POSTER: Compact Linked Data for Constrained Web of Things Using CBOR-LD

Satendra Raj, Aryansh Tripathi, Kamal Singh

POSTER: Compact Linked Data for Constrained Web of Things Using CBOR-LD

Satendra Raj[1], Aryansh Tripathi[1], and Kamal Singh[2]
[1]*Indian Institute of Technology, BHU, India*
[2]*Univ Jean Monnet, IOGS, CNRS, UMR 5516, LaHC, F - 42023 Saint-Etienne, France*
Email: [1]satendra.raj.civ19@iitbhu.ac.in, aryansh.tripathi.civ20@itbhu.ac.in, [2]kamal.singh@univ-st-etienne.fr

***Abstract*—Web of Things (WoT) brings web technologies and knowledge graphs to Internet of Things. JSON-LD, with its popularity in representing and exchanging structured data on the web, can be a suitable data format for WoT. However, its verbose nature can pose challenges for constrained IoT devices with limited bandwidth, and memory. In this paper, we study CBOR-LD scheme for encoding JSON-LD to a lightweight binary format CBOR (Compact Binary Object Representation). We also study possible optimisations such as using a custom dictionary. We provide a C library and evaluate it on several examples. Results demonstrate that our approach provides savings up to 94% in terms of network overhead for IoT devices.**

I. Introduction

Web of Things (WoT) evolves the Internet of Things (IoT) paradigm by adding web technologies and knowledge graphs. WoT can use JSON-LD [1], which is a JSON-based format for representing and exchanging structured data on the web. JSON-LD offers significant benefits for IoT or WoT domain including semantic interoperability, seamless data integration from heterogeneous sources, contextual information, and linked data. Through semantic annotations, JSON-LD enables IoT devices and systems from various manufacturers to interpret data consistently, promoting interoperability. Additionally, JSON-LD's flexible and standardized format facilitates the integration of diverse data sources such as sensors, actuators, and control systems. Contextual information embedded alongside data in JSON-LD enriches IoT applications by adding additional meaning and significance to data generated. Lastly, JSON-LD's linked data property enables the discovery and integration of distributed data sources, empowering more complex applications and analyses. It enables flexible declarative search as data takes the form of a graph. Some illustrative examples of queries could be to search the name of rooms whose light is ON in a building, search for rooms whose electric consumption is more than a threshold, etc. Such graphs can also allow for inference of new facts by using a reasoner.

However, one disadvantage of JSON-LD is it's verbose format that can be a significant overhead for constrained WoT objects. Some of the constrained objects have limited bandwidth, small amount of memory, limited computational capabilities and run on a battery.

The work by Satendra Raj and Aryansh Tripathi was accomplished during their internship stay in Laboratory Hubert Curien, Saint-Etienne, France.

This paper contributes by proposing an open source C library[1] to first encode JSON-LD terms to integers based on a dictionary and then to convert it to CBOR format. CBOR is a compact binary format that supports various data types for efficient memory usage. It is designed to be lightweight, easy to parse, and optimized for low-power devices. CBOR supports nested structures, schema-based validation and facilitates readability through stream processing making it suitable for representing complex data in a compact format. C language was chosen to make it more suitable for embedded devices. Our implementation is based on an in-progress CBOR-LD standardisation in World Wide Web Consortium (W3C)[2]. In this paper, we also explore optimisation possibilities such as adding custom terms to the dictionary used for encoding. Our results show that these optimisations are able to obtain further savings in terms of bandwidth.

This paper is organised as follows. Section II discusses related work. Section III details the encoding scheme from JSON-LD to CBOR-LD, Section IV performs evaluation on some example data and Section V concludes the paper.

II. Related work

HDT (Header, Dictionary, and Triples) [2] represents the state of the art on compacting the RDF (Resource Description Framework) data. RDF [3] is another popular format for representing linked data. The authors in [4] introduced a novel in-memory RDF dictionary using optimized Trie structures to efficiently compress common prefixes. The work in [5] proposes a compact representation of semantic sensor data, in RDF format, by sending the repeated measurement values only once.

