CBOR
Internet-Draft
Intended status: Standards Track
Expires: 8 August 2026

M. S. Lenders
TU Dresden
C. Bormann
Universit辰t Bremen TZI
M. G端tschow
TU Dresden
T. C. Schmidt
HAW Hamburg
M. W辰hlisch
TU Dresden & Barkhausen Institut

4 February 2026

A Concise Binary Object Representation (CBOR) of DNS Messages
draft-lenders-dns-cbor-16

Abstract

This document specifies a compact data format of DNS messages using the Concise Binary Object Representation [RFC8949]. The primary purpose is to keep DNS messages small in constrained networks.

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://cbor-wg.github.io/cbor-dns/draft-lenders-dns-cbor.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-lenders-dns-cbor/.

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

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

Status of This Memo

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

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

Copyright Notice

Table of Contents

1. Introduction

In constrained networks [RFC7228], the link layer may restrict the payload sizes of frames to only a few hundreds bytes. Encrypted DNS resolution, such as DNS over HTTPS (DoH) [RFC8484] or DNS over CoAP (DoC) [I-D.ietf-core-dns-over-coap], may lead to DNS message sizes that exceed this limit, even when implementing header compression such as 6LoWPAN IPHC [RFC6282] or SCHC [RFC8724], [RFC8824].

Although adoption layers such as 6LoWPAN [RFC4944] or SCHC [RFC8724] offer fragmentation to comply with small MTUs, fragmentation should be avoided in constrained networks. Fragmentation combined with high packet loss multiplies the likelihood of loss. Hence, a compression format that reduces fragmentation of DNS messages is beneficial.

This document specifies a compact data format for DNS messages using Concise Binary Object Representation (CBOR) [RFC8949] encoding. Additionally, unnecessary or redundant information are stripped off DNS messages. To use the outcome of this specification in DoH and DoC, this document also specifies a Media Type header for DoH and a Content-Format option for DoC.

Note, that there is another format that expresses DNS messages in CBOR, C-DNS [RFC8618]. C-DNS is primarily a file format to minimize traces of multiple DNS messages and uses the fact that there are multiple messages to do its compression. Common values such as names or addresses are collected in separate tables which are referenced from the messages, comparable to Packed CBOR [I-D.ietf-cbor-packed]. However, this may add overhead for individual DNS messages.

The format described in this document is a transfer format that aims to provide conciseness and compression for individual DNS messages to be sent over the network. This is achieved applying the following objectives:

Conciseness:

* Encoding DNS messages in CBOR and

* Omitting (redundant) fields in DNS queries and responses.

Compression:

* Providing easy to implement name compression that allows for on-the-fly construction of DNS queries and responses and

* Providing optional address and value compression in DNS responses using Packed CBOR [I-D.ietf-cbor-packed].

2. Terminology and Conventions

CBOR types (unsigned integer, byte string, text string, arrays, etc.) are used as defined in [RFC8949].

The terms "DNS server", "DNS client", and "(DNS) resolver" are used as defined in [RFC8499].

A DNS query is a message that queries DNS information from an upstream DNS resolver. The reply to that is a DNS response.

The DNS message format specified in [RFC1035] for DNS over UDP we call "classic DNS format" throughout this document or refer to it by its media type "application/dns-message" as specified in [RFC8484].

The term "constrained networks" is used as defined in [RFC7228].

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.

To define the representation of binary objects we use the Concise Data Definition Language (CDDL) [RFC8610]. For examples, we use the CBOR Extended Diagnostic Notation [I-D.ietf-cbor-edn-literals] with the e'' application extension [I-D.ietf-cbor-edn-e-ref].

3. CBOR Representations (application/dns+cbor)

DNS messages in "application/dns+cbor" are represented as CBOR arrays to minimize overhead. All CBOR items used in this specification are of definite length. CBOR arrays that do not follow the length definitions of this or of follow-up specifications, MUST be silently ignored. CBOR arrays that exceed the message size provided by the transport, MUST be silently ignored. It is assumed that DNS query and DNS response are distinguished message types and that the query can be mapped to the response by the transfer protocol of choice.

```
dns-message = dns-query / dns-response
```

Figure 1: This document defines both DNS Queries and Responses in CDDL

If, for any reason, a DNS message cannot be represented in the CBOR format specified in this document, or if unreasonable overhead is introduced, a fallback to another DNS message format, e.g., the classic DNS format specified in [RFC1035], MUST always be possible.

3.1. Domain Name Representation

Domain names are represented by a sequence of one or more (unicode) text strings. For instance, "example.org" would be represented as "example","org" in CBOR diagnostic notation. The root domain "." is represented as an empty string "". The absence of any label means the name is elided. For the purpose of this document, domain names remain case-insensitive as specified in [RFC1035].

The representation of a domain name is defined in Figure 2. A label may either be encoded in ASCII-compatible encoding (ACE) [RFC5891] embedded within UTF-8 encoding of the text strings or plain UTF-8. It is RECOMMENDED to use the encoding with the shorter length in bytes, otherwise message sizes may increase. A decoder identifies

the ACE encoding by identifying the label as a valid A-label (see [RFC5891]) and MUST assume the label to be encoded in UTF-8 otherwise.

This sequence of text strings is supposed to be embedded into a surrounding array, usually the query or resource record.

Name compression is implemented using an extension to Packed CBOR, see Section 4.1. For readers unfamiliar with Packed CBOR this name compression can be abstracted to a name compression similar to that described in Section 4.1.4 of [RFC1035]. However, instead of using the byte index as reference within the message, text strings are counted, starting at 0, depth-first within the message. That number is used as index for the reference. Since names are the only text strings in a CBOR-based DNS message, the end of a name can be identified when the decoder cursor does not point to a text string or reference to another text string anymore. For the reference itself, either simple values or tag 6 are used (see Section 2.2 of [I-D.ietf-cbor-packed]).

```
domain-name = ( *label )
label = tstr
```

Figure 2: Domain Name Definition

3.2. DNS Resource Records

This document specifies the representation of both standard DNS resource records (RRs, see [RFC1035]) and EDNS option pseudo-RRs (see [RFC6891]. If for any reason, a resource record cannot be represented in the given formats, they can be represented in their binary wire-format form as a byte string.

Further special records, e.g., TSIG [RFC8945], can be defined in follow-up specifications using the $$structured-ts-rd extension point (see Figure 4) and are out of scope of this document.

The representation of a DNS resource records is defined in Figure 3.

```
$$dns-rr = rr / #6.141(opt-rr) / bstr
```

Figure 3: DNS Resource Record Definition

3.2.1. Standard RRs

Standard DNS resource records are encoded as CBOR arrays containing 2 or more entries in the following order:

1. An optional name (as text string, see Section 3.1),

2. A TTL (as unsigned integer),

3. An optional record type (as unsigned integer),

4. An optional record class (as unsigned integer), and lastly

5. A record data entry (as byte string, domain name, or array for dedicated record data representation) or a boolean (true) that indicates that the resource record is actually a resource record set with an array of one or more record data entries following. In the latter case, individual domain names need to be put into their own array.

If the first item of the resource record is a text string, it is the first label of a domain name (see Section 3.1). If the name is elided, the name is derived from the question section of the message. For responses, the question section is either taken from the query (see Section 3.3) or provided with the response see Section 3.4. The query may be derived from the context of the transfer protocol.

If the record type is elided, the record type from the question is assumed. If record class is elided, the record class from the question is assumed. When a record class is required to be expressed, the record type MUST also be provided.

The byte string format of the record data as a byte string follows the classic DNS format as specified in Section 3.3 of [RFC1035] (or other specifications of the respective record type). Note that the CBOR format does not include the RDLENGTH field from the classic format as this value is encoded in the length field of the CBOR header of the byte string.

If the record data represents a domain name (e.g., for CNAME or PTR records), the record data MAY be represented as domain name as specified in Section 3.1. This way of representing the record data means that name compression (see Section 4.1) can also be used on it.

Depending on the record type, the record data may also be expressed as an array. Some initial array types are specified below. Future specifications can extend the definition for $$structured-ts-rd in Figure 4. Extensions to that in this document mainly serve to expose names to name compression (see Section 4.1). There is an argument to be made for CBOR-structured formats of other record data representations (e.g. DNSKEY or RRSIG), but structuring such records as an array usually adds more overhead than just transferring the byte representation. As such, structured record data that do not contain names are to be represented as a byte string in this specification.

