Model CBOR Serialization for Federated Learning

Koen Zandberg
Freie Universität Berlin
koen.zandberg@fu-berlin.de

Mayank Gulati
Freie Universität Berlin
mayank.gulati@fu-berlin.de

Gerhard Wunder
Freie Universität Berlin
g.wunder@fu-berlin.de

Emmanuel Baccelli
Inria
emmanuel.baccelli@inria.fr

Abstract—The typical federated learning workflow requires communication between a central server and a large set of clients synchronizing model parameters between each other. The current frameworks use communication protocols not suitable for resource-constrained devices and are either hard to deploy or require high-throughput links not available on these devices. In this paper, we present a generic message framework using CBOR for communication with existing federated learning frameworks optimised for use with resource-constrained devices and low power and lossy network links. We evaluate the resulting message sizes against JSON serialized messages where compare both with model parameters resulting in optimal and worst case serialization length, and with a real-world LeNet-5 model. Our benchmarks show that with our approach, messages are up to 75 % smaller in size when compared to the JSON alternative.

I. Introduction

Artificial intelligence (AI), and in particular machine learning (ML) using deep neural networks (DNN) have seen a spectacular development over the last decade. In this context, nearly all verticals are substantially impacted by ML, typically based on a data pipeline requiring the use of a model, i.e. a layered structure of algorithms which interpret data and make decisions based on that data. This model must first be trained in the learning phase, before it can be used for inference and put in production.

Recently, the TinyML community has been demonstrating the feasibility of executing model inference even on small microcontrollers, after these models have been trained and compressed on more powerful machines, as the learning phase typically requires enormous amounts of data and computing capacity. Even more recently, however, the TinyML community explores the potential of learning on low-power microcontrollers [1]. In this context, a trend is federated learning (FL) a machine learning paradigm where a model is trained across multiple decentralized devices without directly sharing their data with a central server. FL and on-device training is attractive because of reasons including privacy, as raw privacy sensitive data does not leave the edge device, and customization with fine-tuning happening on-premises.

A. Federated Learning Communication

The typical workflow of FL involves: (1) selecting participating devices; (2) sending a global model to each device; (3) training the model on local data; (4) sending local updates to a central server; (5) aggregating the updates to obtain a new global model; and (6) repeating the process until convergence. The resulting model is then sent back to the participating devices for inference. Throughout the FL process, communication is crucial for managing the exchange of model updates between the central server and the clients. It enables the synchronization of models, preserves privacy, and facilitates collaborative learning across distributed devices. Efficient and secure communication protocols are essential for successful FL implementation.

Typical FL frameworks such as TensorFlow Federated [2], Flower [3], etc. commonly rely on JSON [4] or gRPC [5] as part of their communication stack. gRPC utilizes Protocol Buffers (Protobuf) [6] as its message structure for data serialization and transmission. Protobuf is a language-agnostic format that allows to define message types, fields, and optional values in a concise and efficient manner for serializing, deserializing, and manipulating the message objects. However, setting up gRPC dependencies on severely resource-constrained devices can be challenging and human-readable JSON encoding is not optimized for embedded machine-to-machine communication. To address this, alternative communication protocols are often used. For instance, WebSockets [7] present a lightweight and real-time communication solution that is well-suited for such devices. MQTT [8], specifically designed for resource-constrained environments, minimizes network bandwidth and power consumption while ensuring reliable FL communication. Furthermore, the widely supported HTTP/HTTPS protocols offer a standardized and user-friendly option for transmitting model updates in FL workflows.

However with severely constrained devices these protocols are still too resource-intensive and not available. Edge devices such as microcontrollers used in smart appliances have severely limited processing capabilities and memory, between 10 MHz to 100 MHz and 16 KiB to 256 KiB. Furthermore the network link used by these devices is not optimized for the low latency and high throughput required by most protocols. The typical stack of combining TCP and HTTPS with JSON or gRPC payloads puts a too heavy burden on these devices. While MQTT is already used with FL, other lightweight alternatives for these protocols used in the constrained device space such as MQTT-SN [9], CoAP [10] and CBOR [11] are readily available, but have not been applied to the FL space yet.

