

# SRP Compression for Constrained Networks

**IEFT 121**

Extensions for Scalable DNS Service Discovery (dnssd) WG

Abtin Keshavarzian

# Quick Intro - Abtin Keshavarzian

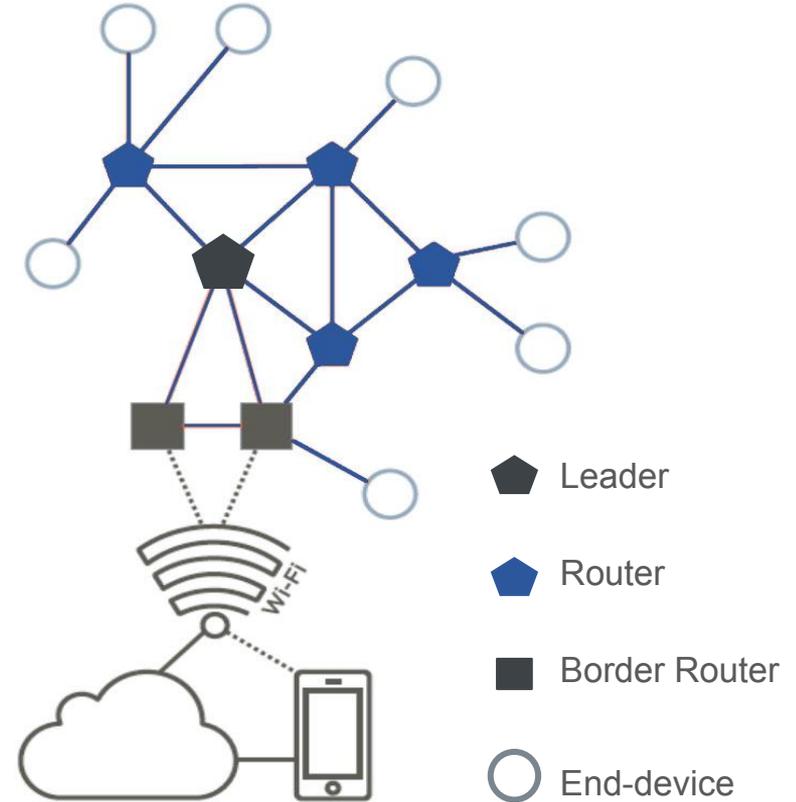
- Background
  - PhD from Stanford in EE
  - ~25 years of experience in research and development
    - Networking
    - Wireless
    - Embedded SW
- Work at Google (abtink@)
  - Thread protocol
  - OpenThread (Open source implementation Thread)

# Background

- SRP - Service Registration Protocol
  - DNS-based Service Discovery using standard DNS Update with unicast packets
- Used by Thread
  - Thread devices use SRP to register services
  - Thread Border Routers act as SRP server
- Matter: Application layer protocol over Thread (and Wi-Fi)
  - Defines registered service names
  - Instance name, sub-types, host name, TXT data format, etc.

# Thread

- Low-power Secure Mesh Network
- IPv6
  - 6LoWPAN
  - CoAP
- PHY: IEEE 802.15.4 (2.4GHz)
  - 250 Kbps data rate
  - PHY layer frame **127 bytes**