Multiple resource records of the same type, class, and TTL can be summarized to a resource record set. A decoder can be notified about this, by including the boolean true value before an array of multiple record data entries of the same type. Note, that this adds more overhead to the message and should only really be considered, when there are more than one resource records of the same type, class, and TTL in the message.

```
max-uint8 = 0..255
max-uint16 = 0..65535
max-uint32 = 0..4294967295
ttl = max-uint32
ipv4-addr = bstr .size 4
ipv6-addr = bstr .size 16
$ip-addr = ipv4-addr / ipv6-addr

rr = [
  ? domain-name,
  ttl: ttl,
  type-spec-rdata,
]
type-spec-rdata = (
  ? type-spec,
  rdata: bstr // ( $ip-addr ) // ( domain-name ) // ( rdata-set ),
)
rdata-set = ((
  is-rrset: true,
  rdata-set: [ +bstr ]
) // (
  is-rrset: true,
  rdata-set: [ +[ domain-name ] ],
))
type-spec-rdata //= ( $$structured-ts-rd )
type-spec = (
  record-type: max-uint16,
  ? record-class: max-uint16,
)
```

Figure 4: DNS Standard Resource Record Definition

3.2.1.1. SOA Record Data

The record data of RRs with record-type = 6 (SOA) MAY be expressed as an array with at least 7 entries representing the 7 parts of the SOA resource record defined in [RFC1035] in the following order:

* MNAME as a domain name (see Section 3.1),

* SERIAL as an unsigned integer,

* REFRESH as an unsigned integer,

* RETRY as an unsigned integer,

* EXPIRE as an unsigned integer,

* MINIMUM as an unsigned integer, and

* RNAME as a domain name (see Section 3.1).

MNAME and RNAME are put to the beginning and end of the array, respectively, to keep their labels apart.

The definition for SOA record data can be seen in Figure 5.

```
$$structured-ts-rd //= (
  6,    ; record-type = SOA
  ? 1,  ; record-class = IN
  ( soa // ( is-rrset: true, rdata-set: [ +soa ] ) ),
)

soa = [
  domain-name,  ; mname
  serial: max-uint32,
  refresh: max-uint32,
  retry: max-uint32,
  expire: max-uint32,
  minimum: max-uint32,
  domain-name,  ; rname
]
```

Figure 5: SOA Resource Record Data Definition

3.2.1.2. MX Record Data

The record data of RRs with record-type = 15 (MX) MAY be expressed as an array with at least 2 entries representing the 2 parts of the MX resource record defined in [RFC1035] in the following order:

* PREFERENCE as an unsigned integer and

* EXCHANGE as a domain name (see Section 3.1).

The definition for MX record data can be seen in Figure 6.

```
$$structured-ts-rd //= (
  15,   ; record-type = MX
  ? 1,  ; record-class = IN
  ( mx // ( is-rrset: true, rdata-set: [ +mx ] ) ),
)

mx = [
  preference: max-uint16,
  domain-name,  ; exchange
]
```

Figure 6: MX Resource Record Data Definition

3.2.1.3. SRV Record Data

The record data of RRs with record-type = 33 (SRV) MAY be expressed as an array with at least 3 entries representing the parts of the SRV resource record defined in [RFC2782] in the following order:

* Priority as an unsigned integer,

* an optional Weight as an unsigned integer,

* Port as an unsigned integer,

* Target as a domain name (see Section 3.1).

If the weight is present or not can be determined by the number of unsigned integers before Target. 2 unsigned integers before the Target mean the weight was elided and defaults to 0. 3 unsigned integers before the Target mean the weight is in the second position of the record data array. The default of 0 was picked, as this is the value domain administrators should pick when there is no server selection to do [RFC2782].

The definition for SRV record data can be seen in Figure 7.

```
$$structured-ts-rd //= (
  33,   ; record-type = SRV
  ? 1,  ; record-class = IN
  ( srv // ( is-rrset: true, rdata-set: [ +srv ] ) ),
)

srv = [
  priority: max-uint16,
  ? weight: max-uint16 .default 0,
  port: max-uint16,
  domain-name,  ; target
]
```

Figure 7: SRV Resource Record Data Definition

3.2.1.4. SVCB and HTTPS Record Data

The record data of RRs with record-type = 64 (SVCB) and record-type = 65 (HTTPS) MAY be expressed as an array with at least 3 entries representing the 3 parts of the SVCB/HTTPS resource record defined in [RFC9460] in the following order:

* An optional SvcPriority as an unsigned integer,

* An optional TargetName as a domain name (see Section 3.1), and

* SvcParams as an array of alternating pairs of SvcParamKey (as unsigned integer) and SvcParamValue (as byte string). The type of SvcParamValue may be extended in future specifications.

If the SvcPriority is present can be determined by checking if the record data array starts with an unsigned integer or not. If the array does not start with an unsigned integer, the SvcPriority is elided and defaults to 0, i.e., the record is in AliasMode (see Section 2.4.2 of [RFC9460]). If the array starts with a unsigned integer, it is the SvcPriority.

If the TargetName is present can be determined by checking if the record data array has a domain name after the SvcPriority, i.e., if the SvcPriority is elided the array would start with a domain name. If there is no domain name after the SvcPriority, the TargetName is elided and defaults to the sequence of text strings "" (i.e., the root domain "." in the common name representation defined in Section 2.3.1 of [RFC1035], see Section 3.1) and Section 2.5 of [RFC9460]. If there is a domain name after the SvcPriority, the TargetName is not elided and in the domain name form specified in Section 3.1.

The definition for SVCB and HTTPS record data can be seen in Figure 8.

```
$$structured-ts-rd //= (
  64 / 65,  ; record-type = SVCB or HTTPS
  ? 1,      ; record-class = IN
  ( svcb // ( is-rrset: true, rdata-set: [ +svcb ] ) ),
)

svcb = [
  ? svc-priority: max-uint16 .default 0,
  ? domain-name,  ; target name
  svc-params: [ *svc-param-pair ],
]

svc-param-pair = (
  svc-param-key: max-uint16,
  svc-param-value: $$svc-param-value,
)
$$svc-param-value = bstr / $ip-addr
```

Figure 8: SVCB and HTTPS Resource Record Data Definition

The SvcParams are provided as an array rather than a map, as their order needs to be preserved [RFC9460] which can not be guaranteed for maps.

3.2.2. EDNS OPT Pseudo-RRs

EDNS OPT Pseudo-RRs are represented as a CBOR array. To distinguish them from normal standard RRs, they are marked with tag TBD141.

Name and record type can be elided as they are always "." and OPT (41), respectively [RFC6891].

The UDP payload size may be the first element as an unsigned integer in the array. It MUST be elided if its value is the default value of 512, the maximum allowable size for unextended DNS over UDP (see Sections 2.3.4 and 4.2.1 of [RFC1035]).

The next element is an array of the options as alternating pairs of option code (as unsigned integer) and option data (as byte string). The type of the option data may be extended in future specifications. As per Section 6.1.2 of [RFC6891], the order of options is not defined in this specification, but the considerations to order provided in Section 6.1.2 of [RFC6891] also apply here.

After that, up to three unsigned integers are following. The first being the extended flags as unsigned integer (implied to be 0 if elided), the second the extended RCODE as an unsigned integer (implied to be 0 if elided), and the third the EDNS version (implied to be 0 if elided). They are dependent on each of their previous elements. If the EDNS version is not elided, both extended flags and extended RCODE MUST not be elided. If the RCODE is not elided the extended flags MUST not be elided.

Note that future EDNS versions may require a different format than the one described above.

```
opt-rr = [
  ? udp-payload-size: max-uint16 .default 512,
  options: [* opt-pair ],
  ? opt-rcode-v-flags,
]
opt-pair = (
  ocode: max-uint16,
  odata: $$odata,
)
opt-rcode-v-flags = (
  flags: max-uint16 .default 0,
  ? opt-rcode-v,
)
rcode = 0..4095
opt-rcode-v = (
  rcode: rcode .default 0,
  ? version: max-uint8 .default 0,
)
$$odata = bstr
```

Figure 9: DNS OPT Resource Record Definition

3.3. DNS Queries

DNS queries are encoded as CBOR arrays containing up to 6 entries in the following order:

1. An optional boolean field,

2. An optional flag field (as unsigned integer),

3. The question section (as array),

4. An optional answer section (as array),

5. An optional authority section (as array), and

6. An optional additional section (as array)

If the first item is a boolean and when true, it tells the responding resolver that it MUST include the question section in its response. If that boolean is not present, it is assumed to be false.

If the first item of the query is an array, it is the question section, if it is an unsigned integer, it is as flag field and maps to the header flags in [RFC1035] and the "DNS Header Flags" IANA registry including the QR flag and the Opcode.

If the flags are elided, the value 0 is assumed.

