CBOR (RFC 7049) Specification Status

draft-ietf-cbor-7049bis-00
Carsten Bormann 2017-07-17

Sat/Sun: WISHI

Workshop on IoT Semantic/Hypermedia Interoperability

- T2TRG organized a meeting of experts working with SDOs standardizing data formats for IoT interchange (IPSO, OMA [LWM2M], iot.schema.org, W3C WoT, OCF, OneM2M, Fairhair, Haystack)
- Besides JSON, CBOR now clearly is a serialization of choice
  - Sometimes just to interchange data authored as JSON
- Some SDOs have tried working with “JSON Schema” for their modeling and are now moving away from that

Moving CBOR to Standard

- [✔] Fix errata
- [_] Check for opportunities for editorial improvement
- [_] Revisit POLS violations (Canonical ordering), possibly improve editorially
- [!] Revisit IANA considerations
- [_] Generate Request for Reclassification

CBOR registries

- CBOR Tags
  - 0–23: Stds action
    (9 taken in RFC 7049,
    3 in RFC 8152) 50 %
  - 24–255: Spec required
    (19 taken) 8.2 %
  - 256 to 18446744073709551615: FCFS
    (10 taken)

- CBOR Simple Values
  - 0–23: Stds action
    (4 taken in RFC 7049)
  - 32–255: Spec required
- (No activity so far. But…)

Maybe get stricter?

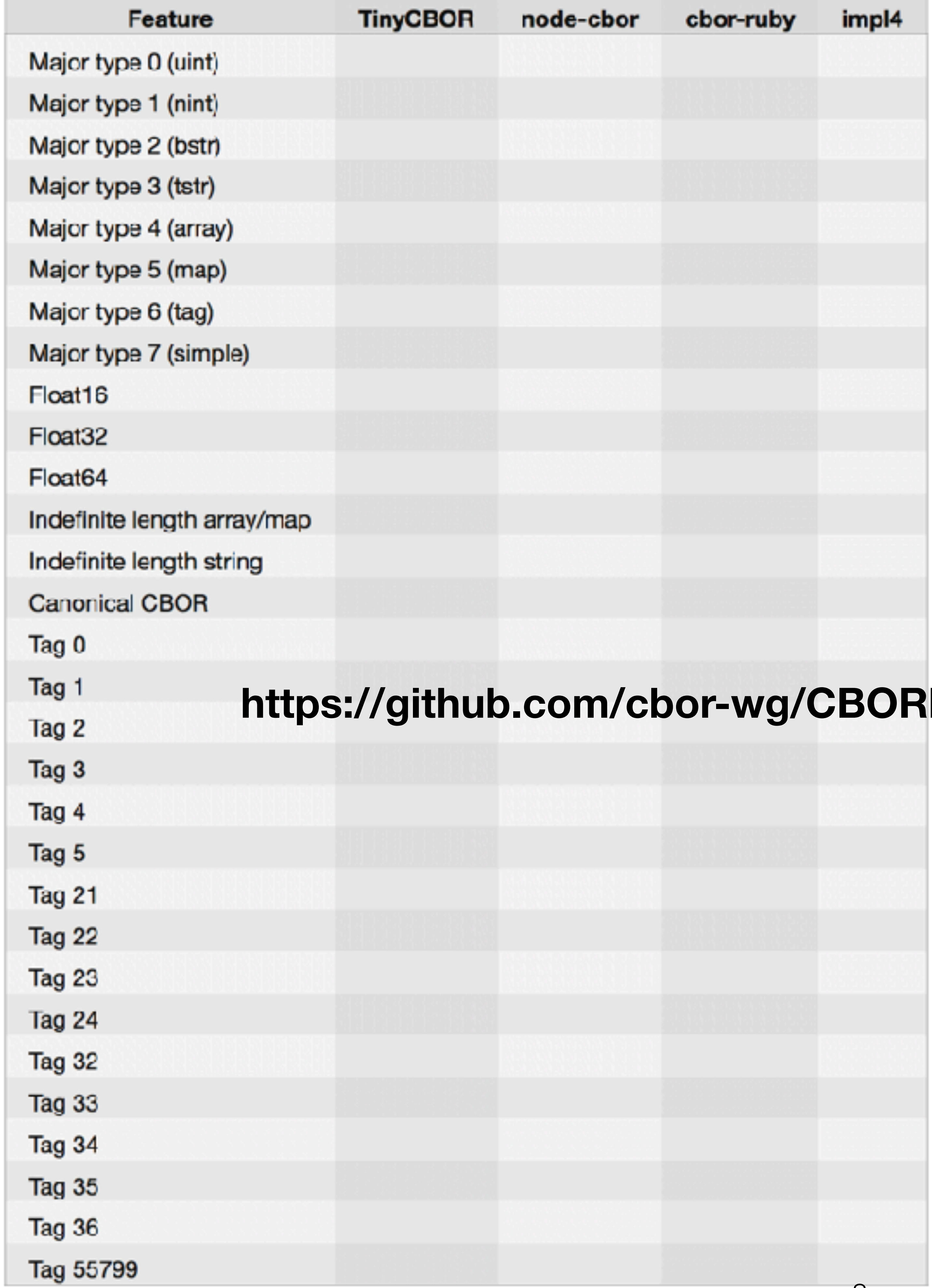
- Could move 2-byte ranges (24/32–255) to Expert Review
- Could give advice that only tags get it that are worth it, i.e.:
  - for a data item that is (can be) short enough that a shorter tag makes a difference
  - is not just a very niche application
- E.g., should UUID (16 bytes) have got a 2-byte tag?
- Add circuit breaker for when we get near 50 % usage?
- Take part of the 3-byte range (256..65535) to Spec Required?