# SRP Message Example (matter device)

## Two services and one host address

8F097FD118441046-00000000B3B3D017

type: **\_matter.\_tcp**

sub-type: **\_I8F097FD118441046**

port:5540, w:0, p:0, TXT data: 31 byte

1FF04909193C16E2-000000006CC07561

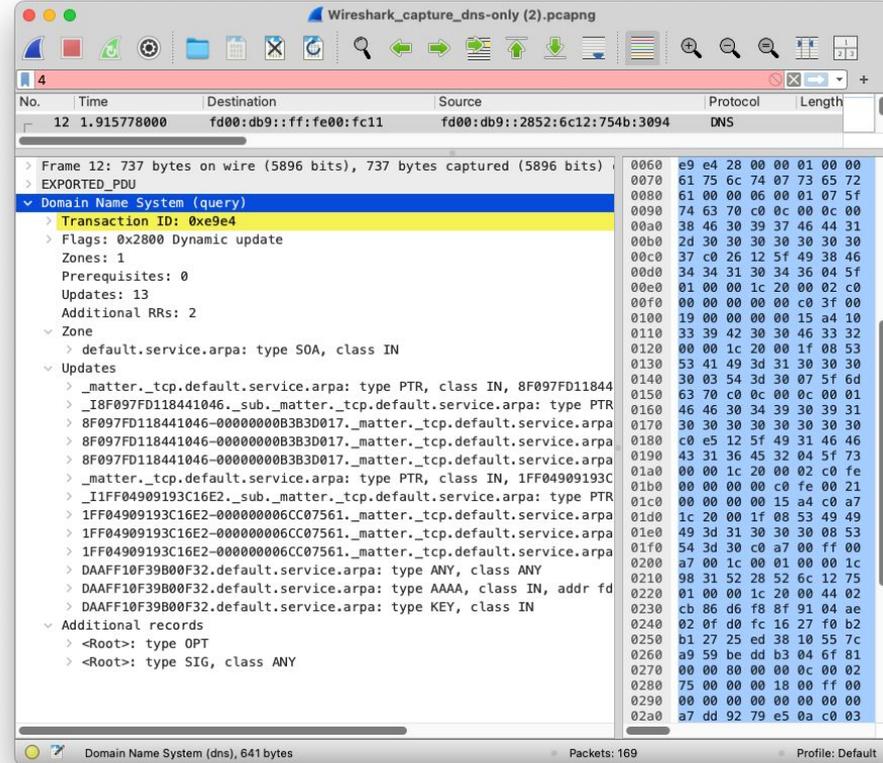
type: **\_matter.\_tcp**

sub-type: **\_I1FF04909193C16E2**

port:5540, w:0, p:0, TXT data: 31 bytes

Host: **DAAFF10F39B00F32**

fdc5:d08a:4e98:3152:2852:6c12:754b:3094

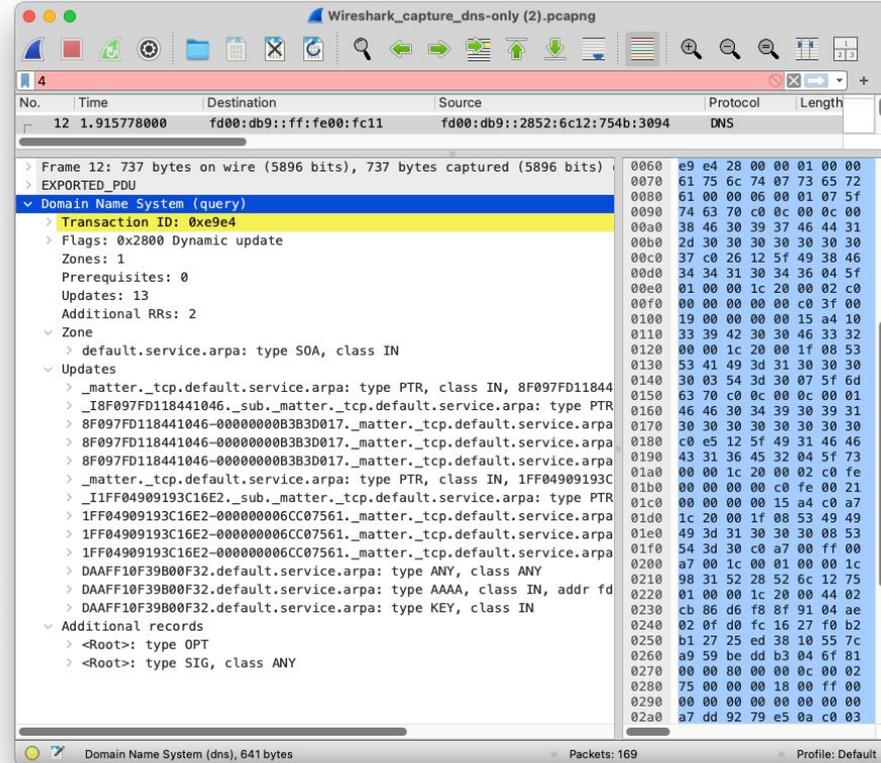


# SRP Message Example (matter device)

641 bytes (UDP payload)

15 Records (13 in update, 2 in addtl)

|                         |     |       |
|-------------------------|-----|-------|
| DNS Header              | 12  | bytes |
| Zone                    | 26  | bytes |
| 1st service (5 records) | 191 | bytes |
| 2nd service (5 records) | 174 | bytes |
| Host (2 records)        | 40  | bytes |
| Key (1 record)          | 80  | bytes |
| Lease Option (1 record) | 23  | bytes |
| Signature (1 record)    | 95  | bytes |