This specification assumes that the DNS messages are sent over a transfer protocol that can map the queries to their responses, e.g., DNS over HTTPS [RFC8484] or DNS over CoAP [I-D.ietf-core-dns-over-coap]. As a consequence, the DNS transaction ID is always elided and the value 0 is assumed.

A question record within the question section is encoded as a CBOR array containing the following entries:

1. The queried name (as domain name, see Section 3.1) which MUST not be elided,

2. An optional record type (as unsigned integer), and

3. An optional record class (as unsigned integer)

If the record type is elided, record type AAAA as specified in [RFC3596] is assumed. If the record class is elided, record class IN as specified in [RFC1035] is assumed. When a record class is required, the record type MUST also be provided.

There usually is only one question record [RFC9619], which is why the question section is a flat array and not nested like the other sections. This serves to safe overhead from the additional CBOR array header. In the rare cases when there is more than one question record in the question section, the next question just follows. In this case, for every question but the last, the record type MUST be included, i.e., it is not optional. This way it is ensured that the parser can distinguish each question by looking up the name first.

The remainder of the query is either empty or MUST consist of up to three extra arrays.

If one extra array is in the query, it encodes the additional section of the query as an array of DNS resource records (see Section 3.2). If two extra arrays are in the query, they encode, in that order, the authority and additional sections of the query each as an array of DNS resource records (see Section 3.2). If three extra arrays are in the query, they encode, in that order, the answer section, the authority, and additional sections of the query each as an array of DNS resource records (see Section 3.2).

As such, the highest precedence in elision is given to the answer section, as it only occurs with mDNS to signify Known Answers [RFC6762]. The lowest precedence is given to the additional section, as it may contain EDNS OPT Pseudo-RRs, which are common in queries (see Section 3.2.2).

The representation of a DNS query is defined in Figure 10.

```
dns-query = [
  ? incl-question: bool .default false,
  ? flags: max-uint16 .default 0x0000,
  question-section,
  ? query-extra-sections,
]
question-section = [
  * full-question,
  ? last-question,
]
full-question = (
  domain-name,
  type-spec,
)
last-question = (
  domain-name,
  ? type-spec,
)
query-extra-sections = (
  ? answer-section,
  extra-sections,
)
answer-section = [+ $$dns-rr]
extra-sections = (
  ? authority: [+ $$dns-rr],
  additional: [+ $$dns-rr],
)
```

Figure 10: DNS Query Definition

3.4. DNS Responses

A DNS response is encoded as a CBOR array containing up to 5 entries.

1. An optional flag field (as unsigned integer),

2. An optional question section (as array, encoded as described in Section 3.3)

3. The answer section (as array),

4. An optional authority section (as array), and

5. An optional additional section (as array)

As for queries, the DNS transaction ID is elided and implied to be 0.

If the CBOR array is a response to a query for which the flags indicate that flags are set in the response, they MUST be set accordingly and thus included in the response. If the flags are not included, the flags are implied to be 0x8000 (everything unset except for the QR flag).

If the response includes only one array, then the DNS answer section represents an array of one or more DNS Resource Records (see Section 3.2).

If the response includes more than 2 arrays, the first entry may be the question section, identified by not being an array of arrays. It MUST be included, if the client requested it using the boolean field in the query. If it is present, it is followed by the answer section. The question section is encoded as specified in Section 3.3.

If the answer section is followed by one extra array, this array is the additional section. Like the answer section, the additional section is represented as an array of one or more DNS Resource Records (see Section 3.2).

If the answer section is followed by two extra arrays, the first is the authority section, and the second is the additional section. The authority section is also represented as an array of one or more DNS Resource Records (see Section 3.2).

The authority section is given precedence in elision over the additional section, as due to EDNS options or, e.g., CNAME answers that also provide the A/AAAA records. The additional section tends to show up more often than the authority section.

```
dns-response = [
  ? flags: max-uint16 .default 0x8000,
  ? question-section,
  answer-section,
  ? extra-sections,
]
```

Figure 11: DNS Response Definition

4. Compression with Packed CBOR

Packed CBOR [I-D.ietf-cbor-packed] is used for name compression in "application/dns+cbor".

If both DNS server and client support table setup tag 113 as described in Section 3.1 of [I-D.ietf-cbor-packed], it MAY be used for further compression in DNS responses. Especially IPv6 addresses, e.g., in AAAA resource records can benefit from straight referencing to compress common address prefixes.

4.1. Name Compression

```
Text-String-Suffix-Sequence-Packed-CBOR = #6.28259(rump)
```

Figure 12: CDDL for name compression based on Packed CBOR and as defined in this document.

For name compression, a new packing table setup tag TBD28259 ('n' and 'c' in ASCII) for Packed CBOR [I-D.ietf-cbor-packed] is defined. Any "application/dns+cbor" encoder and decoder MUST support tag TBD28259 for DNS queries and responses. It provides an implicit text string suffix sequence table for shared items _V_. This table _V_ is appended to the existing table for shared items of any table setup tag enclosing tag TBD28259 (by default empty table). This implicit (i.e., not explicitly represented) table _V_ is constructed as follows: Any coherent sequence of text strings encountered within the rump of tag TBD28259 when reading it depth-first, as well as any of its non-empty suffixes, ordered by their appearance within the rump, are added to the table as arrays marked with the splice integration tag 1115 (see [I-D.ietf-cbor-packed], Section 5.1). If a sequence for which a tagged array is already in _V_ is encountered, a shared item reference _i_ is added to the rump instead, splicing the content of the array within tag 1115 into the existing array (see [I-D.ietf-cbor-packed], Section 5.1) and that sequence as well as its non-empty suffixes are not added again to _V_. The resulting rump should look like referencing the first instance of the _i_-th complete string sequence (depth first) in the sequence.

It is NOT RECOMMENDED to use any other table building tag as defined in Section 3 of [I-D.ietf-cbor-packed] within TBD28259, as the implicit table of TBD28259 may otherwise lead to more complicating en- and decoding steps. Particularly, a one-pass decoding with minimal state of DNS messages SHOULD be ensured to be viable on constrained devices. If another table setup needs to be transported within TBD28259, e.g., certain RDATA or option values might elect to use Packed CBOR as well, they SHOULD be encapsulated as encoded CBOR within a byte string using tag 24 [RFC8949].

Due to the order of strings being important, special care should be taken for the order of map (major type 5, [RFC8949]) elements within tag TBD28259. The "application/dns+cbor" media type specified within this document avoids maps for this reason, in addition to enforcing orders derived from DNS specifications, e.g., [RFC9460] for SvcParams. If any other CBOR object than an object defined by dns-message in Figure 1 is compressed using TBD28259, map elements MUST be encoded in bytewise lexicographic order of their keys, as specified in Section 4.2.1 of [RFC8949], unless the definition of that particular object type provides a different predetermined order.

The "application/dns+cbor" media type comes with an optional parameter "packed". If it is not provided, the value of it is assumed to be 0. With the "application/dns+cbor;packed=0" media type (i.e., even with "application/dns+cbor"), the tag TBD28259 is implicit, i.e., it SHOULD NOT be sent. This avoids sending the 3 bytes of overhead generated by the presence of the tag. If the decoder encounters an object marked with "application/ dns+cbor;packed=0" that is tagged TBD28259, it MUST NOT discard it and parse its content as if it were implicit. Otherwise, it might incur an additional message to be sent and media type negotiation might fail unnecessarily.

The splice integration tag 1115 is the only integration tag in use for "application/dns+cbor" for "packed=0" and "packed=1" (see Section 5 of [I-D.ietf-cbor-packed]).

Name compression with an example message can be found in Section 8.3.

4.2. Further DNS Representation with Tag 113

The representation of DNS responses with packed value 1, i.e., "application/dns+cbor;packed=1", has the same semantics as for tag TBD113 (see Section 3.1 of [I-D.ietf-cbor-packed]) compressing the name-compressed response as rump. The difference to [I-D.ietf-cbor-packed] is that tag TBD113 is implicit with media type parameter "packed=1", see Figure 13.

Take the following definition:

```
packed-dns-message = [
  [*shared-and-argument-item],
  rump
]
```

Figure 13: Definition of packed DNS message with media type "application/ dns+cbor;packed=1".

When marked by the "application/dns+cbor;packed=1" media type and parameter it MUST be implicitly understood as the definition provided in Figure 14.

```
Packed-Text-String-Suffix-Sequence-Packed-CBOR = #6.113(
  [
    [*shared-and-argument-item],
    Text-String-Suffix-Sequence-Packed-CBOR
  ]
)
```

Figure 14: The implicit definition of a packed DNS message with media type "application/dns+cbor;packed=1".

