on a name for which records are being probed, the mDNS registar MUST include TSR options for each such name as described in Section 2. Handling of responses is described in Section 3.5.

### 3.3. Processing mDNS questions for which TSR data exists

When processing a question for which local TSR data is present, the mDNS registrar MUST first check to see if there is corresponding data in the mDNS message being processed. In this case, before constructing a response, the mDNS registrar MUST process the non-question records in the packet, since this could result in stale data being flushed. Processing is performed as described in Section 3.5.

Once all non-question records have been processed, the registrar MUST respond to any questions that match locally-registered resource records for which a known answer is not present in the query. Responses are constructed as described in Section 2.

### 3.4. Receiving mDNS messages that may contain TSR options

mDNS registrars that support TSR need to compute an absolute time based on a time offset. This means that registrars need to know when the packet was received. A naive implementation might assume that the time that the packet is read off the input queue by the registrar is close enough. However, in practice it can be the case on a heavily loaded system that the time of receipt and the time of processing are far enough apart to create the appearance of staleness.

To avoid this, mDNS registrars that have an API available to get the actual time of receipt of a packet should make use of that API. For example, the SO_TIMESTAMP_CONTINUOUS socket option is available on Linux and BSD Unix platforms, including MacOS. When such APIs are not available, another option is to receive such packets on a high priority thread and queue them for later processing.

### 3.5. Processing mDNS messages thay may contain TSR options

mDNS registrars that support the TSR option MUST check incoming messages for the presence of an EDNS(0) option containing TSR options. mDNS registrars that do not support TSR will not do this check, and will behave as if no TSR options are present. For non-proxy use cases, this should make no difference, since in such cases if multiple devices advertise records on the same owner name, these are actually in conflict. However, for the proxy use case, what this means is that two proxies that are proxying the same data cannot interoperate if one supports TSR and the other doesn’t.

It is important to note that mDNS messages, particularly in the case of proxies, can contain combined information answering multiple queries that may be outstanding, and as such, it’s entirely possible for a mDNS message sent by an mDNS registrar that supports TSR to

contain some answers for which there is TSR data, and some answers for which there is not. It's equally possible that such a registrar will send mDNS packets containing no TSR options at all.

When an mDNS message contains TSR options, for each TSR option in an mDNS message, the mDNS registrar first determines the owner name of the TSR option by assigning an index to each non-question resource record in the mDNS message. The 'RR index' of each TSR option is then matched to the index of a resource record, and the owner name for that resource record is applied to the TSR option. The time on the TSR option is then computed by taking the current local clock time and subtracting from it the 'Time Offset' value in the TSR option.

If there is a TSR option in an mDNS message for which there is no matching resource record in the mDNS message, the mDNS registrar MUST ignore that TSR option. The mDNS registrar MUST NOT use the 'RR Index' value in the TSR option to search across the mDNS packet since such an index can easily be out of bounds.

Now, for each record in the mDNS message, the mDNS registrar first determines whether the record is an OPT record, is in the question section, or is a known answer (QD bit = 0 and it's a record in the answer section). For all such records, no special processing is done for TSRs, since no TSR should exist in the mDNS message.

For each remaining resource record in the mDNS message, the mDNS registrar MUST check to see if there is a TSR option in the mDNS message for that owner name. If there is not, the mDNS registrar MUST check to see if there is TSR data with that owner name locally. If there is not, the record is processed normally.

If there is local TSR data for the record's owner name, but no TSR data for that owner name in the mDNS message, the mDNS registrar checks to see if there are any resource records in the local registration database on that name. If there are, all such records are treated as in conflict. This conflict exists even if the locally registered records are all shared records. In cases where there are records on the name in the cache, those records are all discarded, because they are in conflict with the new data.

In the case that there is TSR data for the record in the mDNS message, and there are local records on the same owner name for which there is no local TSR data, this always means that any data is in conflict. How that conflict is addressed depends on the data. First, note that resource records in the answer section of an mDNS Query (QR bit in the header is 0) are "known answers" and therefore are not relevant when adding data to the cache. Such records can

never have TSR options associated with them. However, resource records in the authority and additional sections of a query do need to be processed (but in the case of authority records, are not added to the cache).