# SRP Message Example (matter device)

641 bytes (UDP payload)

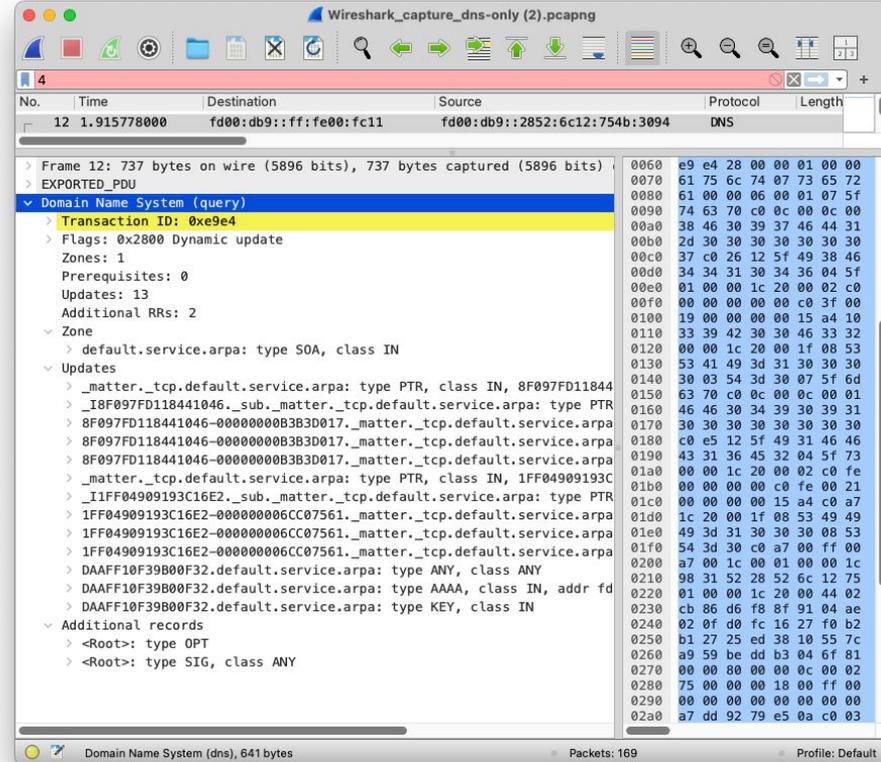
SRP message already uses:

## DNS Name Compression

(use pointer labels to refer to previously encoded names in the message)

## KEY Record Optimization

(include KEY record for host name only, omitting it in service descriptions).



# Typical SRP Update Message Size

- SRP's adherence to the DNS Update general format introduces overhead and can lead to larger SRP message sizes
- Devices may register more services and/or addresses increasing msg size
  - Matter multi-fabric (separate service per fabric/ecosystem)
- Typical SRP message size: **400-1200 bytes**
- Can lead to increased traffic over constrained networks
  - Registration by 10s or 100s of devices
  - Triggered by synchronous events (e.g. Border Router restart)

**Can we compress the SRP message?**

# SRP Coder

- On the client, encode an SRP message into a **compact, compressed format**, reducing the message size
- On the server, the received coded message is decoded to reconstruct an **exact replica of the original message**

Files

49357ff

Go to file

- .github
- doc
- etc
- examples
- include
- script
- src
- tests
- third\_party
- tools
- zephyr
  - .clang-format
  - .clang-tidy
  - .code-spell-ignore
  - .codecov.yml
  - .default-version
  - .gitattributes
  - .gitignore
  - .gn
  - .prettierrc
  - AUTHORS
  - BUILD.gn
  - CMakeLists.txt
  - CODE\_OF\_CONDUCT...
  - CONTRIBUTING.md
  - LICENSE
  - NOTICE
  - README.md
  - SECURITY.md
  - STYLE\_GUIDE.md

abtnk [srp] introduce Srp: Coder

49357ff · 3 months ago

Preview Code Blame 309 Lines (227 Loc) · 14.8 KB

## SRP Coder

This document specifies SRP coder used to encode an SRP message into a compact, compressed format, reducing the message size. On the server, the received coded message can be decoded to reconstruct an exact replica of the original message.

The SRP protocol uses DNS resource record format to convey information about registered services and host addresses. The use of these records and adherence to the DNS Update general format introduces overhead and can lead to larger SRP message sizes, which is undesirable for Thread network. The SRP coder aims to optimize the message, resulting in much smaller messages containing all the same information reducing overall protocol overhead.