For JSON-LD, some previous works [6] have also studied compact representations for JSON-LD as well as RDF data. The work in [6] compared different approaches and proposed JSON-LD compaction by mapping resource identifiers to shorter strings. They found that such compaction coupled CBOR, can lead to compaction ratios around 50-60% compared to HDT. More recently, the work on using CBOR to compact the JSON-LD data has been ongoing in W3C and it is the focus of study in this paper.

[1]https://gitlab.com/coswot/cborld-c
[2]https://json-ld.github.io/cbor-ld-spec/

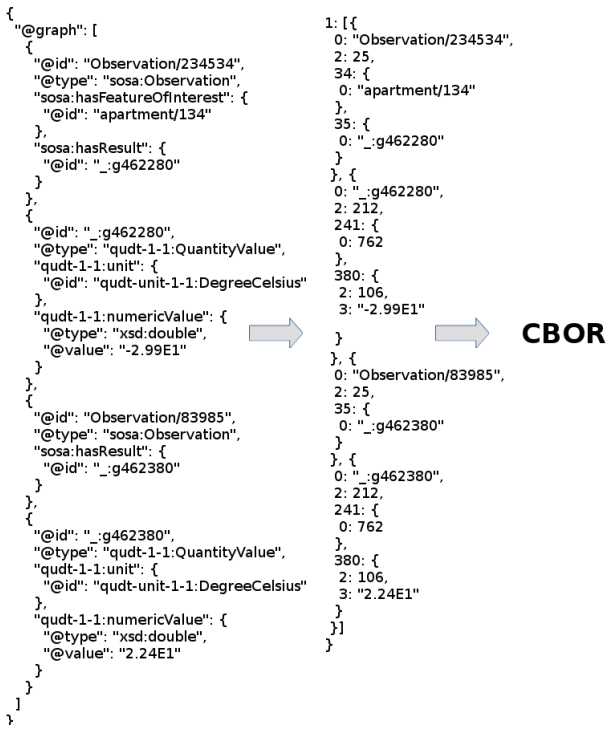

Fig. 1. JSON-LD to CBOR-LD

III. JSON-LD to CBOR Conversion

Converting JSON-LD data to CBOR involves pre-processing and transformation steps. Figure 1 illustrates an example from SSN Ontology, termed as Example 1 in the SSN document [7]. This example comprises a graph of nodes which shows the observation of the difference between the outside and inside temperature. The graph consists of two observations of outside temperature difference as well as the value of inside temperature. The first observation is represented by the node with the ID `Observation/234534`. This observation has a feature of interest, which is an apartment identified as `apartment/134`. The observation also has a result, which is a quantity value represented by the blank node `_:g462280`. The value of this quantity is a temperature difference with a unit of Degree Celsius and a numerical value of "-2.99E1" (which equals -29.9 degrees Celsius).

Second observation is represented by the node with the ID `Observation/83985`. This observation also has a result, which is another quantity value represented by the blank node `_:g462380`. The value of this quantity is a temperature inside the apartment equal to "2.24E1" (22.4 degrees Celsius). Whether the observation is about the difference or apartment's temperature will be described by other ontology features.