B. Contributions

In this paper, the work we present mainly consists in the following:

- We propose TinyFL a generic message data framework for FL, on resource-constrained network nodes such as microcontroller-based devices. This framework enables efficient dissemination, monitoring and retrieval of ML models among these nodes.
- In order to minimize its footprint, TinyFL leverages the network protocol stacks and libraries typically present in the firmware of resource-constrained devices.
- We evaluate the sizes of the messages incurred with TinyFL with a set of different ML model sizes [1]. We evaluate based both on idealized ML models and on LeNet-5, a real-world model usable on microcontrollers;
- We show that the CBOR-encoded messages used by TinyFL reduce the serialized size by up to 75 % compared to a vanilla approach using JSON for instance. We show that the largest messages in the framework are only sent occasionally causing a minimal burden on the network link during the learning phase.

II. Related Work

Previous research, as demonstrated by existing work such as [12] and TinyFedTL [13], has attempted to enable FL training on small devices. However, these studies lack a thorough analysis of computation and communication related to different model sizes, which is essential for understanding the practicality and suitability of FL on tiny devices. In contrast, EgdeML [14] emphasizes the significance of a robust communication stack in FL systems and points out the predominant focus on improving computation rather than optimizing the communication layer in existing research. To address this challenge, we propose a modular framework that facilitates the interaction between the communication and computation layers, specifically designed for conducting FL on constrained devices.

III. Background

FL is a research area that has gathered significant attention in recent years. It is a decentralized ML paradigm where multiple devices or clients collaboratively train a global model without sharing their raw data. Instead, model updates are exchanged between the clients and a central server, allowing the server to aggregate and refine the shared model.

The concept of FL was introduced in the seminal paper [15]. This paper laid the foundation for FL by proposing a framework to train deep neural networks using decentralized data. The authors highlighted the challenges of FL, such as communication efficiency and privacy preservation.

To address these challenges, subsequent research focused on improving communication efficiency in FL. The paper by [16] proposed strategies such as subsampling and quantization to reduce the amount of model updates exchanged between the clients and the server.

As FL gained traction, system design considerations became crucial. Numerous advancements have been made extending the initial FL framework and addressing various aspects of deploying FL at scale. The comprehensive study [17] touchs upon issues related to scalability, fault tolerance, and security in large-scale FL systems.

Our research focuses on using small devices as FL clients to perform computational and communication tasks that contribute to the overall learning process. By utilizing tiny devices, our goal is to distribute workloads and improve the efficiency of ML. We aim to tackle resource limitations and optimize communication protocols, enabling effective learning on these devices. Our ultimate aim is to contribute to the development of efficient and privacy-preserving distributed ML systems.

A. Hardware and Basic Communication aspects

The smallest of edge devices typically run on small constrained microcontrollers with low power interconnected via low throughput network links. These devices have limited memory and computational processing power available, barely sufficient for small ML models. The resources available on these devices varies between 10 MHz to 100 MHz and 16 KiB to 256 KiB, with typical network links available such as IEEE802.15.4 [18] limited to 250 kbit with 127 B maximum frame size. In the usual network topology, these devices connect via a gateway to a centralized and unconstrained entity acting as server. Depending on the exact network topology, multiple network hops via intermediate (constrained) devices are necessary for a device to reach the gateway and contact the server. These restrictions translate into a set of protocols and message formats optimized to deal with these constraints.

1) CoAP: CoAP [10], as constrained counterpart to HTTP, provides a mechanism to interact with the clients in a constrained environment without putting significant burden on the wireless link and the client's processing power. The protocol lends itself for RESTful interaction between clients and server over lossy and low power networks. The observe mechanism allows for a CoAP client to subscribe to changes on a resource on a CoAP server, with publish/subscribe semantics, letting the CoAP server provide it with updates when the resource changes.