### Compact Integer Format

- Works with unsigned integer of different length ( `uint16`, `uint32` )
- Number is encoded as one or more segment(s).
- Segments are one byte (8-bit) long except for the first segment which may have fewer bits (2-7 bits).
- First (MSB) bit in each segment is the "continuation bit" indicating whether there are more segments to follow ( `1` ) or if it is the last segment ( `0` ).
- The remaining bits after the MSB provide the numerical `uint` bit values in big-endian order.

### Examples

- Numbers less than ( $2^7$ ) 128 are encoded as a single byte.
- Numbers between 128 and ( $2^{14} - 1$ ) 16,383 are encoded as two bytes.
- Numbers between 16,384 up to ( $2^{21} - 1$ ) 2,097,151 are encoded as three bytes.

### DNS Name and Label Format

#### Label Dispatch

|                |   |   |   |   |   |   |   |
|----------------|---|---|---|---|---|---|---|
| 0              | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Type ...   ... |   |   |   |   |   |   |   |

- `00 <6-bit len>` : Standard label encoding, where the lower 6 bits indicate the label's length (number of characters). This is followed by the label characters, totaling `len` in number.
- `01 <6-bit len>` : Similar to the above, but the label begins with an underscore `'_'`. The `len` value here doesn't include the underscore itself, which isn't explicitly encoded after the dispatch byte.
- `10 <6-bit offset first seg>` : This references a previously encoded label within the message. The 6-bit value in the dispatch byte serves as the first segment of a compact `uint`

# SRP Coder Example

Matter example: 2 services

Original SRP message

641 bytes



Encoded message

244 bytes

2.4x smaller

Wireshark capture showing the original SRP message (641 bytes) for transaction ID 0xe9e4. The message contains two services: default.service.arpa: type SOA, class IN and two PTR records for the services 1FF04909193C16E2 and 1FF04909193C16E2.

Terminal window showing the decoded SRP message (244 bytes). The output shows the decoded message structure, including the transaction ID 0xe9e4 and the list of services being queried, matching the original message but in a more compact binary format.

# SRP Coder Example

Matter example: 2 services

Original SRP message

**641 bytes**

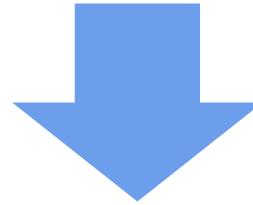


Encoded message

**244 bytes**

**2.4x smaller**

**= 175 bytes (KEY/SIG) + 466 bytes**



**= 128 bytes (KEY/SIG) + 116 bytes**

**4x smaller**

- To better highlight efficiency, it might make sense to exclude Key and Signature records, as they are not compressible.
- Recommend future extensions to SRP to allow shorter key and signature sizes.

# SRP Coded Format

## Message Format

Header Block

*Zero or more*

Add/Remove Service Block(s)

Host Block

Footer Block

Each block begins with a “dispatch byte”

- Indicating the block type
- Bit flags to show which fields are elided
- Followed by encoded fields in the block

## General Dispatch Byte Markers

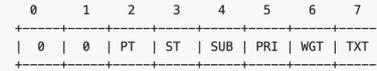
Following the header, each subsequent block begins with a dispatch byte. The initial bits within a dispatch byte designate its type:

- `00` : Add service
- `01` : Remove service
- `10` : Host
- `110` : Footer

Any unused/reserved bits in dispatch bytes MUST be set to zero during encoding and its value MUST be ignored during decoding.

## Add Service Block

### Add Service Dispatch



- **PT** flag (PTR TTL) - `0` : Elide PTR TTL (use default TTL from header), `1` : TTL encoded.
- **ST** flag (SRV/TXT TTL) - `0` : Elide SRV/TXT TTL (use default TTL from header), `1` : TTL encoded.
- **SUB** flag (sub-type) - `0` : No sub-types, `1` : Sub-types labels encoded.
- **PRI** flag (priority) - `0` : Elide priority (use zero), `1` : Priority encoded.
- **WGT** flag (weight) - `0` : Elide weight (use zero), `1` : Weight encoded.
- **TXT** flag (TXT data) - `0` : No TXT data (elided), `1` : TXT data encoded.

### Add Service Block Format

- "Add Service Dispatch" byte
- PTR TTL (if not elided)
- SRV/TXT TTL (if not elided)
- Service instance label
- Service name labels (excluding the domain name).
- SubType labels (if not elided) - A list of labels, terminated by an empty label (single `0x00` byte).
- Port
- Priority (if not elided)
- Weight (if not elided)
- TXT Data block - if not elided.

