CBOR (RFC 7049)

Concise Binary Object Representation

IETF94 CBOR lightning tutorial
Carsten Bormann, 2015-11-01

CBOR: Agenda

- What is it, and when might I want it?
- How does it work?
- How do I work with it?

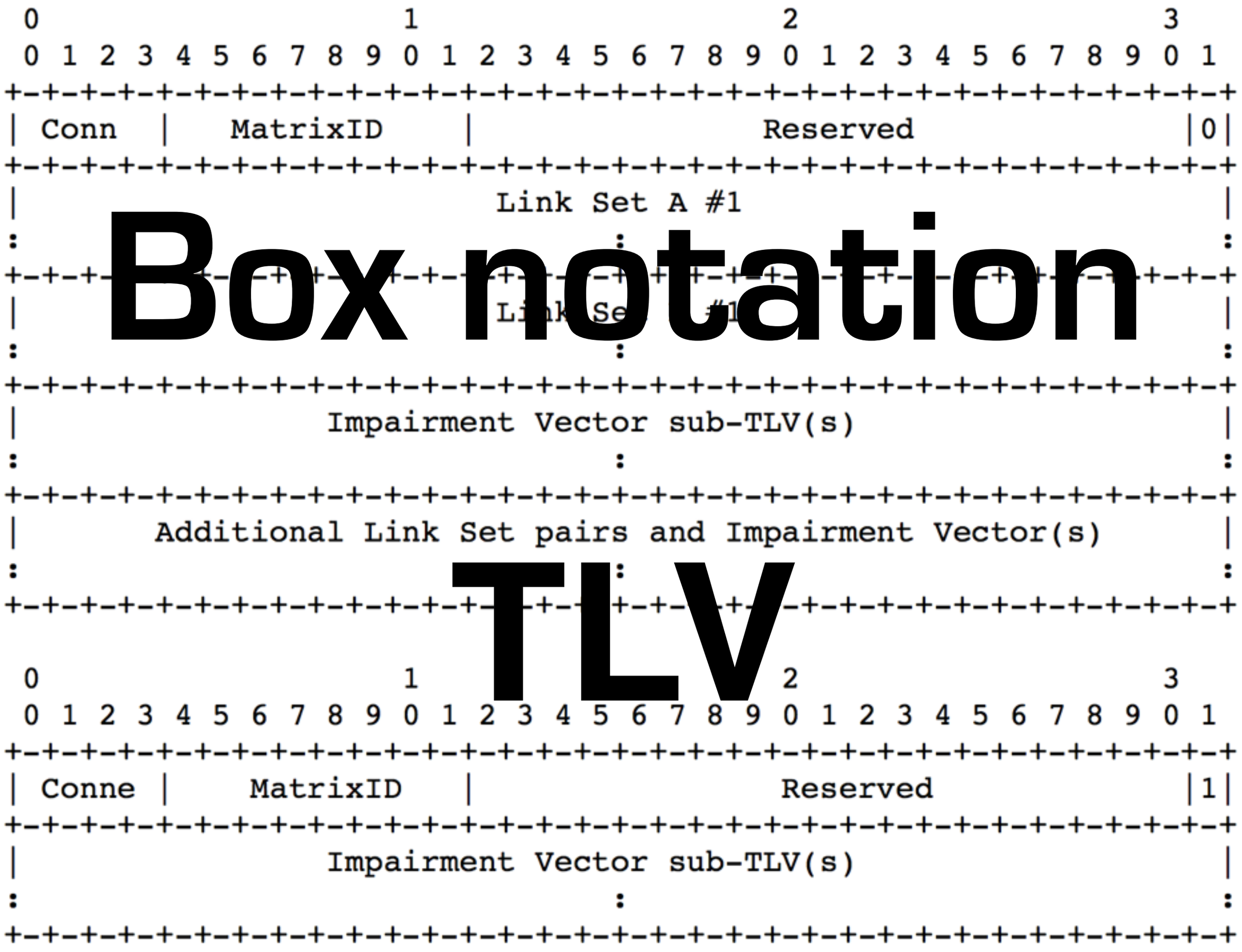
CBOR: Agenda

- What is it, and when might I want it?
- How does it work?
- How do I work with it?