Request for Reclassification (See RFC 6410)

- (1) There are at least **two independent interoperating implementations** with **widespread deployment** and successful **operational experience**.
- (2) There are **no errata** against the specification that would cause a new implementation to fail to interoperate with deployed ones. **n. a.**
- (3) There are **no unused features** in the specification that **greatly increase implementation complexity**.
- (4) If the technology required to implement the specification requires patented or otherwise controlled technology, then the set of implementations must demonstrate at least two independent, separate and successful uses of the licensing process. **n. a.**

Unused Features? Check with Implementation Matrix

- cbor.io shows 40 implementations today
    - Some have known bugs and issues — maybe worth an effort to clean them up?
- We probably should pick a small number of them (columns)
- What are the rows?
    - Major types (easy)
    - Features (indefinite array/map, string)
    - Float variants
    - Tags (really each of them)
      ➔ move rest to separate document that stays on PS?
    - Canonical CBOR



Feature	TinyCBOR	node-cbor	cbor-ruby	impl4
Major type 0 (uint)				
Major type 1 (nint)				
Major type 2 (bstr)				
Major type 3 (tstr)				
Major type 4 (array)				
Major type 5 (map)				
Major type 6 (tag)				
Major type 7 (simple)				
Float16				
Float32				
Float64				
Indefinite length array/map				
Indefinite length string				
Canonical CBOR				
Tag 0				
Tag 1				
Tag 2				
Tag 3				
Tag 4				
Tag 5				
Tag 21				
Tag 22				
Tag 23				
Tag 24				
Tag 32				
Tag 33				
Tag 34				
Tag 35				
Tag 36				
Tag 55799				

https://github.com/cbor-wg/CBORbis/wiki/Implementation-matrix

Non-STD tags?

- Could move some of RFC 7049’s tags to PS document
- Could also document really good ones that are registered only (possibly get their specs as contributions)
- Publish with 7049bis?

Which of the 40?

At least **two:**

- **independent**
- **interoperating**
- **widespread deployment**
- successful **operational experience**.

CDDL Specification Status

draft-greevenbosch-appsawg-cbor-cddl-11

Carsten Bormann 2017-07-17

Adoption call running

- If disagree with Chicago in-room consensus to adopt:
- Comment until 2017-07-19

Status?

- Objective: Be referenceable from standards track documents for their data definitions
- Go for Standards-Track
- Go for Informational, but get a downref allowance

Name change

- Concise data definition language
  (was: CBOR data definition language)
- Do *not* give up the now well-known initialism CDDL
  - Conflict with Sun’s “Common Development and Distribution License” is in a different (but related) field
  - But CDDL is pretty unique google-wise otherwise

New in -11: unwrap

```
basic-header-group = (
  field1: int,
  field2: text,
)

basic-header = {
  basic-header-group,
}

advanced-header = {
  basic-header-group,
  field3: bytes,
  field4: number, ; as in the tagged
                  ; type "time"
}
```

```
basic-header = {
  field1: int,
  field2: text,
}

advanced-header = {
  ~basic-header,
  field3: bytes,
  field4: ~time,
}
```

ABNF changes

- Grammar was unnecessarily (?) restrictive about requiring parentheses
    - Mostly affected Control operator
- Made sure all non-C0 Unicode characters can be used in strings and comments
- And of course the ~ (unwrap) operator

Appendix E.1: Use with JSON

- Concentrated examples, added text for using CDDL with JSON proper
- Exposes weirdness about floating point vs. integer numbers in constants (different in CBOR, same in JSON)
    - Probably requires a few more statements
- Reflects decision to leave representation details out of CDDL
    - Might want to add global representation flags (e.g., indefinite array/map, string; float 16/32/64?)