If the decoder encounters an object marked with "application/ dns+cbor;packed=1" that is tagged TBD113 or with its rump tagged TBD28259, it MUST NOT discard it and parse its content as if it were implicit. Any object marked with "application/dns+cbor;packed=1" that does not fit the definition of packed-dns-message provided in Figure 13 or variants of explicit tags TBD113 or TBD28259 MUST be interpreted as an invalid message.

"packed=1" compression of queries is not specified, as apart from EDNS(0) (see Section 3.2.2), they only consist of one question most of the time, i.e., there is close to no redundancy that is not already covered by "packed=0" and encoding such an object can be too complex for a potentially constrained resolver.

An example of this mechanism, as well as pseudo-code for a simple decoder can be found in Section 8.5.

4.3. Media Type Negotiation

A DNS client tells a server that it would accept the media type "application/dns+cbor;packed=1" to negotiate (see, e.g., [RFC9110] or [RFC7252], Section 5.5.4) with the DNS server whether the server supports setup table tag TBD113. If it does, it MAY request the response to be in packed value 1 (media type "application/dns+cbor;packed=1"). The server then SHOULD reply with the response in Packed CBOR, which it also signals with media type "application/dns+cbor;packed=1". Otherwise, both fall back to the implicit "packed=0".

4.4. Compression

The method of the compressor to construct the packing table, i.e., how the compression is applied, is out of scope of this document.

5. Implementation Status

This section records the status of known implementations of the protocol defined by this specification at the time of posting of this Internet-Draft, and is based on a proposal described in [RFC7942]. The description of implementations in this section is intended to assist the IETF in its decision processes in progressing drafts to RFCs. Please note that the listing of any individual implementation here does not imply endorsement by the IETF. Furthermore, no effort has been spent to verify the information presented here that was supplied by IETF contributors. This is not intended as, and must not be construed to be, a catalog of available implementations or their features. Readers are advised to note that other implementations may exist.

According to [RFC7942], "this will allow reviewers and working groups to assign due consideration to documents that have the benefit of running code, which may serve as evidence of valuable experimentation and feedback that have made the implemented protocols more mature. It is up to the individual working groups to use this information as they see fit".

5.1. Python decoder/encoder

The authors of this document provide a decoder/encoder implementation (https://github.com/netd-tud/cbor4dns) of both the unpacked and packed format specified in this document in Python.

Level of maturity: prototype

Version compatibility: draft-lenders-dns-cbor-10

License: MIT

Contact information: Martine Lenders <martine.lenders@tu-dresden.de>

Last update of this information: July 2024

5.2. Embedded decoder/encoder

The authors of this document provide a decoder/encoder implementation (https://github.com/RIOT-OS/RIOT/pull/19989) of the unpacked format specified in this document for the RIOT operating system. It can only encode queries and decode responses.

Level of maturity: prototype

Version compatibility: draft-lenders-dns-cbor-08

License: MIT

Contact information: Martine Lenders <martine.lenders@tu-dresden.de>

Last update of this information: October 2023

6. Security Considerations

TODO Security

7. IANA Considerations

7.1. Media Type Registration

This document registers a media type for the serialization format of DNS messages in CBOR. It follows the procedures specified in [RFC6838].

7.1.1. "application/dns+cbor"

Type name: application

Subtype name: dns+cbor

Required parameters: None

Optional parameters: packed

Encoding considerations: Must be encoded as using [RFC8949]. See [TBD-this-spec] for details.

Security considerations: See Section 6 of this draft

Interoperability considerations: TBD

Published specification: [TBD-this-spec]

Applications that use this media type: TBD DNS over X systems

Fragment Identifier Considerations: TBD

Additional information:

Deprecated alias names for this type: N/A

Magic number(s): N/A

File extension(s): dnsc

Macintosh file type code(s): none

Person & email address to contact for further information: IETF CBOR Working Group (cbor@ietf.org) or IETF Applications and Real-Time Area (art@ietf.org)

Intended usage: COMMON

Restrictions on Usage: None?

Author: Martine S. Lenders m.lenders@fu-berlin.de (mailto:m.lenders@fu-berlin.de)

Change controller: IETF

Provisional registrations? No

7.2. CoAP Content-Format Registration

IANA is requested to assign CoAP Content-Format ID for the new DNS message media types in the "CoAP Content-Formats" sub-registry, within the "CoRE Parameters" registry [RFC7252], corresponding the "application/dns+cbor" media type specified in Section 7.1:

7.2.1. "application/dns+cbor"

Media-Type: application/dns+cbor

Encoding: -

Id: TBD53

Reference: [TBD-this-spec]

7.2.2. "application/dns+cbor;packed=1"

Media-Type: application/dns+cbor;packed=1

Encoding: -

Id: TBD54

Reference: [TBD-this-spec]

7.3. CBOR Tags Registry

In the registry "CBOR Tags" [IANA.cbor-tags], IANA is requested to allocate the tags defined in Table 1.

Tag	Data Item	Semantics	Reference
TBD141	array	CBOR EDNS option record	draft-lenders-dns-cbor
TBD28259	any	Packed CBOR; implicit text string suffix sequence shared-item table	draft-lenders-dns-cbor

Table 1: Values for Tag Numbers

8. Examples

8.1. CDDL model for e'' application extension

Figure 15 shows the CDDL model used for the e'' application extension (see [I-D.ietf-cbor-edn-e-ref]) in our examples. C- contants define DNS classes as unsigned integers from the "DNS CLASSes" sub-registry of the IANA "Domain Name System (DNS) Parameters" registry. RR-constants define resource record types from the "Resource Record (RR) TYPEs" sub-registry of the IANA "Domain Name System (DNS) Parameters" registry.

```
C-IN = 1
C-ANY = 255

RR-A = 1
RR-NS = 2
RR-CNAME = 5
RR-PTR = 12
RR-AAAA = 28
RR-ANY = 255
```

Figure 15: CDDL model defining constants for this document for e''.

8.2. DNS Queries

A DNS query of the record AAAA in class IN for name "example.org" is represented in CBOR extended diagnostic notation (EDN) [I-D.ietf-cbor-edn-literals] with e'' application extension [I-D.ietf-cbor-edn-e-ref] as follows:

```
[["example", "org"]]
```

The binary (in hexadecimal encoding) of the query looks as follows (14 bytes):

```
81                     # array(1)
   82                  # array(2)
      67               # text(7)
         6578616d706c65 # "example"
      63               # text(3)
         6f7267         # "org"
```

A query of an A record for the same name is represented as

```
[["example", "org", e'RR-A']]
```

or in binary (15 bytes)

```
81                     # array(1)
   83                  # array(3)
      67               # text(7)
         6578616d706c65 # "example"
      63               # text(3)
         6f7267         # "org"
      01               # e'RR-A' (unsigned(1))
```

A query of ANY record for that name is represented as

```
[["example", "org", e'RR-ANY', e'C-ANY']]
```

or in binary (18 bytes)

```
81                      # array(1)
   84                   # array(4)
      67                # text(7)
         6578616d706c65 # "example"
      63                # text(3)
         6f7267         # "org"
      18 ff             # e'RR-ANY' (unsigned(255))
      18 ff             # e'RR-ANY' (unsigned(255))
```

8.3. Name Compression

Take the following response _o_ in CBOR extended diagnostic notation (EDN) [I-D.ietf-cbor-edn-literals]. It contains a question for the "www.example.org" AAAA record, with a "svc.www.example.org" CNAME answer and its AAAA record, as well as the "org.example.org" NS record for the "example.org" domain in the authority section, each record with TTL 3600:

```
[
  ["www", "example", "org"],
  [
    [3600, "www", "example", "org", e'RR-CNAME', "svc", "www", "example", "org"],
    [3600, "svc", "www", "example", "org", ip'2001:db8::1']
  ],
  [
    [3600, "example", "org", e'RR-NS', "org", "example", "org"]
  ],
  []
]
```

Figure 16: Unpacked example for implicit name compression.