In cases where the TSR data for a particular name is present both locally and in the mDNS message, the mDNS registrar MUST compare the key checksums. If these are different, then the records are always in conflict, and are handled according to the context of the conflict, as described in Section 9 of [RFC6762].

In cases where the key checksums match, the mDNS registrar MUST compare the times. When the TSR time from the mDNS message is more recent than the local TSR time, the local data in the cache is flushed. Stale data in the local registration database is removed, and the mDNS registrant is informed that this data is stale.

When the TSR times are the same, any resource records on that name in the answer section and additional section are added to the cache.

When the local TSR time is more recent, the data in the message is not added to the cache, and no action is taken with respect to any locally-registered data.

### 3.6. Duplicate answer suppression behavior

Section 7.4 of [RFC6762], Duplicate Answer Suppression, describes behavior intended to prevent the redundant transmission of duplicate answers. When an mDNS query is received for which the mDNS registrar has authoritative data, the mDNS registrar will wait for a random amount of time before sending a response. If, during that time, a response is received that contains all the answers it would have sent, it suppresses sending these answers, since they are redundant. Such a response is referred to as a "pre-empting response".

In the case where TSR data is present locally or in a pre-empting response, it can be the case that the data in a pre-empting response is stale or conflicting. For this reason, an mDNS registrar MUST NOT suppress duplicate answers when:

* the TSR key checksum for the owner name in a pre-empting response does not match the local TSR key checksum for that owner name

* TSR data is present locally for that owner name but is not present in the pre-empting response

* A TSR option is present for that owner name in the pre-empting response, but local authoritative data for that owner name is present but has no associated TSR data

* local authoritative data for that owner name is present and includes TSR data, a TSR option for that owner name is present in the pre-empting response, but the local TSR time for that owner name is more recent than the TSR time for that owner name in the pre-empting response

Appendix A shows what this might look like from the perspective of an mDNS requester.

### 3.7. Suppression of Goodbye announcements

When authoritative data is removed on an mDNS registrar because an mDNS message has been received with more recent data, the mDNS registrar MUST NOT send a "goodbye" announcement for any RR on that owner name as a result of flushing this stale data.

The reason for this is that in the case where an mDNS registrant updates one or more RRsets on an owner name covered by TSR data, and as a result of this some records are removed, but some remain, the mDNS registrar that was directly updated will either send "goodbye" announcement or an announcement with the cache flush bit set as specified in Section 8.4 of [RFC6762]. Since the mDNS registrar with the most current information has already done what is needed, if an mDNS registrar that is flushing locally-registered data were to send a "goodbye" announcement this would at best be redundant and hence wasteful use of multicast, and at worse might cause valid data to be flushed from the cache on some mDNS registrar.

### 3.8. Suppression of redundant probing

When mDNS proxies are doing any form of replication of the records they are publishing, it can be the case that one proxy sends its probes for a new set of records first. If this is the case, some or all proxies on the same multicast link that receive the replicated data may already have (at the time of probing) the correct data in their local registration database with matching TSR times, waiting to be probed.

In such case, these proxies will determine that the records in the received probe message are identical to the locally-registered set of records, and therefore per Section 8.2.1 of [RFC6762] no conflict is detected and no response to the probe will be sent. An mDNS registrar that receives a probe, for a new set of records that it has just placed in the probing state, MUST suppress sending its own probe messages as this would be redundant: the probing is already being performed on the link.

[TBD for WG: for the above, the response to a probe in case of a true conflict (by another mDNS host) will typically be ’fast’ with unicast as per RFC 6762. For this reason, the suppression of probing would be dangerous in the above case, as most of the proxies that suppress their probing would not see the actual conflict (if any). Unless the replication algorithm takes care of withdrawing the records on all proxies.]

Conversely, it can also be the case that one proxy already sent its mDNS probes and announced the new records before some other proxy on the same multicast link fully completed the replication process for these records. In that case, the announced records will be stored already in the cache of this other proxy at the time that its local mDNS registrant (i.e. the replicating proxy process) performs the registration of these records and the associated probing.