Appendix G: Extended Diagnostic Notation

- Added << … >> to be able to nest encoded CBOR (sequences)
- <<"foo", null>>   is now synonym of   h’63666F6F F6’
- (Should this be defined here or be a separate addition to RFC7049bis?)

Editorial changes in -11

- “Annotations” (-10) are now “Controls” (-11)

  ```
  type1 .ctlop type2
  ```

  - Editorial change in description of CDDL only — no changes in CDDL specs
  - Controls continue to be most important extension point of CDDL
- Move Generics from *Nursery* to main body
- Thinned examples (more useful ones remain)
- Moved JSON

Implementation Status (CDDL tool)

- << … >> implemented in cbor-diag parser
  - Easy to parse, but hard to generate automatically
- Unwrap (~) and ABNF changes await prior refactoring of CDDL tool (old parser is creaking)
  - In progress…
  - Might lead to draft about “JSON format” for CDDL

One implementation is not enough

- What does it even mean to “implement CDDL”?
- CDDL tool is used today to
  - Check CDDL specs by generating random examples from them
  - Check CBOR examples against their CDDL spec
  - Generate annotated diagnostic notation from CBOR
  - Extract C #defines and similar from CDDL spec
- Many other opportunities for processing CDDL (pretty-printing the spec itself, checking some application-specific consistency condition, management of number spaces, …)
- Support ecosystem by defining JSON formats for CDDL tool interoperation (draft forthcoming)
- Looking for additional implementers!

Next technical round

- Make sure the numeric system is well-defined (float vs. integer discussion, relationship with JSON numbers)
- Possibly address serialization constraints
  - Probably best to keep them well separated from the data model defined by the constructive grammar (both technically and editorially)
- Help CDDL tools to emit better error messages?

Next editorial round

- Terminology walkthrough
  - E.g.: CDDL groups are composed of fields,
    CBOR arrays of elements, CBOR maps of members
- Introductory material
  - E.g., explain scopes of YANG vs. CDDL (YANG is for data at rest, creating an API; vs. CDDL for data in motion)
- Editorial comments very welcome
  - For small comments: Easy to make a github comment or open a github issue!

Array tags

draft-jroatch-cbor-tags-05

Carsten Bormann 2017-07-17

Pretty much cooked

- 24 tags for representing homogeneous arrays within byte strings (4 lengths x uint/sint/float x BE/LE [yes!]),
    - modeled after JavaScript TypedArray (which was in turn taken from Graphics standards)
- 1 tag for turning a one-dimensional array into a multi-dimensional one
- 1 tag for declaring an array as homogeneous (i.e., not something like `[3, “foo”, << 1, 2, 3 >>]`)

Remaining discussion: Which Tag length?

- Currently proposing to go for 2-byte tags throughout
  - Would consume 11.2 % of 2-byte tag space at once
- Justified: These are quite fundamental additions to the CBOR type system and will be widely used
- Arrays *can* be long, but not all are.

🚧 Unchartered work beyond here 🚨

Assess interest in these proposals

Extended Time tags

draft-bormann-cbor-time-tag-01

Carsten Bormann 2017-07-17

Requests from various communities

- Haskell: provide for seconds + picoseconds timestamps
- Security automation: Provide more intent information
- IoT applications: Timescales beyond Posix-mangled UTC

Approach

- Do not try to expand tags 0 (RFC time) or 1 (Posix time)
- Instead, define new tag 1001 (allocated out of FCFS)
    - (Add tags 1002 for duration and 1003 for period, to be defined later…)
- Define this as a map which can contain several pieces of information

Member keys

- 1: Posix time in seconds
  - Alternatively: 4 for Decimal Fraction, 5 for Bigfloat
- –3, –6, –9, –12, –15, –18: Add μs/ns/ps/fs/as to that
- To do:
  - Define keys for time scales (e.g., TAI, unmangled UTC?)
  - Define keys for intent information (e.g., time zones
- Examples for those were in –00, removed for stable subset