The object _o_ would be sent over the wire as the following 136 bytes of binary data, represented here with hexadecimal encoding (type and data noted in comment).

```
84                                  # array(4)
   83                               # array(3)
      63                            # text(3)
         777777                     # "www"
      67                            # text(7)
         6578616D706C65             # "example"
      63                            # text(3)
         6F7267                     # "org"
   82                               # array(2)
      89                            # array(9)
```

```
            19 0E10                          # unsigned(3600)
            63                               # text(3)
               777777                        # "www"
            67                               # text(7)
               6578616D706C65                # "example"
            63                               # text(3)
               6F7267                        # "org"
            05                               # e'RR-CNAME' (unsigned(5))
            63                               # text(3)
               737663                        # "svc"
            63                               # text(3)
               777777                        # "www"
            67                               # text(7)
               6578616D706C65                # "example"
            63                               # text(3)
               6F7267                        # "org"
         86                                  # array(6)
            19 0E10                          # unsigned(3600)
            63                               # text(3)
               737663                        # "svc"
            63                               # text(3)
               777777                        # "www"
            67                               # text(7)
               6578616D706C65                # "example"
            63                               # text(3)
               6F7267                        # "org"
            50                               # bytes(16)
               20010DB8000000000000000000000001 # ip'2001:db8::1'
      81                                     # array(1)
         87                                  # array(7)
            19 0E10                          # unsigned(3600)
            67                               # text(7)
               6578616D706C65                # "example"
            63                               # text(3)
               6F7267                        # "org"
            02                               # e'RR-NS' (unsigned(2))
            63                               # text(3)
               6F7267                        # "org"
            67                               # text(7)
               6578616D706C65                # "example"
            63                               # text(3)
               6F7267                        # "org"
      80                                     # array(0)
```

Figure 17: Binary of the unpacked example for implicit name compression (136 bytes).

This would generate the following virtual table _V_.

```
[
    /index 0:/ 1115(["www", "example", "org"]),
    /index 1:/ 1115(["example", "org"]),
    /index 2:/ 1115(["org"]),
    /index 3:/ 1115(["svc", "www", "example", "org"]),
    /index 4:/ 1115(["org", "example", "org"])
]
```

Figure 18: Implicit table of shared items for the example _o_.

An implementation MAY choose to use references here to reduce memory consumption, as represented in Figure 19. Circular references are not possible due to the construction algorithm of _V_.

```
[
    /index 0:/ 1115(["www", "example", "org"]),
    /index 1:/ 1115(["example", "org"]),
    /index 2:/ 1115(["org"]),
    /index 3:/ 1115(["svc", simple(0)]),
    /index 4:/ 1115(["org", simple(1)])
]
```

Figure 19: Implicit table of shared items for the example _o_ with self- references.

Note that the sequence "org", simple(0) is added at index 4 with leading "org", instead of referencing index 2 + index 1 (simple(2), simple(1)), as it is its own distinct suffix sequence. However, its suffix "example", "org" is not added to the table again, as it is already present at index 1.

Assuming media type "application/dns+cbor;packed=0" (i.e., an implicit tag TBD28259), the packed representation of _o_ would thus be:

```
[
  ["www", "example", "org"],
  [
    [
      3600,
      simple(0) / expands to "www", "example", "org" /,
      e'RR-CNAME',
      "svc",
      simple(0) / expands to "www", "example", "org" /
    ],
    [
      3600,
      simple(3) / expands to "svc", "www", "example", "org" /
      ip'2001:db8::1'
    ]
  ],
  [
    [
      3600,
      simple(1) / expands to "example", "org" /,
      e'RR-NS',
      "org",
      simple(1) / expands to "example", "org" /,
    ]
  ],
  []
]
```

Figure 20: The packed representation of the name compression example.

In binary the packed representation of _o_ would be:

```
84                                  # array(4)
   83                               # array(3)
      63                            # text(3)
         777777                     # "www"
      67                            # text(7)
         6578616D706C65             # "example"
      63                            # text(3)
         6F7267                     # "org"
   82                               # array(2)
      85                            # array(5)
         19 0E10                    # unsigned(3600)
         E0                         # simple(0)
         05                         # e'RR-CNAME' (unsigned(5))
         63                         # text(3)
            737663                  # "svc"
         E0                         # simple(0)
      83                            # array(3)
         19 0E10                    # unsigned(3600)
         E3                         # simple(3)
         50                         # bytes(16)
            20010DB800000000000000000000001 # ip'2001:db8::1'
   81                               # array(1)
      85                            # array(5)
         19 0E10                    # unsigned(3600)
         E1                         # simple(1)
         02                         # e'RR-NS' (unsigned(2))
         63                         # text(3)
            6F7267                  # "org"
         E1                         # simple(1)
   80                               # array(0)
```

Figure 21: Binary of the packed representation of the example (65 bytes).

8.4. DNS Responses

The responses to the examples provided in Section 8.2 are shown below. We use the CBOR extended diagnostic notation (EDN) (see [I-D.ietf-cbor-edn-literals] and Appendix G of [RFC8610]) with e'' application extension [I-D.ietf-cbor-edn-e-ref], most notably the "ip" extension to represent binary IP addresses as a IP address app-string literal.

To represent an AAAA record with TTL 300 seconds for the IPv6 address 2001:db8::1, a minimal response to [["example", "org"]] could be

```
[[[300, ip'2001:db8::1']]]
```

or in binary (23 bytes)

```
81                                          # array(1)
   81                                       # array(1)
      82                                    # array(2)
         19 012c                            # unsigned(300)
         50                                 # bytes(16)
            20010db8000000000000000000000001  # ip'2001:db8::1'
```

In this case, the name is derived from the query.

If the name or the context is required, the following response would also be valid:

```
[["example", "org", 300, ip'2001:db8::1']]
```

In binary that response looks like the following (35 bytes):

```
81                                          # array(1)
   81                                       # array(1)
      84                                    # array(4)
         67                                 # text(7)
            6578616d706c65                  # "example"
         63                                 # text(3)
            6f7267                          # "org"
         19 012c                            # unsigned(300)
         50                                 # bytes(16)
            20010db8000000000000000000000001  # ip'2001:db8::1'
```

If the query can not be mapped to the response for some reason, a response would look like:

```
[["example", "org"], [[300, ip'2001:db8::1']]]
```

In binary that response looks like the following (36 bytes):

```
82                                          # array(2)
   82                                       # array(2)
      67                                    # text(7)
         6578616d706c65                     # "example"
      63                                    # text(3)
         6f7267                             # "org"
   81                                       # array(1)
      82                                    # array(2)
         19 012c                            # unsigned(300)
         50                                 # bytes(16)
            20010db8000000000000000000000001  # ip'2001:db8::1'
```

To represent a minimal response of an A record with TTL 3600 seconds for the IPv4 address 192.0.2.1, a minimal response to [["example", "org", 1]] could be

```
[[[300, ip'192.0.2.1']]]
```

or in binary (11 bytes)

```
81                  # array(1)
   81               # array(1)
      82            # array(2)
         19 012c    # unsigned(300)
         44         # bytes(4)
            c0000201 # ip'192.0.2.1'
```

Note that here also the 1 of record type A can be elided, as this record type is specified in the question section.

Lastly, a response to [["example", "org", e'RR-ANY', e'C-ANY']] could be

```
[
  # PTR (12) question for "example.org"
  [
    # appends 0 => ["example", "org"] to virtual packing table
    "example",
    # appends 1 => ["org"] to virtual packing table
    "org",
    e'RR-PTR'
  ],
  # Answer section:
  [[
    # PTR (12) for "example.org"
    # (both elided since they are the same as in question)
    # is "_coap._udp.local" with TTL 3600
    3600,
    # appends 2 => ["_coap", "_udp", "local"] to virtual packing table
    "_coap",
    # appends 3 => ["_udp", "local"] to virtual packing table
    "_udp",
    # appends 4 => ["local"] to virtual packing table
    "local"
  ]],
  # Authority section:
  [
    [
      # NS (2) for "example.org"
      # (name elided since its the same as in question)
```

```
        # is "ns1.example.org" with TTL 3600
        3600, e'RR-NS',
        # appends 5 => ["ns1", simple(0)] to virtual packing table
        "ns1", simple(0)  # expands to ["example", "org"]
      ],
      [
        # NS (2) for "example.org"
        # (name elided since its the same as in question)
        # is "ns2.example.org" with TTL 3600
        3600, e'RR-NS',
        # appends 6 => ["ns2", simple(0)] to virtual packing table
        "ns2", simple(0)  # expands to ["example", "org"]
      ]
    ],
    # Additional section
    [
      [
        # AAAA (28) for "_coap._udp.local"
        # is 2001:db8::1 with TTL 3600
        simple(2),    # expands to ["_coap", "_udp", "local"]
        3600, e'RR-AAAA', ip'2001:db8::1'
      ],
      [
        # AAAA (28) for "_coap._udp.local"
        # is 2001:db8::2 with TTL 3600
        simple(2),    # expands to ["_coap", "_udp", "local"]
        3600, e'RR-AAAA', ip'2001:db8::2'
      ],
      [
        # AAAA (28) for "ns1.example.org"
        # is 2001:db8::35 with TTL 3600
        simple(5),    # expands to ["ns1", ["example", "org"]]
        3600, e'RR-AAAA', ip'2001:db8::35'
      ],
      [
        # AAAA (28) for "ns2.example.org"
        # is 2001:db8::3535 with TTL 3600
        simple(6),    # expands to ["ns2", ["example", "org"]
        3600, e'RR-AAAA', ip'2001:db8::3535'
      ]
    ]
  ]
```