2) CBOR: CBOR [11] is a serialisation format similar to JSON with the main difference that it is a binary format. It is optimised to result in a small serialised format and suitable for constrained links. As CBOR dynamically extends the number of bytes required to encode numerical values, the encoded size heavily depends

[1]Source code for generating the results available at: https://anonymous.4open.science/r/TinyFL-results/

on the exact values encoded in the structure. For example, low value integers up to 23 can be encoded in a single byte, with the number of bytes used increasing gradually to increase the required numerical range by the encoded value. Additionally, CBOR also supports tagged data, a set of standardized tags are used to provide additional information on the following item. This is used for example when encoding Universally Unique Identifiers (UUIDs), which are encoded as a byte string with tag 37. Existing protocols such as SUIT [19] and SenML [20] make use of CBOR for serialising their messages when used with constrained devices.

3) CDDL: Describing the different CBOR data structures is possible using CDDL [21]. The full expressiveness of CBOR, which exceeds that of JSON, can be unambiguously defined via CDDL. It is both machine and human readable and can be used to validate CBOR data instances.

IV. Scenario

In the scenario described here we assume that the FL clients consists of a large number of microcontrollers, networked together over an already secured IEEE802.15.4 network using 6LoWPAN such as described by Figure 1. Existing protocols are already available to provide mesh network capabilities and connect devices together in a secure way using protocols such as 6TiSCH [22]. A number of software components are already present on the client firmware, reusing these for the FL workflow would offer multiple benefits such as reduced memory usage. The firmware running on the microcontrollers is assumed to include a number of existing modules for protocols among at least: CoAP to provide REST-like communication with a server and CBOR for serializing messages. Furthermore, it is assumed that there is a protocol in place for system-level management of the clients, such as CORECONF [23] to provide the initial on-boarding and configuration of clients. While we limit the scenario to using CoAP, other protocols optimised for constrained environments such as MQTT-SN could also be used in the workflow without changing the message formats.

V. Architecture

The full FL architecture is a multi-step process started and orchestrated by a central server. As the first step, the server sets up the orchestration process, defining the number of participating clients for training, minimum fraction of these clients required for aggragation, the number of FL rounds to be performed, and the stop condition for individual clients. Furthermore, the server decides a minimum of data samples required in order to accomplish local training. This is necessary to ensure that the local model is trained suffiently enough to make its contribution count for global model aggregation.

Once the orchestration is configured, the server initializes a global model. At the beginning of each round, a

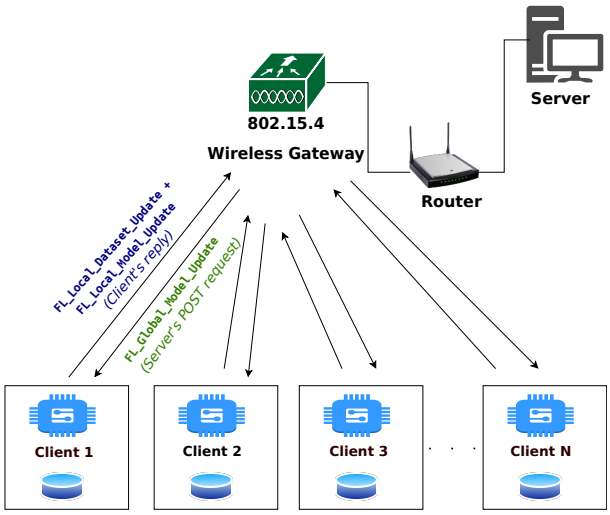

Figure 1: CoAP communication between FL Server and Clients.

global model is sent to each participating client device. Each client then independently trains the model by performing several iterations of gradient update steps using their local dataset. The clients have two types of datasets: a training set and a separate validation set, used for training and evaluating models, respectively. Meanwhile, the server observes if a particular client has been trained sufficiently. In response, clients reply with the number of data samples seen so far during training and other performance metrics, such as loss and accuracy, which can be later used to determine the stopping condition for an individual client when the metrics show no significant improvement. During our experimental process, we incorporated a stopping condition based on the comparison between the validation loss and the training loss. Specifically, when the validation loss became lower than the training loss, the server would intervene and halt the training process for that specific client.