To avoid redundant probing for such cases, when an mDNS registrant registers data with an mDNS registrar for which the same data is already cached with the same TSR key checksum and an equal TSR time, the mDNS registrar MUST skip probing. This requirement follows the second case ("Both TSR times are the same") in Section 3.1.

[TBD for WG: the below ’recent’ rule text instead of ’equal’ as used in above sentence seems to conflict the section ’locval’ Section 3.1 3 cases. Only when TSR time is equal, will the probing be suppressed.

Recent here should take into account network delays: a difference of less than ten seconds between the cached TSR time and the registrant’s TSR time should be considered "recent." ]

In addition, when the TSR time for a set of RRs is updated by an mDNS registrant, but nothing else changes, the mDNS registrar MUST NOT re-probe those RRs. In this situation, if some RRs are removed, then a goodbye announcement should be sent for such RRs, but no probe is sent for RRs that are not removed. If some RRs are added, the probing stage can be skipped because the registrar already knows it is up to date. So the new RRs can simply be announced immediately. One can assume that updates do not happen frequently enough for there to be competing mDNS updates being probed or announced at the same time.

### 3.9. Constructing a mDNS message with TSR options

For each non-question record that is added to the mDNS message, one of three things must be true:

* The mDNS registrar has locally-registered resource record(s) on that owner name, which may or may not be in the probing state.

* It is sending an answer which is either an announcement or a response containing data it has already validated and for which it is authoritative.

* The message is a query (QD=0) and the record is in the answer section, and is therefore a "known answer."

As described in Section 7.1 of [RFC6762], an mDNS registrar asking a question about one or more RRs on a particular name populates the answer section of its mDNS message with the answers it already knows, to avoid unnecessary responses. However, in this case it can’t also be probing for records on the same name, because probes are only done for unique (non-shared) records.

The requirements in Section 3.1 mean that there can never be an mDNS probe that contains known answers on an owner name for which any RR is being probed to which a TSR option applies.

This means that for any particular owner name that might be represented in an mDNS packet, it must be the case either that it is not a known answer, or that it is a known answer and no other records exist in the mDNS packet with the same owner name to which a TSR option would apply. That is, one of two things must be true about the set of all records with a particular owner name being added to the mDNS packet: either a TSR option applies to all of the records, or it applies to none of the records. Furthermore, either a record is a known answer from cache, or it is a locally-registered record.

When constructing an mDNS message, the mDNS registrar maintains a set of names and associated TSR data. Initially this set is empty. When the registrar adds a record to the mDNS message, if that record is locally registered, and if the registrar has TSR data for that name, it first checks to see if it has already added TSR data for that name to the set. If not, then it adds a new entry to the set containing the TSR data for the owner name of the RR. The data added consists of the owner name, the index of the record being added (since it is the first), the key checksum, and the time offset.

Once the mDNS registrar has finished adding resource records to the mDNS message, it adds an OPT record in the additional section. In this OPT record it adds a TSR option for every name in the set that was generated when adding resource records to the message. The time of receipt is subtracted from the current time to produce the value for the 'Time Offset' field, and this value is clamped to a maximum of seven days (604,800 seconds) Section 2.

## 4. The effect of network latency on time computations

Because TSR computations are affected by network latency, comparisons can' t be considered accurate. It is therefore necessary to tolerate some amount of error. In practice, however, it should generally not be the case that two advertising proxies receive SRP updates from the same SRP requester at nearly the same time. So it should always be the case either that there is a clear ordering to the TSR 'Time Offset' values, or that there is no conflict in the data. For example with anycast, a retransmission could go to a different SRP registrar, but in this case both SRP registrars would simultaneously receive identical data, so the close ordering or even equality of the TSR time offsets should not affect the outcome.