or in binary (155 bytes)

```
84                                          # array(4)
   83                                       # array(3)
      67                                    # text(7)
         6578616d706c65                     # "example"
      63                                    # text(3)
         6f7267                             # "org"
      0c                                    # e'RR-PTR' (unsigned(12))
   81                                       # array(1)
      84                                    # array(4)
         19 0e10                            # unsigned(3600)
         65                                 # text(5)
            5f636f6170                      # "_coap"
         64                                 # text(4)
            5f756470                        # "_udp"
         65                                 # text(5)
            6c6f63616c                      # "local"
   82                                       # array(2)
      84                                    # array(4)
         19 0e10                            # unsigned(3600)
         02                                 # e'RR-NS' (unsigned(2))
         63                                 # text(3)
            6e7331                          # "ns1"
         e0                                 # simple(0)
      84                                    # array(4)
         19 0e10                            # unsigned(3600)
         02                                 # e'RR-NS' (unsigned(2))
         63                                 # text(3)
            6e7332                          # "ns2"
         e0                                 # simple(0)
   84                                       # array(4)
      84                                    # array(4)
         e2                                 # simple(2)
         19 0e10                            # unsigned(3600)
         18 1c                              # e'RR-AAAA' (unsigned(28))
         50                                 # bytes(16)
            20010db8000000000000000000000001  # ip'2001:db8::1'
      84                                    # array(4)
         e2                                 # simple(2)
         19 0e10                            # unsigned(3600)
         18 1c                              # e'RR-AAAA' (unsigned(28))
         50                                 # bytes(16)
            20010db8000000000000000000000002  # ip'2001:db8::2'
      84                                    # array(4)
         e5                                 # simple(5)
         19 0e10                            # unsigned(3600)
         18 1c                              # e'RR-AAAA' (unsigned(28))
         50                                 # bytes(16)
            20010db8000000000000000000000035  # ip'2001:db8::35'
```

```
   84                                      # array(4)
      e6                                   # simple(6)
      19 0e10                              # unsigned(3600)
      18 1c                                # e'RR-AAAA' (unsigned(28))
      50                                   # bytes(16)
         20010db8000000000000000000003535  # ip'2001:db8::3535'
```

This response advertises two local CoAP servers (identified by service name _coap._udp.local) at 2001:db8::1 and 2001:db8::2 and two nameservers for the example.org domain, ns1.example.org at 2001:db8::35 and ns2.example.org at 2001:db8::3535. Each of the transmitted records has a TTL of 3600 seconds. Note the use of name compression (see Section 4.1) in this example.

8.5. Name Compression and Packed CBOR

Figure 22 shows the example from Figure 16 with "packed=1".

```
[
  ["org", 3600],
  [
    ["www", "example", simple(0) / expands to "org" /],
    [
      [
        simple(1) / expands to 3600 /,
        simple(2) / expands to "www", "example", "org" /,
        e'RR-CNAME',
        "svc",
        simple(2) / expands to "www", "example", "org" /
      ],
      [
        simple(1) / expands to 3600,
        simple(5) / expands to "svc", "www", "example", "org" /
        ip'2001:db8::1'
      ]
    ],
    [
      [
        simple(1) / expands to 3600 /,
        simple(3) / expands to "example", "org" /,
        e'RR-NS',
        simple(0) / expands to "org" /,
        simple(3) / expands to "example", "org" /,
      ]
    ],
    []
  ]
]
```

Figure 22: The packed representation of the example of a message marked with "application/dns+cbor;packed=1".

Note, that the encoder needs to rewrite the references from Figure 16 as the implicit table _V_ is appended to the shared argument table ["org", 600].

In binary, that example looks like the following (represented in hexadecimal):

```
82                                  # array(2)
   82                               # array(2)
      63                            # text(3)
         6F7267                     # "org"
      19 0E10                       # unsigned(600)
   84                               # array(4)
      83                            # array(3)
         63                         # text(3)
            777777                  # "www"
         67                         # text(7)
            6578616D706C65          # "example"
         E0                         # simple(0)
      82                            # array(2)
         85                         # array(5)
            E1                      # simple(1)
            E2                      # simple(2)
            05                      # e'RR-CNAME' (unsigned(5))
            63                      # text(3)
               737663               # "svc"
            E2                      # simple(2)
         83                         # array(3)
            E1                      # simple(1)
            E5                      # simple(5)
            50                      # bytes(16)
               20010DB8000000000000000000000001 # ip'2001:db8::1'
      81                            # array(1)
         85                         # array(5)
            E1                      # unsigned(600)
            E3                      # simple(3)
            02                      # e'RR-NS' (unsigned(2))
            E0                      # simple(0)
            E3                      # simple(3)
      80                            # array(0)
```

Figure 23: The binary of the packed representation of the example marked with "application/dns+cbor;packed=1" (62 bytes).

Implicitly, a decoder would interpret this as the following EDN:

```
TBD113(
  [
    ["org", 3600],
    TBD28259(
      [
        ["www", "example", simple(0)],
        [
          [
            simple(1),
            simple(2) / expands to "www", "example", "org" /,
            e'RR-CNAME',
            "svc",
            simple(2) / expands to "www", "example", "org" /
          ],
          [
            simple(1),
            simple(5) / expands to "svc", "www", "example", "org" /
            ip'2001:db8::1'
          ]
        ],
        [
          [
            simple(1),
            simple(3) / expands to "example", "org" /,
            e'RR-NS',
            simple(0),
            simple(3) / expands to "example", "org" /,
          ]
        ],
        []
      ]
    )
  ]
)
```

Figure 24: The explicit interpretation of the packed representation of the example of a message marked with "application/dns+cbor;packed=1".

8.5.1. Example Decoder

A decoder to interpret an object like the one in Figure 22 would have the following pseudo-code:

```
function decode_cbor_dns(binary: bytes, packed: uint = 0): CBORObject {
  obj: CBORObject = cbor.decode(binary)
  unpacker: Unpacker = Unpacker()
  match (packed) {
    when 0 then {
      packing_table = []
      rump = obj
    },
    when 1 then {
      /* except explicit 113 */
      if (typeof(obj) is CBORTag and tag-number of obj == 113) {
        obj = tag-content of obj
      }

      assume that typeof(obj) is CBORArray and obj.length == 2
      assume that typeof(obj[0]) is CBORArray  # packing table

      /* step into tag 113 context for unpacker with all its references
       * (simple values and tag 6) and function tags */
      Tell ‘unpacker‘ that we are in rump of tag 113 now

      /* prepend to current packing table of unpacker */
      unpacker.packing_table = obj[0] concat unpacker.packing_table
      rump = obj[1]
    },
    when anything else then {
      throw "Not supported yet!"
    },
  }

  return unpack_names(rump, unpacker)
}

function unpack_names(rump: Any, unpacker: Unpacker): CBORObject {
  /* except explicit 28259 */
  if (typeof(rump) is CBORTag and tag-number of rump == 28259) {
    rump = tag-content of rump
  }

  /* step into tag 28259 context for unpacker with all its references
   * (simple values and tag 6) */
  Tell ‘unpacker‘ that we are in rump of tag 28259 now

  return recursive_unpack_names(rump, unpacker, unpacker.packing_table.length)
}

function is_splice_tag(obj: CBORObject): bool {
  return (typeof(packed_idx) is CBORTag and tag-number of packed_idx == 1115)
```

```
}

function recursive_unpack_names(
    obj: CBORObject, unpacker: Unpacker, outer_table_len: Integer
): CBORObject {
  match typeof(obj) {
    when CBORInt, CBORByteString, or CBORTextString then {
      return obj
    }
    when CBORArray then {
      result: CBORArray = []
      /* name_start_idx will point to first element of a name and its suffixes
       * in the packing table */
      name_start_idx: CBORInt or null = null
      for (elem in obj) {
        if (typeof(elem) is CBORTextString) {
          /* Create a local name reference in packing table for this name
           * and all its suffixes */
          if (name_start_idx is null) {
            unpacker.packing_table = unpacker.packing_table concat [
              CBORTag(tag-number = 1115, tag-content = []))
            ]
            name_start_idx = unpacker.packing_table.length - 1
          }
          else {
            /* create packing table entry for new suffix */
            unpacker.packing_table = unpacker.packing_table concat [
              CBORTag(tag-number = 1115, tag-content = []))
            ]
          }

          /* Append to all suffixes in local name reference */
          for (i from i == name_start_idx
                 to i < unpacker.packing_table.length)) {
            append elem to tag-content of unpacker.packing_table[i]
          }
          append elem to result
        }
        elif (unpacker.is_shared_reference(elem)) {   /* is simple() or 6() */
          /* calculate index from simple() or 6() */
          ref_idx: uint = unpacker.ref_idx(elem)
          assume that (ref_idx < unpacker.packing_table.length)

          packed_obj: CBORObject = unpacker.packing_table[ref_idx]

          /* check if this reference is part of a longer name */
          if (is_splice_tag(packed_obj) or
              typeof(packed_obj) is CBORTextString) {
```

```
            if (name_start_idx is not null) {
              /* if name_start_idx is already set, append to all suffixes
               * from name_start_idx to end of packing table */
              for (i from i == name_start_idx
                     to i < unpacker.packing_table.length)) {
                append elem to tag-content of unpacker.packing_table[i]
              }

              /* check if ref_idx references implicit table V */
              if (ref_idx > outer_table_len) {
                /* close name_start_idx pointing to first element of name
                 * sequence, as only CBORTextStrings (including those
                 * referenced in outer tables) continue the name */
                name_start_idx = null
              }
            }
          }
          if (is_splice_tag(packed_obj)) {
            /* name suffix is spliced in */
            assume that (
              typeof(tag-content of packed_obj) is CBORArray containing
              only CBORTextString and references to CBORTextSTring
            )
            result = result concat tag-content of packed_obj
          }
          else {
            append packed_obj to result
          }
        }
        else {
          /* not part of a name anymore, so close name_start_idx pointing to
           * first element of name sequence */
          name_start_idx = null
          match typeof(elem) {
            /* step into arrays and tags */
            when CBORArray then {
              result = result concat recursive_unpack_names(
                  elem, unpacker, outer_table_len
              )
            }
            when CBORTag then {
              append CBORTag(
                tag-number = tag-number of elem,
                tag-content = recursive_unpack_names(
                    tag-content of elem, unpacker, outer_table_len
                ),
              ) to result
            }
```

```
            when CBORMap then {
              throw warning "Ignore maps which are not part of this spec"
            }
            when anything else then {
              append elem to result
            }
          }
        }
      }
      return result
    }
    when anything else then {
      throw warning "Ignore other types not part of this spec"
    }
  }
}
```