Slide stolen from Douglas Crockford

History of Data Formats

- **Ad Hoc**
- **Database Model**
- **Document Model**
- **Programming Language Model**


Box notation
TLV
0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
Conn
MatrixID
Reserved
0
Link Set A #1
Impairment Vector sub-TLV(s)
Additional Link Set pairs and Impairment Vector(s)
Conne
1

```
S:<?xml version="1.0" encoding="UTF-8" standalone="no"?>
S:<epp xmlns="urn:ietf:params:xml:ns:epp-1.0"
S:     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
S:     xsi:schemaLocation="urn:ietf:params:xml:ns:epp-1.0
S:     epp-1.0.xsd">
S: <response>
S:  <result code="1000">
S:   <msg>Command completed successfully</msg>
S:  </result>
S:  <resData>
S:   <domain:infData
S:    xmlns:domain="urn:ietf:params:xml:ns:domain-1.0"
S:    xsi:schemaLocation="urn:ietf:params:xml:ns:domain-1.0
S:    domain-1.0.xsd">
S:    <domain:name>3.8.0.0.6.9.2.3.6.1.4.4.e164.arpa</domain:name>
S:    <domain:roid>EXAMPLE1-REP</domain:roid>
S:    <domain:status s="ok"/>
S:    <domain:registrant>jd1234</domain:registrant>
S:    <domain:contact type="admin">sh8013</domain:contact>
S:    <domain:contact type="tech">sh8013</domain:contact>
S:    <domain:ns
S:     <domain:ho    ns1.   le.   /do   n:hostObj>
S:     <domain:hos   >ns2.   le.   /do   n:hostObj>
S:    </domain:ns
S:    <domain:ho    ns   exam   e.   /d   ain   ost>
S:    <domain:h     >ns2   xam   e.   /d   ain
S:    <domain:clID>ClientX</domain:clID>
S:    <domain:crID>ClientY</domain:crID>
S:    <domain:crDate>1999-04-03T22:00:00.0Z</domain:crDate>
S:    <domain:upID>ClientX</domain:upID>
S:    <domain:upDate>1999-12-03T09:00:00.0Z</domain:upDate>
S:    <domain:exDate>2005-04-03T22:00:00.0Z</domain:exDate>
S:    <domain:trDate>2000-04-08T09:00:00.0Z</domain:trDate>
S:    <domain:authInfo>
S:     <domain:pw>2fooBAR</domain:pw>
S:    </domain:authInfo>
S:   </domain:infData>
S:  </resData>
S:  <extension>
S:   <e164:infData xmlns:e164="urn:ietf:params:xml:ns:e164epp-1.0"
S:    xsi:schemaLocation="urn:ietf:params:xml:ns:e164epp-1.0
S:    e164epp-1.0.xsd">
S:    <e164:naptr>
S:     <e164:order>10</e164:order>
S:     <e164:pref>100</e164:pref>
S:     <e164:flags>u</e164:flags>
```