Trivia

- This WG session started at Posix time 1500299400

🎉🎊🍻🎁🍾🥂

Template Tag

draft-bormann-lpwan-cbor-template-00

Carsten Bormann 2017-07-17

Templates: data structures with placeholder variables

- Example: Mix template
```
{ "name": "Carsten Bormann",
  "place": 42(0) }
```

- with »set variable 0 to “Bremen”«

- and get
```
{ "name": "Carsten Bormann",
  "place": "Bremen" }
```

- Potential customer: LWPAN static header compression

Packed Tag

draft-bormann-cbor-packed-00

Carsten Bormann 2017-07-17

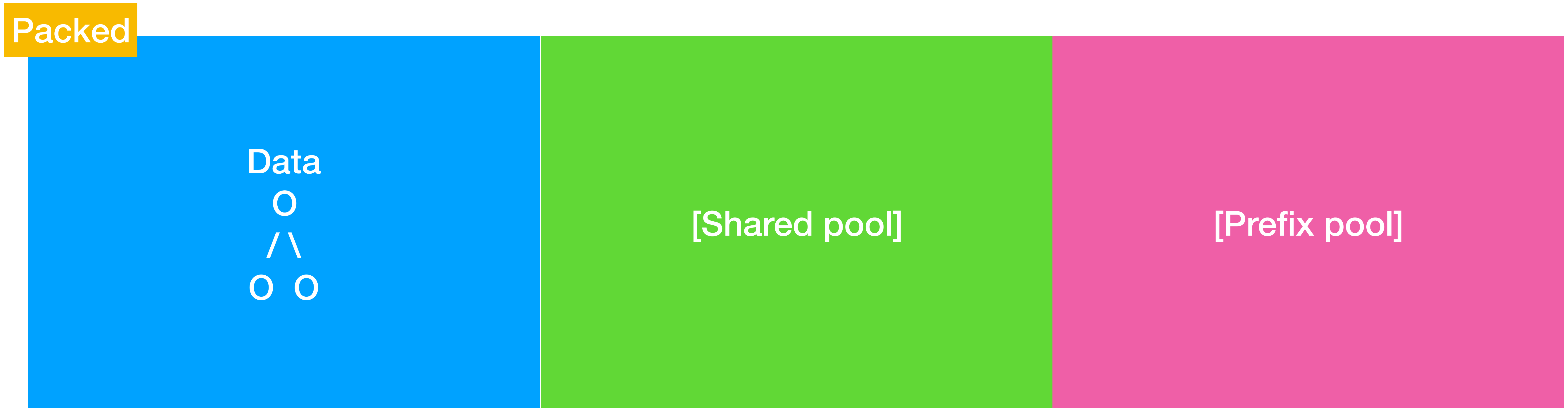
Background: W3C TD

- LSON-LD document
- Lots of shared structure
- Lots of repeated strings
- Lots of strings with mostly the same prefixes

Data compression is great, but…

- Data compression (e.g., RFC 1951 deflate) can reduce TDs to 1/4 of their size
- Compressed form not processable
  - Need to decompress (copy!) before use

Packed Tag



- Subtrees with more than one occurrence go to shared pool — referenced from data
- Common string prefixes go to prefix pool — referenced from reference-tagged string
- Compression + decompression: 205 lines of code

Creating shared references

- 1-byte: Simple(0)..Simple(15)
- 2-byte: Simple(144)..Simple(255)
- 2+-byte TagN(int), (folded [0, -1, 1, -2]…)
- 16 1-byte, 160 2-byte, ~ unlimited
- Eats 80 % of 1-byte simple, 50 % of 2-byte simple,
  1 1-byte tag — all only within context of Packed Tag

Creating prefix references

- 2-byte: TagN(suffix), $224 \leq N \leq 255$ (32 total)
- 3-byte: TagN(suffix), $512 \leq N \leq 1023$ (512 total)
- Eats 12.5 % of 2-byte Tag, 0.4 % of 3-byte Tag — all only within context of Packed Tag
- Artificially limited to 564 “best” prefixes

Experiment

`wot-thing-description/test-bed/data/plugfest/2017-05-osaka/MyLED_f.jsonld`

- JSON file: 3116
- **JSON no whitespace: 1447**
  - deflate: 323, lz4: 415, lz4hc: 411
- **CBOR: 1210**
  - deflate: 325, lz4: 416, lz4hc: 404
- CBOR packed (semantic sharing only): 793
- CBOR packed (prefix compression, too): 564

Conclusion

- Packing (exploiting structural sharing)
  - maintains processability
  - saves ~ 1/3 (implementation not yet complete)
- Prefix sharing helps with URLs, another 20 %
  - but reduces processability
- Could further improve by adding static dictionary
  - In the example: 119 bytes of mostly static data:

```
["name","@type","links","application/json","outputData","mediaType","href",
 {"valueType":{"type":"number"}},["Property"],"writable","valueType","type"]
```