## 5. Internal Handling of TSR data

The TSR 'Time Offset' value that is sent on the wire is expressed in seconds relative to the time of receipt of the registration. In order to derive this value, the mDNS registrar must remember the (local) time at which the registration occurred. This time is recorded as an absolute time, not a relative time. We refer to this as the time of receipt. When constructing a TSR option, the registrar computes the difference between the current time and the time of receipt, which must always be in the past. This difference, which should be a positive integer, is converted to seconds, and that unsigned value is then used to synthesize the TSR option.

## 6. Timeliness of Conflict Resolution

If a conflict exists in which some mDNS registrant(s) store stale data, it is expected that the conflict is recent, and that it will be resolved quickly since mDNS message exchanges with TSR options cause removal of the stale data. Different hosts may be able to record shorter or longer time offsets, depending on their implementation. However, because of this expectation of recentness, mDNS registrars should never need to report a TSR 'Time Offset' value of longer than seven days. It' s reasonable to expect that every mDNS implementation should be able to remember time intervals of at least seven days.

This maximum time interval is normatively defined in Section 2.

## 7. Legacy Behavior

mDNS registrars and queriers that do not support the TSR option are expected to ignore the option, so they will behave as if no TSR option was sent. This may result in such registrars temporarily caching stale data. However, in the normal course of processing, more recent data will win. In cases where it does not, the Reconfirm process which is part of [RFC6762] already works to clear stale data: since SRP registrars are expected to implement TSR, by the time a Reconfirm is attempted, all authoritative stale data should have been cleared.

## 8. When to Use TSR

There are no cases where using TSR is harmful, but in the case of individual devices advertising individual services with mDNS, it may be of little benefit. The reason for this is that when two devices both claim the same name to use for advertising services, their advertisements will be seen as in conflict whether or not TSR is present. In this case, TSR does no harm, but most likely does not help.

### 8.1. Use of TSR with redundant proxies

Use of TSR is most beneficial in the case of redundant mDNS proxies. The reason for this is that such proxies tend to publish data that could potentially produce name conflicts as a result of updates: when one proxy is publishing an old version of the data and another proxy is publishing a new version of the data, and these data differ, this can appear as a name conflict and result in renaming. So for this use case, the use of TSR is very beneficial.

### 8.2. Use of TSR with multihomed devices

It can also happen that a multihomed device uses SRP to register when connected to one sort of network, and uses mDNS to advertise the same service when connected to another sort of network. For example, the device may have capabilities to connect to a constrained network to reduce power use, but also to occasionally connect to a WiFi network either for backup or for bulk data transfers. In this situation we have two potentially competing mDNS advertisements. The first is that of the mDNS registrar on the multihomed device, which is directly advertising its own service using mDNS. The second is the one or more advertising proxies that are advertising services registered via SRP using mDNS. Both datasets are coming from the same source, and the advertising proxy will use TSR to identify the source. To avoid spurious conflicts, a device that sometimes registers with SRP and sometimes advertises with mDNS SHOULD use TSR when advertising with mDNS.

### 8.3. TSR in networks with non-compliant mDNS caches

Some network infrastructure devices available commercially provide "mDNS cache" services or "mDNS proxy" services that purport to allow mDNS discovery across links that are separate at the IP layer and do not share a multicast domain. These services generally cause mDNS to become unreliable in various ways, and it would be helpful to be able to use TSR to distinguish between stale data advertised by such devices, and fresh data advertised directly from the device that is authoritative for that data.

Unfortunately, for the most part these devices do not cache mDNS packets, but rather cache data advertised using mDNS. As a result, if such devices do not support TSR, we get no benefit from TSR.

That said, such devices SHOULD support TSR, and if they do this might prevent some of their failure modes. What this would mean would be that any DNS RRs cached or forwarded by such devices would be accompanied by any TSR data applicable to them. When forwarded with no time delay, the TSR data could be copied verbatim (following the rules for generating packets with TSR given in Section 3.9).

When such RRs are cached and later resent, they would also be accompanied by their TSR data. The mDNS cache service would need to record the time at which they were received. When retransmitting such cached data, the cache service would need to adjust the time offset in the TSR option, increasing it by the subtracting the time at which the cached RRs were received from the current time, and then increasing the offset in the applicable TSR option by that amount, up to the limit of 7 days.