XML

```
                    type="idmef:file-permission"
                    use="required" />
</xsd:complexType>

<xsd:complexType name="FileAccess">
  <xsd:sequence>
    <xsd:element name="UserId"
                 type="idmef:UserId" />
    <xsd:element name="permission"
                 type="idmef:Permission"
                 minOccurs="1"
                 maxOccurs="unbounded" />
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Inode">
  <xsd:sequence>
    <xsd:element name="change-time"
                 type="xsd:string"
                 minOccurs="0"
                 x0cc     />
    <xsd:s    en   minO   urs=   max   curs=   >
      <xsd:   t na   "number"
              type   ring   />
      <xsd:   t name="majo   evi   "
              ty   "xsd:s   ing   />
      <xs   leme   name=   dev
              type="xsd:string" />
    </xsd:sequence>
    <xsd:sequence minOccurs="0" maxOccurs="1">
      <xsd:element name="c-major-device"
                   type="xsd:string" />
      <xsd:element name="c-minor-device"
                   type="xsd:string" />
    </xsd:sequence>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Linkage">
  <xsd:choice>
    <xsd:sequence>
      <xsd:element name="name" type="xsd:string" />
      <xsd:element name="path" type="xsd:string" />
    </xsd:sequence>
    <xsd:element name="File" type="idmef:File" />
  </xsd:choice>
  <xsd:attribute name="category"
                 type="idmef:linkage-category"
```

XSD

Slide stolen from Douglas Crockford

JSON

- **JavaScript Object Notation**
- **Minimal**
- **Textual**
- **Subset of JavaScript**

Values

- **Strings**
- **Numbers**
- **Booleans**
- **Objects**
- **Arrays**

- `null`

Array

```
["Sunday", "Monday",
 "Tuesday", "Wednesday",
 "Thursday", "Friday",
 "Saturday"]
```

```
[
   [0, -1, 0],
   [1, 0, 0],
   [0, 0, 1]
]
```

Object


```
{
    "name":        "Jack B. Nimble",
    "at large":    true,
    "grade":       "A",
    "format":  {
        "type":        "rect",
        "width":       1920,
        "height":      1080,
        "interlace":   false,
        "framerate":   24
    }
}
```

~~Object~~ Map

```
{
        "name":        "Jack B. Nimble",
        "at large":    true,
        "grade":       "A",
        "format":      {
                "type":        "rect",
                "width":       1920,
                "height":      1080,
                "interlace":   false,
                "framerate":   24
        }
}
```

Application usage of JSON

- **No schema needed** for parsing
- Representation types:
  bool, number, string, null; **map**, **array**
- Semantics:
  - **struct**: key names in map (vs. **table** use of maps)
  - **record**: position in array (vs. **vector** use of arrays)
- Extensibility: new keys in a key/value struct
  - Ignore what you don’t understand

JSON limitations

- No **binary** data (byte strings)
- Numbers are in **decimal**, some parsing required
- Format requires copying:
  - **Escaping** for strings
  - Base64 for binary
- **No extensibility** (e.g., date format?)
- Interoperability **issues**
  - I-JSON further reduces functionality (RFC 7493)

Data Formats

	Character-based	Concise Binary
Document-Oriented	XML	EXI
Data-Oriented	JSON	???

BSON and friends

- Lots of “binary JSON” proposals
- Often optimized for data at rest, not protocol use (BSON ➔ MongoDB)
- Most are **more** complex than JSON

Why a new binary object format?

- Different design goals from current formats
  – stated up front in the document
- Extremely **small code size**
  – for work on constrained node networks
- Reasonably **compact data size**
  – but no compression or even bit-fiddling
- Useful to any protocol or application that *likes* the design goals

Concise Binary Object Representation (CBOR)

“Sea Boar”

Data Formats

	Character-based	Concise Binary
Document-Oriented	XML	EXI
Data-Oriented	JSON	CBOR

Design goals (1 of 2)

1. unambiguously encode most **common data formats** (such as JSON-like data) used in Internet standards

2. **compact implementation** possible for encoder and decoder

3. able to parse **without a schema description**.

Design goals (2 of 2)

4. Serialization reasonably **compact**, but data compactness **secondary to** implementation compactness
5. applicable to both **constrained nodes** and **high-volume applications**
6. support all **JSON** data types, conversion to and from JSON
7. **extensible**, with the extended data being able to be parsed by earlier parsers

2013-09-13: CBOR RFC

- “Concise Binary Object Representation”:
  JSON equivalent for constrained nodes
    - start from JSON data model (no schema needed)
    - add binary data, extensibility (“tags”)
    - concise binary encoding (byte-oriented, counting objects)
    - add diagnostic notation
- Done without a WG (with APPSAWG support)

Overview | Specification | Implementations | Tools

http://cbor.io

CBOR

RFC 7049 Concise Binary Object Representation

"The Concise Binary Object Representation (CBOR) is a data format whose design goals include the possibility of extremely small code size, fairly small message size, and extensibility without the need for version negotiation."