Figure 25: Pseudo-code for an example decoder

9. Comparison to Classic DNS Wire Format

Table 2 shows a comparison between the classic DNS wire format and the application/dns+cbor format. Note that the worst case results typically appear only rarely in DNS. The classic DNS format is preferred in those cases. A key for which configuration was used in which case can be seen in Table 3. Any name label that is longer than 23 bytes adds a name overhead of 1 byte to its CBOR type header.
// TBD: Also add structured RRs?.
//
// -- 窶芭lenders

Item	Classic DNS format [bytes]	application/dns+cbor [bytes]		
		best case	realistic worst case	theoretical worst case
Header (ID & Flags)	4	1	4	4
Count fields	2	1	3	3
Question section	6 + name len.	2 + name len.	6 + name len. + name overhead	9 + name len. + name overhead
Standard RR	12 + name len. + rdata len.	3 + rdata len.	14 + name len. + rdata len. + name overhead	17 + name len. + rdata len. + name overhead
Standard RR with name rdata	12 + name len. + rdata len.	4	14 + name len. + rdata len. + name overheads	16 + name len. + rdata len. + name overheads
EDNS Opt Pseudo-RR	11 + options	2 + options	6 + options	14 + options
EDNS Option	4 + value len.	2 + value len.	4 + value len.	6 + value len.

Table 2: Comparison of application/dns+cbor to classic DNS format.

Item	application/dns+cbor configuration		
	best case	realistic worst case	theoretical worst case
Header (ID & Flags)	Flags elided	QR, Opcode, AA, TC, or RD are set	QR, Opcode, AA, TC, or RD are set
Count fields	Encoded in CBOR array header	Encoded in CBOR array header, >255 records in section	Encoded in CBOR array header, >255 records in section
Question section	Class, type, and name elided	Type > 255, label len. > 23	Type > 255, Class > 255, label len. > 23
Standard RR	Class, type, and name elided, rdata len. < 24	Type > 255, label len. > 23 rdata len. > 255	Type > 255, Class > 255, label len. > 23 rdata len. > 255
Standard RR with name rdata	Class, type, and name elided, simple(i) with i < 16	Type > 255, label len. > 23 name uncompressed	Type > 255, Class > 255, label len. > 23 name uncompressed
EDNS Opt Pseudo-RR	All EDNS(0) fields elided	Rcode < 24, DO flag set,	UDP payload len. > 255 Rcode > 255 Version > 255 DO flag set
EDNS Option	Code < 24 Length < 24	Code < 24 Length > 255	Code > 255 Length > 255

Table 3: Configuration key for Table 2 .

10. Change Log

10.1. Since draft-lenders-dns-cbor-15 (https://datatracker.ietf.org/doc/html/draft-lenders-dns-cbor-15)

* Add Vadim Goncharov as contributor
* Add Mikolai Gテシtschow as co-author
* Make options in OPT record a flat list of alternating key-value-pairs
* Use e'' application extension in examples for better readability of RR types and clases
* Pass on name compression
  - TBD28259 now appends, rather than prepends to existing packing tables to allow for single pass decoders with minimum state.
  - Consequently, disallow inner table setup tags to TBD28259, as that would increase complexity of both decoders and encoders
  - Add considerations on maps within TBD28259 (see also #15)
  - Clarify implicit nature with packed
  - Provide actual DNS examples
  - Add example decoder as pseudo-code

10.2. Since draft-lenders-dns-cbor-14 (https://datatracker.ietf.org/doc/html/draft-lenders-dns-cbor-14)

* Correction and nits
* Explicitly state which integration tags are in use
* Add binary examples in CBOR-pretty
* Unify formatting for and mention explicitly application/dns+cbor from the top
* Explicitly define behavior for query[0] = True in response text
* Clarifications around media type parameter packed=1
* Remove TBD reference (it never came to be)

10.3. Since draft-lenders-dns-cbor-13 (https://datatracker.ietf.org/doc/html/draft-lenders-dns-cbor-13)

* Make use of splicing integration tag 1115
  - Make domain names flat text string sequences again
* Add capability to summarize rrsets
* Provide extension point for IP addresses

10.4. Since draft-lenders-dns-cbor-12 (https://datatracker.ietf.org/doc/html/draft-lenders-dns-cbor-12)

* Fix bug in packed examples
* Improve compression examples for clarity

10.5. Since draft-lenders-dns-cbor-11 (https://datatracker.ietf.org/doc/html/draft-lenders-dns-cbor-11)

* Update repo links to cbor-wg org in draft
* s/CBOR-packed/Packed CBOR/
* Small pass on wording
* Remove commented-out parts
* Make name compression be based on Packed CBOR

10.6. Since draft-lenders-dns-cbor-10 (https://datatracker.ietf.org/doc/html/draft-lenders-dns-cbor-10)

* Address IANA #1392416 early review
* Fix external section references
* Update implementation status

10.7. Since draft-lenders-dns-cbor-09 (https://datatracker.ietf.org/doc/html/draft-lenders-dns-cbor-09)

* Add recommendation on label encoding
* Provide extension points
  - Mark dns-rr specifically as extension point

  - Provide extension points for parameter values (options and svc-params)

* Point out CBOR-packed needs to be unpacked when identifying names

* Distinguish from C-DNS [RFC8618]

* State objectives in introduction

* Fix nits and typos

10.8. Since draft-lenders-dns-cbor-08 (https://datatracker.ietf.org/doc/html/draft-lenders-dns-cbor-08)

* Clarify why question section was designed the way it is

* Add answer section to queries for Known Answers in mDNS

* Express names as sequence of labels

* Provide dedicated types for more structured RDATA

* Add RFC1035-like name compression

* Add switching boolean to query message to explicitly have question present in response

* Make EDNS options a map

* Update examples and comparison table in appendices

* Update implementation section

10.9. Since draft-lenders-dns-cbor-07 (https://datatracker.ietf.org/doc/html/draft-lenders-dns-cbor-07)

* Add Section 9 with comparison to classic DNS wire format

* "wire format" -> "classic DNS wire format"

10.10. Since draft-lenders-dns-cbor-06 (https://datatracker.ietf.org/doc/html/draft-lenders-dns-cbor-06)

* Fixes wording and spelling mistakes

10.11. Since draft-lenders-dns-cbor-05 (https://datatracker.ietf.org/doc/html/draft-lenders-dns-cbor-05)

* Fix Section 7.2.1 title
* Amend for capability to carry more than one question
* Hint at future of name compression in later draft versions
* Use canonical name for CBOR-packed