The use of such caches and forwarders is NOT RECOMMENDED. We mention the use of TSR for such use cases because it can mitigate some of the failure modes of such caches and forwarders. However, the use of Discovery Proxy [RFC8766] in such instances is preferred: as the size of the network covered by such caching and forwarding services increases, the amount of mDNS traffic they create increases, and this is often addressed by limiting which services are available for discovery. Discovery Proxy uses unicast DNS, and therefore does not suffer from these limitations.

## 9. Registrant API considerations

When a registrant registers a service at its mDNS registrar and requests the use of a time of receipt, the registrant MUST also specify when it received the original registration. In order to support this, the API is required not only to allow the registrant to specify that TSR conflict resolution is wanted, but also to provide a way for the registrant to specify an absolute time at which the original registration was received, and the key checksum used to identify the entity that’s actually authoritative for the data.

This is important, for example, in the case of SRP Replication [I-D.ietf-dnssd-srp-replication], where an SRP registrar may receive a registration from a peer during startup synchronization. This registration will have occurred at some significant amount of time in the past, and so it would be incorrect for the mDNS registrar receiving the registration to use the time that the registrant registers the service as the time of receipt.

### 9.1. Removing data that is still valid

In some cases, a proxy may need to stop being a proxy, but may be proxying RRs that are also being proxied by one or more other proxies. In this case, if the proxy sends a "goodbye" announcement for such RRs, they will be removed from the caches of mDNS registrars that receive such announcements.

To prevent this, an mDNS registrar implementation that implements TSR MUST provide a way for an mDNS registrant to indicate that such data is being withdrawn from publication by that registrant, but is still valid. When the registrant indicates that this is the case, the mDNS registrar MUST NOT send goodbye announcements for such data.

### 9.2. Primary/Secondary indication

When more than one proxy is authoritative for a particular RR, this can generate excessive answer traffic, and also redundant goodbye announcements. mDNS registrar implementations that support TSR MUST provide a way for a new proxy mDNS registrant to indicate that it is primary or secondary. When an RR registered by a secondary proxy is later removed, the mDNS registrant MUST NOT send a goodbye packet for that RR.

Similarly, when an RR is registered by a registrant that indicates that it is secondary, the mDNS registrar MUST NOT respond to the initial mDNS query for that RR. Only if a second mDNS query retransmission is received within 5 seconds of the initial query reception, should it respond.

Note that any change of indication from primary to secondary, or vice versa, while an mDNS registrant already has records registered on the mDNS registrar is out of scope of this specification and an API to indicate such changes is not required on the mDNS registrar.

## 10. Security Considerations

The TSR option is an optimization: it ameliorates an edge case for mDNS proxies. A malicious host on the same link could use the TSR option to win conflict resolution processes. However, because TSR is only used by proxies, this technique will not work for normal mDNS service registrations: in that case, normal mDNS conflict resolution is done, and the attacker gains no benefit from using TSR.

Whether or not an mDNS registration has a recorded time of receipt, an attacker can deny service by announcing its own conflicting data and then answering the subsequent probe as described in Section 9 of [RFC6762]. Because it does not include a TSR option in its authority section, it can win the simultaneous conflict resolution process that follows its bogus announcement.

So the TSR-based conflict resolution process creates no new vulnerability. Addressing the existing vulnerability is out of scope for this document. Protocols that rely on mDNS MUST NOT assume that mDNS service is secure or private. If security (authentication, authorization and/or secrecy) are needed, these must be provided at the application layer, or by using DNSSEC rather than mDNS for service discovery.

## 11. IANA Considerations

IANA is requested to allocate a new OPT RR option code from the DNS EDNS0 Option Codes (OPT) registry for the ’Time Since Received’ Option. The Name shall be ’mDNS-TSR’. The value shall be allocated by IANA. The meaning shall be ’Multicast DNS Time Since Received". Reference shall refer to this document, once published. IANA shall determine the registration date.

## 12. References

### 12.1. Normative References

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

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

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

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

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/rfc/rfc8174>.