JSON data model

CBOR is based on the wildly successful JSON data model: numbers, strings, arrays, maps (called objects in JSON), and a few values such as false, true, and null.

No Schema needed

Embracing binary

Some applications that would like to use JSON need to transport binary data, such as encryption keys, graphic data, or sensor values. In JSON, these data need to be encoded (usually in base64 format), adding complexity and bulk.

Concise encoding

Stable format

CBOR is defined in an Internet Standards Document, RFC 7049. The format has been designed to be stable for decades.

Extensible

To be able grow with its applications and to

CBOR: Agenda

- What is it, and when might I want it?
- How does it work?
- How do I work with it?

CBOR vs. “binary JSONs”

- Encoding [1, [2, 3]]:

	compact	stream
ASN.1 BER*	30 0b 02 01 01 30 06 02 01 02 02 01 03	30 80 02 01 01 30 06 02 01 02 02 01 03 00 00
MessagePack	92 01 92 02 03	
BSON	22 00 00 00 10 30 00 01 00 00 00 04 31 00 13 00 00 00 10 30 00 02 00 00 00 10 31 00 03 00 00 00 00 00	
UBJSON	61 02 42 01 61 02 42 02 42 03	61 ff 42 01 61 02 42 02 42 03 45*
CBOR	82 01 82 02 03	9f 01 82 02 03 ff

Very quick overview of the format

- Initial byte: **major type** (3 bits) and **additional information** (5 bits: immediate value or length information)
- Eight major types:
  - unsigned (0) and negative (1) **integers**
  - **byte** strings (2), **UTF-8** strings (3)
  - **arrays** (4), **maps** (5)
  - optional **tagging** (6) and **simple types** (7) (floating point, Booleans, etc.)

Additional information

- 5 bits
  - 0..23: immediate value
  - 24..27: 1, 2, 4, 8 bytes value follow
  - *28..30: reserved*
  - 31: indefinite length
    - terminated only by 0xFF in place of data item

- Generates unsigned integer:
  - **Value** for mt 0, 1 (unsigned/neg integers), 7 (“simple”)
  - **Length** (in bytes) for mt 2, 3 (byte/text strings)
  - **Count** (in items) for mt 4, 5 (array, map)
  - **Tag** value for mt 6

Major types 6 and 7

- mt 7:
  - **special** values for ai = 0..24
    - false, true, null, undef
    - IANA registry for more
  - ai = 25, 26, 27: IEEE **floats**
    - in 16 (“half”), 32 (“single”), and 64 (“double”) bits
- mt 6: semantic **tagging** for things like dates, arbitrary-length bignums, and decimal fractions

Tags

- A Tag contains **one** data item
- 0: RFC 3339 (~ ISO 8601) **text string** date/time
- 1: UNIX time (**number** relative to 1970-01-01)
- 2/3: bignum (**byte string** encodes unsigned)
- 4: **[exp, mant]** (decimal fraction)
- 5: **[exp, mant]** (binary fraction, “bigfloat”)
- 21..23: expected conversion of **byte string**
- 24: nested CBOR data item in **byte string**
- 32…: URI, base64[url], regexp, mime (**text strings**)

New Tags

- Anyone can register a tag (IANA)
  - 0..23: Standards action
  - 24..255: Specification required
  - 256..18446744073709551615: FCFS
- 25/256: stringref for simple compression
- 28/29: value sharing (beyond trees)
- 26/27: constructed object (Perl/generic)
- 22098: Perl reference (“indirection”)