10.12. Since draft-lenders-dns-cbor-04 (https://datatracker.ietf.org/doc/html/draft-lenders-dns-cbor-04)

* Add Implementation Status section
* Remove int as representation for rdata
* Add note on representation of more structured rdata

10.13. Since draft-lenders-dns-cbor-03 (https://datatracker.ietf.org/doc/html/draft-lenders-dns-cbor-03)

* Provide format description for EDNS OPT Pseudo-RRs
* Simplify CDDL to more idiomatic style
* Remove DNS transaction IDs

10.14. Since draft-lenders-dns-cbor-02 (https://datatracker.ietf.org/doc/html/draft-lenders-dns-cbor-02)

* Add Discussion section and note on compression

10.15. Since draft-lenders-dns-cbor-01 (https://datatracker.ietf.org/doc/html/draft-lenders-dns-cbor-01)

* Use MIME type parameter for packed instead of own MIME type
* Update definitions to accommodate for TID and flags, as well as more sections in query
* Clarify fallback to wire-format

10.16. Since draft-lenders-dns-cbor-00 (https://datatracker.ietf.org/doc/html/draft-lenders-dns-cbor-00)

* Add support for DNS transaction IDs
* Name and Address compression utilizing CBOR-packed
* Minor fixes to CBOR EDN and CDDL

Acknowledgments

We want to extend special thanks to Christian Amsテシss for his input in the brainstorming session that resulted in the current form of name compression.

References

Normative References

[I-D.ietf-cbor-edn-e-ref]
Bormann, C., "External References to Values in CBOR Diagnostic Notation (EDN)", Work in Progress, Internet-Draft, draft-ietf-cbor-edn-e-ref-02, 2 July 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-cbor-edn-e-ref-02>.

[I-D.ietf-cbor-edn-literals]
Bormann, C., "CBOR Extended Diagnostic Notation (EDN)", Work in Progress, Internet-Draft, draft-ietf-cbor-edn-literals-19, 16 October 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-cbor-edn-literals-19>.

[I-D.ietf-cbor-packed]
Bormann, C. and M. Gテシtschow, "Packed CBOR", Work in Progress, Internet-Draft, draft-ietf-cbor-packed-19, 2 February 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-cbor-packed-19>.

[IANA.cbor-tags]
IANA, "Concise Binary Object Representation (CBOR) Tags", <https://www.iana.org/assignments/cbor-tags>.

[RFC1035] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, DOI 10.17487/RFC1035, November 1987, <https://www.rfc-editor.org/rfc/rfc1035>.

[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>.

[RFC2782] Gulbrandsen, A., Vixie, P., and L. Esibov, "A DNS RR for specifying the location of services (DNS SRV)", RFC 2782, DOI 10.17487/RFC2782, February 2000, <https://www.rfc-editor.org/rfc/rfc2782>.

[RFC3596] Thomson, S., Huitema, C., Ksinant, V., and M. Souissi, "DNS Extensions to Support IP Version 6", STD 88, RFC 3596, DOI 10.17487/RFC3596, October 2003, <https://www.rfc-editor.org/rfc/rfc3596>.

[RFC5891] Klensin, J., "Internationalized Domain Names in Applications (IDNA): Protocol", RFC 5891, DOI 10.17487/RFC5891, August 2010, <https://www.rfc-editor.org/rfc/rfc5891>.

[RFC6838] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, DOI 10.17487/RFC6838, January 2013, <https://www.rfc-editor.org/rfc/rfc6838>.

[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>.

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <https://www.rfc-editor.org/rfc/rfc7252>.

[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>.

[RFC8610] Birkholz, H., Vigano, C., and C. Bormann, "Concise Data Definition Language (CDDL): A Notational Convention to Express Concise Binary Object Representation (CBOR) and JSON Data Structures", RFC 8610, DOI 10.17487/RFC8610, June 2019, <https://www.rfc-editor.org/rfc/rfc8610>.

[RFC8949] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", STD 94, RFC 8949, DOI 10.17487/RFC8949, December 2020, <https://www.rfc-editor.org/rfc/rfc8949>.

[RFC9460] Schwartz, B., Bishop, M., and E. Nygren, "Service Binding and Parameter Specification via the DNS (SVCB and HTTPS Resource Records)", RFC 9460, DOI 10.17487/RFC9460, November 2023, <https://www.rfc-editor.org/rfc/rfc9460>.

Informative References

[I-D.ietf-core-dns-over-coap] Lenders, M. S., Amsテシss, C., Gテシndoト斂n, C., Schmidt, T. C., and M. Wテ、hlisch, "DNS over CoAP (DoC)", Work in Progress, Internet-Draft, draft-ietf-core-dns-over-coap-20, 16 September 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-core-dns-over-coap-20>.

[RFC4944] Montenegro, G., Kushalnagar, N., Hui, J., and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks", RFC 4944, DOI 10.17487/RFC4944, September 2007, <https://www.rfc-editor.org/rfc/rfc4944>.

[RFC6282] Hui, J., Ed. and P. Thubert, "Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks", RFC 6282, DOI 10.17487/RFC6282, September 2011, <https://www.rfc-editor.org/rfc/rfc6282>.

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

[RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <https://www.rfc-editor.org/rfc/rfc7228>.

[RFC7942] Sheffer, Y. and A. Farrel, "Improving Awareness of Running Code: The Implementation Status Section", BCP 205, RFC 7942, DOI 10.17487/RFC7942, July 2016, <https://www.rfc-editor.org/rfc/rfc7942>.

[RFC8484] Hoffman, P. and P. McManus, "DNS Queries over HTTPS (DoH)", RFC 8484, DOI 10.17487/RFC8484, October 2018, <https://www.rfc-editor.org/rfc/rfc8484>.

[RFC8499] Hoffman, P., Sullivan, A., and K. Fujiwara, "DNS Terminology", RFC 8499, DOI 10.17487/RFC8499, January 2019, <https://www.rfc-editor.org/rfc/rfc8499>.

[RFC8618] Dickinson, J., Hague, J., Dickinson, S., Manderson, T., and J. Bond, "Compacted-DNS (C-DNS): A Format for DNS Packet Capture", RFC 8618, DOI 10.17487/RFC8618, September 2019, <https://www.rfc-editor.org/rfc/rfc8618>.

[RFC8724] Minaburo, A., Toutain, L., Gomez, C., Barthel, D., and JC. Zuniga, "SCHC: Generic Framework for Static Context Header Compression and Fragmentation", RFC 8724, DOI 10.17487/RFC8724, April 2020, <https://www.rfc-editor.org/rfc/rfc8724>.

[RFC8824] Minaburo, A., Toutain, L., and R. Andreasen, "Static Context Header Compression (SCHC) for the Constrained Application Protocol (CoAP)", RFC 8824, DOI 10.17487/RFC8824, June 2021, <https://www.rfc-editor.org/rfc/rfc8824>.

[RFC8945] Dupont, F., Morris, S., Vixie, P., Eastlake 3rd, D., Gudmundsson, O., and B. Wellington, "Secret Key Transaction Authentication for DNS (TSIG)", STD 93, RFC 8945, DOI 10.17487/RFC8945, November 2020, <https://www.rfc-editor.org/rfc/rfc8945>.

[RFC9110] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Semantics", STD 97, RFC 9110, DOI 10.17487/RFC9110, June 2022, <https://www.rfc-editor.org/rfc/rfc9110>.

[RFC9619] Bellis, R. and J. Abley, "In the DNS, QDCOUNT Is (Usually) One", RFC 9619, DOI 10.17487/RFC9619, July 2024, <https://www.rfc-editor.org/rfc/rfc9619>.

Contributors

Vadim Goncharov
Email: vadimnuclight@gmail.com

Authors' Addresses

Martine Sophie Lenders
TUD Dresden University of Technology
Helmholtzstr. 10
D-01069 Dresden
Germany
Email: martine.lenders@tu-dresden.de

Carsten Bormann
Universitテ、 t Bremen TZI
Postfach 330440
D-28359 Bremen
Germany
Phone: +49-421-218-63921
Email: cabo@tzi.org

Mikolai Gテシtschow
TUD Dresden University of Technology
Helmholtzstr. 10
D-01069 Dresden
Germany
Email: mikolai.guetschow@tu-dresden.de

Thomas C. Schmidt
HAW Hamburg
Email: t.schmidt@haw-hamburg.de

Matthias Wテ、hlisch
TUD Dresden University of Technology & Barkhausen Institut
Helmholtzstr. 10
D-01069 Dresden
Germany
Email: m.waehlisch@tu-dresden.de