TTLs, port, priority, and weight fields (when not elided) are encoded using the compact integer format.

# SRP Coder - Encoding Labels

Can refer to a previously encoded label in the msg

Commonly used labels encoded as a single byte

`_udp`, `_tcp`, `_matter`, ...

Generative patterns are encoded more efficiently

## Host name

`DAAFF10F39B00F32`

16-char ASCII hex representation of a 64-bit value

## Service Instance

`8F097FD118441046-00000000B3B3D017`

“<16-char ASCII hex>-<16-char ASCII hex>”

(two 64-bit hex values separated by hyphen)

## DNS Name and Label Format

### Label Dispatch



- **00** <6-bit len> : Standard label encoding, where the lower 6 bits indicate the label's length (number of characters). This is followed by the label characters, totaling `len` in number.
- **01** <6-bit len> : Similar to the above, but the label begins with an underscore `'_'`. The `len` value here doesn't include the underscore itself, which isn't explicitly encoded after the dispatch byte.
- **10** <6-bit offset first seg> : This references a previously encoded label within the message. The 6-bit value in the dispatch byte serves as the first segment of a compact `uint` representing the offset to that label. This offset must point to the label dispatch byte of the previous label.
  - Note: This mechanism applies to a single label only. Any subsequent labels are encoded directly after the current one. This differs from traditional DNS pointer name compression, where a pointer indicates that the remaining labels should be read from the specified offset.
- **110** <5-bit code> : Commonly used constant labels. Codes are:
  - **0**: `_udp`
  - **1**: `_tcp`
  - **2**: `_matter`
  - **3**: `_matterc`
  - **4**: `_matterd`
  - **5**: `_hap`
- **111** <5-bit code> : Commonly Used Generative Patterns
  - **0**: `<hex_value>` - 16-character uppercase hexadecimal 64-bit value.
    - Example: `DAAFF10F39B00F32`
    - Encoding: After the label dispatch byte, the 64-bit value is encoded as 8-byte binary format (big-endian order).
  - **1**: `<hex_value_1>-<hex_value_2>` - Two 16-character uppercase hexadecimals, separated by a hyphen `-`.
    - Example `AABBCCDDEEFF0011-1122334455667788` .
    - Encoding: After the label dispatch byte, the two 64-bit values are encoded (each as 8-byte binary format big-endian).
  - **2**: Subtype label `_<char><hex_value>` - starting with an underscore `_`, followed by a single character `<char>`, ending with a 16-character uppercase hexadecimal 64-bit value `<hex_value>`.
    - Example: `_IAA557733CC00EE11`
    - Encoding: After the label dispatch byte, `<char>` is encoded as a single byte, followed by the 8-byte big-endian binary representation of the 64-bit value `<hex_value>` .
  - **3**: Subtype Label - same as the previous case.
    - Encoding: Instead of directly encoding the 8-byte value, an offset pointing to an earlier occurrence of the same byte sequence within the message is encoded. The offset is

# OpenThread SRP Coder

## Implementation of SRP Coder (encoder/decoder)

- Integrated with OT SRP client and server
- Standalone decoder (use with external server impl)
- Documentation of the protocol
- Extensive set of tests

<https://github.com/openthread/openthread/pull/10606>

## Next Steps

- Adopt the SRP Coder in the Thread specification (deploy and test)
- Open to a future IETF draft (assuming interest)

[srp] introduce Srp::Coder #10606

Open abtink wants to merge 1 commit into openthread:main from abtink:srp/coder

Conversation 24 Commits 1 Checks 103 Files changed 40

abtink commented on Aug 13 • edited Member

This commit introduces `Srp::Coder`, to encode an SRP message into a compact, compressed format, reducing the message size. On the server, the received coded message is decoded to reconstruct an exact replica of the original message.

The SRP protocol uses DNS resource record format to convey information about registered services and host addresses. The use of these records and adherence to the DNS Update general format introduces overhead and can lead to larger SRP message sizes, which is undesirable for Thread network. The `Srp::Coder` aims to optimize the message, resulting in much smaller messages containing all the same information reducing overall protocol overhead.

This commit updates both SRP client and server implementations in OpenThread to utilize the SRP coder. New APIs are added to control the behavior of the SRP client (whether it should prepare and send coded messages or not). Additionally, public APIs are added to enable external SRP server implementations to utilize the `Srp::Coder` for decoding received coded SRP messages.

