Stumbling stone of cbor-packed: resource sharing vs usurpation

In some slides (or by voice) Carsten said about resource allocation so that every app protocol do not have to allocate again.

But what about specifications?

— Case study: CBOR-LD issue - they wanted to allocate 256 tags range and then continue with array from 257'th

— this triggers an idea...

Possible Solomon solution: sharing range between specs

Especially actual for simple values - currently [cbor-packed] and CBAR compete for them. But observe both specs have it **inside some namespace tag** (e.g. 113 or 10).

What if separate RFC registers simple(0..16) for "packing and templating purposes, application MUST define their exact usge or inherit from standard of choice like inefficient but readiliy available cbor-packed or later standard like CBAPT" ?

— Then `A` variable **could be raised to 16 again!**

— Not so straightforward for tags (must define tagged item), however...

Interoperation of DNS-CBOR and cbor-packed still unclear

[draft-lenders-dns-cbor-13] quote:

```
TBD113(
 TBD28259(
   [
     ["org", 42],
     [
       "www", "example", simple(5) / expands to "org" /,
       ["svc", simple(0) / expands to ["www", "example", "org"] /],
       simple(5),  / expands to "org" /
       simple(1),  / expands to ["www", "example", "org"] /
       simple(6),  / expands to 42 /
       simple(3),  / expands to ["svc", ["www", "example", "org"]] /
       simple(6)   / expands to 42 /
     ]
   ]
 )
)
```

Interoperation of DNS-CBOR and cbor-packed still unclear

— Why `TBD113(TBD28259(obj))` if Tag 113 needs >1 argument?

— If Tag 28259 passes it's expansion to outer tag 113, why first entries (`["org", 42]`) become last entries?

— If it is for one-pass algorithm of either array, then starting from second (`["www",...]`) array - how it knows what `simple(5)` equals to, before finishing processing?

— ...yet it must know this to finish - because `simple(5)` could have been array (not just plain scalar `"org"`) !

OK, let's try it some as-understood way: original

Last example from draft-13 A.2 after some cleaning for `diag2cbor.rb`, 155 bytes:

```
00000000  84 83 67 65 78 61 6d 70  6c 65 63 6f 72 67 0c 81  |..gexamplecorg..|
00000010  84 19 0e 10 65 5f 63 6f  61 70 64 5f 75 64 70 65  |....e_coapd_udpe|
00000020  6c 6f 63 61 6c 82 84 19  0e 10 02 63 6e 73 31 e0  |local......cns1.|
00000030  84 19 0e 10 02 63 6e 73  32 e0 84 84 e2 19 0e 10  |.....cns2.......|
00000040  18 1c 50 20 01 0d b8 00  00 00 00 00 00 00 00 00  |..P ............|
00000050  00 00 01 84 e2 19 0e 10  18 1c 50 20 01 0d b8 00  |..........P ....|
00000060  00 00 00 00 00 00 00 00  00 00 02 84 e5 19 0e 10  |................|
00000070  18 1c 50 20 01 0d b8 00  00 00 00 00 00 00 00 00  |..P ............|
00000080  00 00 35 84 e6 19 0e 10  18 1c 50 20 01 0d b8 00  |..5.......P ....|
00000090  00 00 00 00 00 00 00 00  00 35 35                 |.........55|
```

OK, let's try it some as-understood way: cbor-packed

Apply prefix on IPv6 addresses, 129 bytes:

```
$ diag2cbor.rb < draft-lenders-dns-cbor-13_A.2_last.packed.js | hd
00000000  d8 71 82 81 4e 20 01 0d  b8 00 00 00 00 00 00 00  |.q..N ..........|
00000010  00 00 00 d9 6e 63 84 83  67 65 78 61 6d 70 6c 65  |....nc..gexample|
00000020  63 6f 72 67 0c 81 84 19  0e 10 65 5f 63 6f 61 70  |corg......e_coap|
00000030  64 5f 75 64 70 65 6c 6f  63 61 6c 82 84 19 0e 10  |d_udpelocal.....|
00000040  02 63 6e 73 31 e0 84 19  0e 10 02 63 6e 73 32 e0  |.cns1......cns2.|
00000050  84 84 e2 19 0e 10 18 1c  d8 e6 42 00 01 84 e2 19  |..........B.....|
00000060  0e 10 18 1c d8 e6 42 00  02 84 e5 19 0e 10 18 1c  |......B.........|
00000070  d8 e6 42 00 35 84 e6 19  0e 10 18 1c d8 e6 42 35  |..B.5.........B5|
00000080  35                                                |5|
$ grep '[0-9](' draft-lenders-dns-cbor-13_A.2_last.packed.js
113([ [h'20010db800000000000000000000'],
 28259([
       3600, 28, 230(h'0001')
       3600, 28, 230(h'0002')
       3600, 28, 230(h'0035')
       3600, 28, 230(h'3535')
```

OK, let's try it some as-understood way: CBAR "purist"

Same, using single dictionary (table) for all strings in setup, 135 bytes:

```
10([ simple(0),
 h'20010db80000000000000000000000',
 'nc',
 [
   ["example", "org"],
   ["_coap", "_udp", "local"],
   ["ns1", 0],
   ["ns2", 0]
 ],
 h'...' /rump using 1D as atom 0/
])
```

CBAR two-phase: function nesting

Fixed parse-aliases not used, generic call - 128 bytes (could be 126):

```
10([
 'nc',
 10([
 simple(0),
   h'20010db800000000000000000000',
   h'/rump has encoding from "original" above, with
      20010db800000000000000000000 replaced with 1d/'
 ])
])
```

— PRO: 'nc' function receives same input as in current [dns-cbor] - same implementation code

— PRO: less bytes than cbor-packed, especially when not as in this example but if you will need to compress in middle of strings (e.g. runs of 00's in different global IPv6 subnets)

— CON: not a "purist correct right way" with a single atom dictionary (table)

— CON: more complicated implementation of in-place access