Examples

- Lots of examples in RFC (making use of JSON–like “diagnostic notation”)
- 0 ➔ 0x00, 1 ➔ 0x01, 23 ➔ 0x17, 24 ➔ 0x1818
- 100 ➔ 0x1864, 1000 ➔ 0x1903e8, 1000000 ➔ 0x1a000f4240
- 18446744073709551615 ➔ 0x1bffffffffffffff, 18446744073709551616 ➔ 0xc249010000000000000000
- –1 ➔ 0x20, –10 ➔ 0x29, –100 ➔ 0x3863, –1000 ➔ 0x3903e7
- 1.0 ➔ 0xf93c00, 1.1 ➔ 0xfb3ff199999999999a, 1.5 ➔ 0xf93e00
- Infinity ➔ 0xf97c00, NaN ➔ 0xf97e00, –Infinity ➔ 0xf9fc00
- false ➔ 0xf4, true ➔ 0xf5, null ➔ 0xf6
- h'' ➔ 0x40, h'01020304' ➔ 0x4401020304
- "" ➔ 0x60, ”a" ➔ 0x6161, ”IETF" ➔ 0x6449455446
- [] ➔ 0x80, [1, 2, 3] ➔ 0x83010203, [1, [2, 3], [4, 5]] ➔ 0x830182020382040 5
- {} ➔ 0xa0, {1: 2, 3: 4} ➔ 0xa201020304, {"a": 1, "b": [2, 3]} ➔ 0xa26161016162820203

CBOR: Agenda

- What is it, and when might I want it?
- How does it work?
- How do I work with it?

http://cbor.me: CBOR playground

- Convert back and forth between **diagnostic notation** (~JSON) and binary encoding

CBOR

Diagnostic →

```
[1, [2, 3]]
```

← 5 Bytes

```
82          # array(2)
   01       # unsigned(1)
   82       # array(2)
      02    # unsigned(2)
      03    # unsigned(3)
```

Implementations

- Parsing/generating CBOR easier than interfacing with application
  - Minimal implementation: 822 bytes of ARM code
- Different integration models, different languages
- > 25 implementations (after first two years)

http://cbor.io

CBOR and CDDL

- CBOR takeup within IETF is increasing.
  How to write specs?

- CDDL: CBOR **Data Definition** Language
  https://tools.ietf.org/html/draft-greevenbosch-appsawg-cbor-cddl-07
  - The best of ABNF, Relax-NG, JSON Content Rules
  - Rough tool available: **gem install cddl**
    - **Generate** example instances (CBOR or JSON)
    - **Check** instances against the definition

How **RFC 7071** would have looked like in CDDL

```
reputation-object = {              ; This is a map (JSON object)
  application: text                ;  text string (vs. binary)
  reputons: [* reputon]            ;  Array of 0-∞ reputons
}

reputon = {                        ; Another map (JSON object)
  rater: text
  assertion: text
  rated: text
  rating: float16                  ; OK, float16 is a CBORism
  ? confidence: float16            ; optional…
  ? normal-rating: float16
  ? sample-size: uint              ; unsigned integer
  ? generated: uint
  ? expires: uint
  * text => any                    ; 0-∞, express extensibility
}
```

GRASP

- Generic Autonomic Signaling Protocol (GRASP)
- For once, try not to invent another TLV format: just use CBOR
- Messages are arrays, with type, id, option:

```
message /= [MESSAGE_TYPE, session-id, *option]
MESSAGE_TYPE = 123 ; a defined constant
session-id = 0..16777215
; option is one of the options defined below
```

- Options are arrays, again:

```
option /= waiting-time-option
waiting-time-option =
                        [O_WAITING, waiting-time]
O_WAITING = 456 ; a defined constant
waiting-time = 0..4294967295 ; in milliseconds
```

Where from here?

- RFC 7049
- http://cbor.io
- cbor@ietf.org
- http://tools.ietf.org/html/cddl