This commit also includes extensive tests covering the behavior of the newly added `Srp::Coder`:

- A new unit test, `test_srp_coder`, is added to validate specific functions/methods of `Srp::Coder`.
- `test_srp_server` and `test_srp_adv_proxy` tests are updated to run all test cases twice: with and without the use of the SRP coder.
- Similarly relevant `tests/scripts/thread-cert/test_srp_*` tests are updated to run twice, with and without the coder on the client.
- New specific test cases are added in `test_srp_server` to validate the format of encoded messages, covering various edge cases.

Let's consider the example from [SPEC-1281](#) representing a typical Matter device that's part of two fabrics

Backup Slides

- > Flags: 0x2800 Dynamic update
- Zones: 1
- Prerequisites: 0
- Updates: 13
- Additional RRs: 2
- ▼ Zone
  - > default.service.arpa: type SOA, class IN
- ▼ Updates
  - > \_matter.\_tcp.default.service.arpa: type PTR, class IN, 8F097FD118441046-00000000B3B3D017.\_matter.\_tcp.defa
  - > \_I8F097FD118441046.\_sub.\_matter.\_tcp.default.service.arpa: type PTR, class IN, 8F097FD118441046-00000000B3
  - > 8F097FD118441046-00000000B3B3D017.\_matter.\_tcp.default.service.arpa: type ANY, class ANY
  - > 8F097FD118441046-00000000B3B3D017.\_matter.\_tcp.default.service.arpa: type SRV, class IN, priority 0, weigh
  - > 8F097FD118441046-00000000B3B3D017.\_matter.\_tcp.default.service.arpa: type TXT, class IN
  - > \_matter.\_tcp.default.service.arpa: type PTR, class IN, 1FF04909193C16E2-000000006CC07561.\_matter.\_tcp.defa
  - > \_I1FF04909193C16E2.\_sub.\_matter.\_tcp.default.service.arpa: type PTR, class IN, 1FF04909193C16E2-000000006C
  - > 1FF04909193C16E2-000000006CC07561.\_matter.\_tcp.default.service.arpa: type ANY, class ANY
  - > 1FF04909193C16E2-000000006CC07561.\_matter.\_tcp.default.service.arpa: type SRV, class IN, priority 0, weigh
  - > 1FF04909193C16E2-000000006CC07561.\_matter.\_tcp.default.service.arpa: type TXT, class IN
  - > DAAFF10F39B00F32.default.service.arpa: type ANY, class ANY
  - > DAAFF10F39B00F32.default.service.arpa: type AAAA, class IN, addr fdc5:d08a:4e98:3152:2852:6c12:754b:3094
  - > DAAFF10F39B00F32.default.service.arpa: type KEY, class IN
- ▼ Additional records
  - > <Root>: type OPT
  - > <Root>: type SIG, class ANY

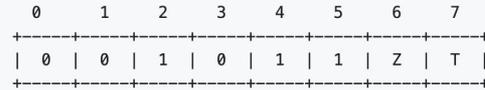
# Distinguishing Coded Msg

The first 3 bytes (Header Block) can be used to differentiate a coded message from an SRP Update message:

- First two bytes encode message ID (same as general DNS header)
- Coded message uses fixed pattern as the 3rd byte to distinguish itself from an SRP update message

## Header Block

### Header Dispatch



- The initial bits in dispatch byte are fixed to differentiate it from the general DNS header used in SRP updates.
- **Z** flag (Zone)
  - 0 : Zone name is elided - use the default `default.service.arpa`.
  - 1 : Zone name is encoded within the message.
- **T** flag (Default TTL)
  - 0 : Default TTL is elided - use `7200` (2 hours).
  - 1 : Default TTL is encoded in the message using compact integer format.

### Distinguishing from SRP Update Message

An SRP Update message uses the following DNS Update header format (RFC2136 section 2.2):



The first 3 bytes can be used to differentiate a coded message from an SRP Update message. In the coded message, the message ID is encoded in the first two bytes, similar to a general DNS header. The fixed initial bit pattern in the header dispatch byte, when mapped to the DNS header, corresponds to `QR = 0`, `Opcode = 0b0101 = 5`, with bit 5 also set. An SRP Update message also employs `Opcode = 5` but with zero flags (`Z`). Setting bit 5 to `1` is used to identify a coded message.