After a sufficient number of clients respond, the server will request the trained model parameters from these clients, along with the sizes of their respective datasets. This is done in order to perform model aggregation, such as weighted model averaging, as seen in the FedAvg [15] approach.

A. Communication

As described above, the FL workflow is a multi-step process over multiple rounds. This requires multiple requests and responses from the server to the clients. An overview of the communication flow between the server and a single client is shown in Figure 2.

1) Client configuration: The first requests from the server to the clients is a POST request with the initial or new global model. This request is CBOR-encoded and is formatted as shown in Listing 1. The data in the request contains an UUID (the `fl-model-identifier`, with the UUID encoded as tagged byte string) to identify the

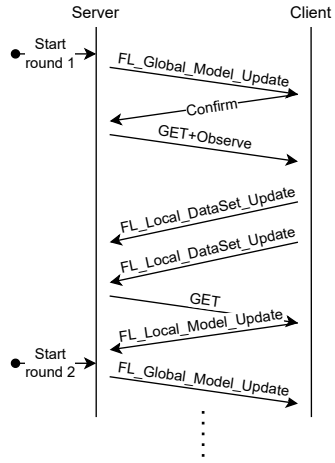

Figure 2: CoAP communication timing diagram between FL Server and a single client.

```
FL_Global_Model_Update = [
  fl-model-identifier,
  fl-model-round,
  fl-model-params,
  fl-continue-training : bool
]

fl-model-identifier = #6.37(bstr)
fl-model-round = uint
fl-model-params /= [+ float]
fl-model-params /= ta-float16le
fl-model-params /= ta-float32le
fl-model-params /= ta-float64le
```

Listing 1: CDDL description of a global model update payload

model and to allow for multiple models simultaneously on the clients. A round number is included to version the global model between rounds. Multiple encodings are allowed for serialising the list of parameters, the simplest being an array of floating point numbers, the CBOR encoding allowing for dynamically scaling between half-float, float and doubles depending on the required accuracy for the provided values. The four other encoding options can be used when the number of parameters all use the same type and can be encoded as an homogeneous type following the CBOR typed array format [24]. In this format the parameters are encoded as a byte string with each parameter concatenated in the representation specified by a CBOR tag around the byte string. Finally the request includes a flag to specify whether the client should start training a new local model based on the supplied global model or run in inference-only mode.

After starting the new training period on the clients, the server submits the CoAP GET with observe request to the clients. The level of training required by the server is submitted as query parameter in the CoAP GET request. Via the observe mechanism, this request provides the server with a notification as soon as a client reached the the minimum level of training required by the client. The format in CDDL used by the clients for the replies are shown in Listing 2. The main content of this reply

```
FL_Local_DataSet_Update = [
  fl-local-dataset-size : uint,
  ? fl-model-metadata,
]

fl-model-metadata = (
  fl-local-model-train-loss: float
  fl-local-model-val-loss : float
)
```

Listing 2: CDDL description of a client update payload

is the size of the local data set so far. Furthermore the performance and loss values can be provided as additional information to the server. These updates provided by the clients are used by the server to track the FL process among the clients. Based on the information provided by the client, the server can select the most promising candidates from the clients for the global model update.

After a sufficient number of clients have all gathered sufficient local data for training the local model, the server queries the selected clients for their local model update. This is done via a GET request and the clients reply with a structure shown in Listing 3.

```
FL_Local_Model_Update = [
  fl-model-identifier,
  fl-model-round,
  fl-model-params,
  fl-model-metadata,
]
```

Listing 3: CDDL description of a client model payload

The update contains the same UUID and round number used with the initial global model update described in Listing 1. As main data, the client provides the local dataset size used to train the model with, and the local model itself as list of floating point numbers.

VI. Evaluation

A. Measurement setup

To evaluate the message framework proposed here, we evaluate the different messages on a two metrics.

1) Message size: The main metric for evaluation is the message size in bytes. As the scenario involves severely constrained network links, keeping the message size below the maximum frame size of the link (127 B) prevents requiring multiple radio transmissions for a single message. As resulting size of the CBOR encoding depends a lot on the value of the data, we provide both optimistic and pessimistic values for the message sizes. The measurements were done using three different simulated model sizes: a small model of 4 floating point numbers, an intermediate

model size of 1000 floating point numbers and a large model with 10 000 floating point numbers. We also measure the size of the messages when they are encoded as Protobuf and as minified JSON messages, to show the reduction in message size due to the CBOR encoding. For the floating point numbers, the value 1.0 was chosen, as this requires the least amount of encoding size with JSON. The measured size for JSON represents the minimal size that can be achieved and should be compared against the best case with the CBOR encoding.

2) Message interval: The rate at which messages are send between the server and clients can be classified. Preferably larger messages are only required in exceptional cases, in contrast to the updates from clients to server, which should be small as they happen often. The exact interval of the messages depend on the training configuration.

B. Measurements

1) Message size: The messages sizes for different model sizes is shown in Table I. First, the FL_Local_DataSet_Update message that contains the updates from the client is between 8 B and 28 B with CBOR, depending on the exact values used. While the JSON encoded variant is slighly larger than the best case CBOR encoding, both are suitably small to fit inside a single frame on a constrained network. The protobuf message is in same range as the CBOR message size. For CBOR, the message sizes for messages that contain a small model are between 33 B and 84 B. With larger model sizes between 1000 and 10 000 parameters, the message sizes increase with the same magnitude. The best case for CBOR is using half floats, which is visible when comparing with the Protobuf sizes, as Protobuf always uses at least 32 bit floats. As a JSON-encoded float inside an array needs 4 characters and a CBOR-encoded half float inside a tagged array needs 2 bytes, the size of the CBOR-encoded messages approximates 50 % of the JSON-encoded messages. These messages are no longer small enough to fit inside a single transmission frame and require the CoAP blockwise transfer mechanism to get fully transferred. The FL_Local_DataSet_Update size is independent of the number of the model parameters, it does not contain the model parameters.

2) Message interval: Looking at how often the different messages need to be transferred we can distinguish two update frequencies. The FL_Local_DataSet_Update will be transmitted relative often and is unique per client during the training round. To save throughput and in turn power on the clients, it is important that this message is as small as possible. At 28 B maximum it will always fit inside a single transmission from the client.

The FL_Global_Model_Update depends on the size of the model and can potentially be a large message that needs to be distributed to all clients. However it only needs to be distributed once per round. As all clients need the same message, strategies to disseminate this message through the network can be applied, such as using a single multi-cast message reaching all clients. The FL_Local_Model_Update from the clients to the server is also transferred only once per round. Each selected client will have to transfer the FL_Local_Model_Update message to the server with their locally trained model parameters. As this message is unique per client and contains the parameters of the model, it puts the largest burden on the network. However, not all clients have to transfer this message, but only the clients selected by the server.

3) Real world example: Finally we compare the message sizes when using the LeNet-5 [25] model (approx. 45 000 parameters) in the message structures. The results are shown in Table II. The FL_Local_DataSet_Update message has been omitted as the size is independent of the model size and is identical to the previous shown measurements. Visible is the significant gain by encoding the messages in CBOR. Where previous tables compared a best case CBOR messages with a best case JSON messages in terms of size, here the average case with real-world values is shown. Compared to JSON, the message size is around 24 % of the size.

VII. Discussion and Future Work

With the minimal encoding structure from CBOR, flexibility is available to encode the models in their smallest size possible. The CBOR tagged-array format results in a message size at most the same size as the as the required memory for the model parameters on the device. However depending on the specific model parameter values it can result in a size smaller than the on-device representation.

In future work, it could be valuable to explore the possibility of personalizing the global model on the client side before deploying it for inference. Allowing clients to fine-tune the global model using their local data and domain-specific knowledge could lead to improved performance and tailored predictions. Additionally, it would be worth investigating the capability of discarding server updates and relying solely on local updates if the performance of the global model deteriorates. This approach would require monitoring the model's performance at the client level and selectively using local updates to maintain or improve performance. However, integrating these capabilities into constrained FL system requires careful consideration of factors such as compute resource availability, model complexity, and communication overhead. Developing novel algorithms and protocols will be essential to enable efficient and effective collaboration between clients and the server. Moreover, our framework could be used to transfer of partial models, allowing for flexible and efficient transfer learning scenarios, where only specific layers or components need to be exchanged.

Message	Model Size	CBOR Best	CBOR Worst	Protobuf	JSON
FL_Local_DataSet_Update		8 B	28 B	22 B	11 B
FL_Global_Model_Update	4	33 B	67 B	40 B	65 B
	1000	2027 B	9033 B	4025 B	4049 B
	10 000	20 025 B	90 033 B	40 026 B	40 049 B
FL_Local_Model_Update	4	38 B	84 B	58 B	68 B
	1000	2032 B	9050 B	4043 B	4052 B
	10 000	20 032 B	90 050 B	40 044 B	40 052 B

Table I: Message sizes when encoded as CBOR, Protobuf and JSON, for varying model sizes

Message	CBOR	ProtoBuf	JSON
FL_Global_Model_Update	177 733 B	177 730 B	928 171 B
FL_Local_Model_Update	177 738 B	177 748 B	928 168 B

Table II: Message sizes when encoding the LeNet-5 model as CBOR and JSON

VIII. Conclusion

Our message and communication framework provides a highly effective approach for implementing FL on microcontrollers, surpassing the limitations of traditional Protobuf and JSON-based methods and communication protocols. There are several key advantages to our framework. Firstly, it strikes an optimal balance between message size and frequency, ensuring efficient and resource-conscious communication. Secondly, it seamlessly integrates with existing IoT management systems, leveraging widely adopted network stack building blocks. This enables easy integration and scalability within larger IoT infrastructures. This capability enhances the versatility and adaptability of FL on microcontrollers, unlocking new possibilities for distributed learning applications.

Acknowledgements

The research leading to these results partly received funding from the MESRI-BMBF German/French cybersecurity program under grant agreements No. ANR-20-CYAL-0005 and 16KIS1395K. The paper reflects only the authors' views. MESRI and BMBF are not responsible for any use that may be made of the information it contains.

References

[1] S. S. Saha et al., "Machine learning for microcontroller-class hardware-a review," IEEE Sensors Journal, 2022.
[2] "TensorFlow Federated." [Online]. Available: https://www.tensorflow.org/federated
[3] D. J. Beutel et al., "Flower: A friendly federated learning framework," 2022.
[4] E. International, "Ecma-404—the json data interchange format," 2013.
[5] "gRPC." [Online]. Available: https://grpc.io/
[6] "Protocol buffers." [Online]. Available: https://protobuf.dev/
[7] I. Fette et al., "The websocket protocol," RFC 6455, December 2011. [Online]. Available: http://www.rfc-editor.org/rfc/rfc6455.txt
[8] U. Hunkeler et al., "Mqtt-s - a publish/subscribe protocol for wireless sensor networks." in COMSWARE, S. Choi et al., Eds. IEEE, 2008, pp. 791–798.
[9] A. Stanford-Clark et al., "Mqtt for sensor networks (mqtt-sn) protocol specification," IBM Corporation version, vol. 1, no. 2, pp. 1–28, 2013.
[10] Z. Shelby et al., "The Constrained Application Protocol (CoAP)," RFC 7252, Jun. 2014. [Online]. Available: https://www.rfc-editor.org/info/rfc7252
[11] C. