The Concise Binary Object Representation (CBOR) [**bormannConciseBinaryObject2020**], along with Concise Data Description Language (CDDL) [**birkholzConciseDataDefinition2019**] is another data description language and encoding format. While there are a large number of similarities between CBOR with CDDL and protobuf, some important differences persist.

Perhaps the most important difference is that CBOR is a self-describing format, meaning that any valid blob of CBOR can be parsed without a schema to parse against. The header of each CBOR field completely define the type and value of the field. On the other hand, the exact value of a variable length integer field in a protobuf blob can't be exactly known until the schema is consulted to know if the field is a `sint32` or `uint64`. It can be viewed that CBOR splits the parsing process into two steps. First, the blob is parsed into a record, and then it is validated against the specification defined in CDDL. Another difference is that protobuf (at least `proto3`) has no way to mark a field as required while fields in CDDL are required by default.

Given that a valid CBOR blob can be parsed independently of any schema, the base unit of consideration between CBOR and protobuf is different. We can start with the parsed CBOR value while protobuf forces considerations about the encoding and schema together. Another difference is that CDDL can describe four different types of data, defined as a vector, record, table or struct [**birkholzConciseDataDefinition2019**], while Protobuf messages are always a struct in CDDL terminology.

It is also worth noting the CDDL is much more prescriptive. Using what the CDDL specification calls "controls", it is possible to describe an integer which is within a specific, arbitrary range or an array (protobuf `repeated` field) of a specific length [**birkholzConciseDataDefinition2019**]. While this feels at first glance like it would open up even more questions, running with the integer example, with only one integer encoding it is clear that the range can only grow. The CDDL schema has to start restrictive to be updated to be less restrictive since it can never shrink. In this sense, CDDL is more rigid than protobuf, without the flexibility to transition between types of integer in protobuf, stripping out a lot of the rules in Section **??**. In that sense, protobuf offers a lot of interesting features, like a form of post-processing as integer types change.

0.1 CBOR Headers

The CBOR data item (since CBOR values don't always exist as fields in a message like Protobuf values) also have a header, which contains the major type and argument. CBOR major types are described in Table 1.

Major Type	Meaning
0	Non-negative Integer
1	Negative Integer
2	Byte String
3	Text String (UTF-8)
4	Array
5	Map
6	Tagged Value
7	Floating Point Values & Simple Values

Table 1: CBOR Major Types

The major type is encoded in the upper 3 bits of the first byte of the header, with the lower 5 bits being used for "additional information". This additional information is used to load the argument to the major type, which is typically an unsigned integer. Small argument values may be encoded directly into these 5 bits, or then may indicate that the next 1, 2, 4 or 8 bytes hold the argument. The argument itself may be used to hold the length of a string, the number of elements or key-value pairs in an array or map, or the tag number of a tagged value. Integer values are encoded directly into the argument slot. After the header is the rest of the payload.

0.2 CBOR Value Relation

CBOR itself is designed to be self-describing, so any valid CBOR value can be parsed without referencing a schema definition. This doesn't mean that CBOR is deterministic, with only one way to represent a value, it certainly is not. The value relation, as described for Protobuf in Section **??** only has a trivial relation with a reflexivity rule (and I guess a transitivity rule, but that doesn't expand the domain of the relation at all).

However, just because CBOR doesn't have the same flexibility with the value relation doesn't mean that CDDL also doesn't. Remember, CBOR and CDDL draw a distinct line between the binary wire format and the descriptor language.