[RFC8766] Cheshire, S., "Discovery Proxy for Multicast DNS-Based Service Discovery", RFC 8766, DOI 10.17487/RFC8766, June 2020, <https://www.rfc-editor.org/rfc/rfc8766>.

[RFC9665] Lemon, T. and S. Cheshire, "Service Registration Protocol for DNS-Based Service Discovery", RFC 9665, DOI 10.17487/RFC9665, June 2025, <https://www.rfc-editor.org/rfc/rfc9665>.

### 12.2. Informative References

[I-D.ietf-dnssd-advertising-proxy]
Cheshire, S. and T. Lemon, "Advertising Proxy for DNS-SD Service Registration Protocol", Work in Progress, Internet-Draft, draft-ietf-dnssd-advertising-proxy-04, 4 March 2024, <https://datatracker.ietf.org/doc/html/draft-ietf-dnssd-advertising-proxy-04>.

[I-D.ietf-dnssd-srp-replication]
Lemon, T., Keshavarzian, A., and J. Hui, "Automatic Replication of DNS-SD Service Registration Protocol Zones", Work in Progress, Internet-Draft, draft-ietf-dnssd-srp-replication-02, 4 March 2024, <https://datatracker.ietf.org/doc/html/draft-ietf-dnssd-srp-replication-02>.

## Appendix A. Duplicate Answer Suppression Example

As described in Section 3.6, it is possible that two proxies may respond to an mDNS query with answers to the same question, where the data for a particular owner name in both answers is authoritative and unique, but where the TSR time in the earlier message is earlier than the TSR time in the later message. In this case, we will see the data from the earlier message added and then removed, followed by the data in the later message being added. Consider the following example using Apple’s mDNSResponder implementation.

1. mDNS requester A sends a multicast query for AAAA records on the name "example.local"

2. mDNS registrar for proxy B sends a multicast response of 2001:db8:0:42::1 with TSR time T

3. mDNS registrar for proxy C sends a multicast response of 2001:db8:0:17::1 with TSR time T+300

Note that TSR times are absolute times, but these are represented in the mDNS message as relative times, so for example "TSR time T+300" when sent at time T+400 would be represented in the mDNS message as a time offset of 100 seconds, and if the message in (2) were sent at nearly the same time, it would have a time offset of about 400 seconds.

In between (2) and (3) we would expect mDNS Requester A to see a callback to its DNSServiceQueryRecord call, providing a AAAA record of 2001:db8:0:42::1 with the kDNSServiceFlagsAdd bit set and the kDNSServiceFlagsMoreComing bit clear.

After (3) we would expect mDNS Requester A to see two callbacks to its DNSServiceQueryRecord call. The first would provide a AAAA record of 2001:db8:0:42::1 with the kDNSServiceFlagsAdd clear and the kDNSServiceFlagsMoreComing bit set. The second would provide a AAAA record of 2001:db8:0:17::1 with the kDNSServiceFlagsAdd bit set and the kDNSServiceFlagsMoreComing bit clear.

Note that we would not normally see the reverse sequence:

1. mDNS requester A sends a multicast query for AAAA records on the name "example.local"

2. mDNS registrar for proxy C sends a multicast response of 2001:db8:0:17::1 with TSR time T+300

3. mDNS registrar for proxy B sends a multicast response of 2001:db8:0:42::1 with TSR time T

This is because in this case we would expect known answer suppression to result in the mDNS registrar for proxy B suppressing its response. However, it is possible that B might not see the response from C. In that situation, A would receive both responses, but since the TSR time on the second response is earlier than the TSR time on the first response, we would see only one callback, providing an AAAA record of 2001:db8:0:17::1 with the kDNSServiceFlagsAdd bit set and the kDNSServiceFlagsMoreComing bit clear.

## Acknowledgments

TODO acknowledge reviewers and contributors.

## Authors' Addresses

Ted Lemon
Apple Inc.
One Apple Park Way
Cupertino, California 95014
United States of America
Email: mellon@fugue.com

Esko Dijk
IoTconsultancy.nl
Utrecht
Netherlands
Email: esko.dijk@iotconsultancy.nl