Representing data like above has several advantages such as interoperability among others, as discussed before. It can be seen that the knowledge about data and semantics are represented in the data itself, i.e, what is data about, what is the relation of a data entity with other standardised concepts. However, this format can be verbose leading to significant bandwidth overhead for constrained objects. Thus, CBOR-LD encodes and compacts the data as follows. First the standard JSON-LD terms like `@id, @graph, @type` are mapped to integers like $0, 1, 2$, etc. assuming a universal dictionary which can be standardised. Note that these values for mapping are subject to change as the standardisation is in process. This mapping to integers is important as CBOR is more efficient for encoding integers as compared to text. Next, the standard terms from different ontologies like SSN, SOSA, QUDT, (i.e. the ones used in the example) are mapped to integers assuming that these terms in the ontology can be obtained from the given ontology, sorted and then mapped one by one to integers. As ontologies are standardised and their terms are well known, if this above algorithm of constructing the dictionary is standardized as well then any application can construct this dictionary independently. Thus, all terms from different ontologies like `sosa:observation`, which is a string of size 16 bytes, are transformed to integer values such as 25 which takes only 1 byte. Timestamps and dates like `2017-04-16T00:00:12+00:00` are also converted to Unix time epochs as integers. Finally the graph consisting of integers is then serialised into CBOR format which finally compacts the size of the graph.

Once a graph is converted to CBOR then some processing operations, querying and partial graph extraction etc., can always be applied on it without converting it back to the original format. Thus, such data can stay in CBOR format inside the network of constrained objects and save bandwidth during exchanges. That way the need to decode it will arise only during some final operations in the application.

A. Custom dictionary

Note that after the CBOR-LD encoding, as described above, some terms will still be left which could not be encoded as integers. This is because they may be data specific terms and URLs. In this paper, we focus on 6 examples from SSN ontology document [7] numbered according to W3C recommendation: Example $1, 10, 12, 14, 17$ and 19. They are about the Indoor and Outdoor Temperature, electric consumption of an apartment, sensor used to observe tree height, observation of seismograph, movements of spinning cups on wind sensor and CO2 level observed in an ice core, as in Table I.

Some of the graphs can have specific URLs such as `Observation/234534` and `apartment/134` in Example 1 or `apartment/134/electricConsumption`, `sensor/926` in Example 10 [7]. If these terms or URIs remain static then they can be encoded to integers using a custom dictionary. If some fields are dynamic, but are integers like the `Observation/234534` then the static and dynamic parts can be separately encoded and sent as an array of 2 numbers. First integer in the array will match `Observation` and other will be the number `234534` itself.

Such a custom dictionary will be specific to the application and for example can be exchanged in an offline manner. For

JSON-LD Data	Comment	Size Bytes	Gzip Bytes	CBOR-LD Bytes, Savings	Custom Dictionary Bytes, Savings	Custom Dictionary+Gzip Bytes, Savings
SSN Example 1	Indoor and Outdoor Temperature	904	301	178, 80.3%	136, 84.9%	118, 86.9%
SSN Example 10	Electric consumption of an apartment	5322	830	1616, 69.6%	510, 90.4%	340, 93.6%
SSN Example 12	Sensor used to observe tree height	3748	573	1069, 71.5%	519, 86.2%	314, 91.6%
SSN Example 14	Observation of seismograph	3503	773	1156, 67%	480, 86.3%	390, 88.9%
SSN Example 17	Movements of spinning cups on wind sensor	2817	476	1057, 62.5%	227, 91.9%	166, 94.1%
SSN Example 19	CO2 level observed in an ice core	2414	535	754, 68.8%	209, 91.3%	194, 91.9%

TABLE I
BYTE SAVINGS WHEN USING CBOR-LD COMBINED WITH CUSTOM DICTIONARY AND GZIP

memory optimisation, only a sub-dictionary containing the required terms by the IoT object may be used. Finally, the resulting CBOR data can in some cases be compressed a bit more by using compression methods and tools such as gzip.

B. CBOR-LD C library

Our implementation provides three major functions -

- `encode_to_cbor_compressed` to convert JSON-LD data into CBOR-LD format
- `decompress_decode_to_json` to convert CBOR-LD data back into JSON-LD format
- `cborsearch` functions, which can be useful to perform searches for triples or key value pairs in CBOR-LD data.

Either a predefined dictionary or a `contextMap` is required to map context URIs and standard terms to integers. It is needed for encoding and decoding CBOR-LD data. The `documentLoader` callback loads context documents, constructing terms-to-integer mappings.

During encoding, a transform map is generated, compressing JSON by mapping its keys to integers using an algorithm applied to the context document's keys. All terms (keys) of context documents are sorted and assigned unique integers, hence keys of the JSON-LD document will be an integer in compressed form and the term map can be generated again using the same algorithm while decoding.

IV. EVALUATION

We now evaluate the performance of CBOR-LD library with and without a custom dictionary. The 6 examples were taken from SSN document [7] for this study and they were converted to JSON-LD using RDF distiller[3]. The URLs of different ontologies which are provided by the key `@context` were removed as they are well known URLs.

Table I shows the results. We can see that CBOR-LD can reduce the size of data significantly. We estimate savings as $100 \cdot \frac{\texttt{Original size - Encoded size}}{\texttt{Original size}}$. We can see that CBOR-LD can provide savings ranging from 62% to 80% approx. Some graphs like Example 14 could not reach higher savings because they have several specific URLs.

Now, when we apply the custom dictionary based conversion then we see that higher savings are achievable as compared to the default CBOR-LD algorithm. With custom dictionary encoding specific URLs to integers we can achieve between 85% to 92% savings with the examples considered.

There is still some more scope for improvement. For example, the blank node identities like `_:g462380` were not encoded and the ideas discussed before can be explored.

We also observe that gzip can help in compacting the data as well. Sometimes gzip provides better performances as compared to vanilla CBOR-LD. However, CBOR-LD combined with custom dictionary as well as a compression tool like gzip provides the best performance and we are able to obtain savings up to 94.1%. Though, sometimes when small size data is compressed using gzip then it takes more space than the original. Hence, the resulting size should be checked before sending the data over the network.

V. CONCLUSION

This paper proposed a C library for encoding JSON-LD terms into integers and converting them to CBOR format. The proposed approach offers significant savings in terms of network overhead, making it an ideal solution for constrained WoT objects. Future work includes further optimizations for dictionary creation and testing in real world scenarios.

VI. ACKNOWLEDGEMENTS

This work is supported by grant ANR-19-CE23-0012 from the Agence Nationale de la Recherche, France, for the CoS-WoT project[4].

REFERENCES

[1] M. Sporny, D. Longley, G. Kellogg, M. Lanthaler, and N. Lindström, "JSON-LD 1.1," *W3C Recommendation, Jul*, 2020.

[2] J. D. Fernández, M. A. Martínez-Prieto, and C. Gutierrez, "Compact representation of large RDF data sets for publishing and exchange," in *International Semantic Web Conference*. Springer, 2010, pp. 193–208.

[3] B. McBride, "The resource description framework (rdf) and its vocabulary description language rdfs," in *Handbook on ontologies*. Springer, 2004, pp. 51–65.

[4] H. R. Bazoobandi, S. de Rooij, J. Urbani, A. ten Teije, F. van Harmelen, and H. Bal, "A compact in-memory dictionary for RDF data," in *The Semantic Web. Latest Advances and New Domains: 12th European Semantic Web Conference, ESWC 2015, Portoroz, Slovenia, May 31–June 4, 2015. Proceedings 12*. Springer, 2015, pp. 205–220.

[5] F. Karim, M.-E. Vidal, and S. Auer, "Compact representations for efficient storage of semantic sensor data," *Journal of Intelligent Information Systems*, pp. 1–26, 2021.

[6] V. Charpenay, S. Käbisch, and H. Kosch, "Towards a binary object notation for RDF," in *The Semantic Web: 15th International Conference, ESWC 2018, Heraklion, Crete, Greece, June 3–7, 2018, Proceedings 15*. Springer, 2018, pp. 97–111.

[7] S. S. N. Ontology, "W3C Recommendation. 19 October 2017," *URL: https://www. w3. org/TR/vocab-ssn*.

[3] http://rdf.greggkellogg.net/distiller?command=serialize

[4] https://coswot.gitlab.io/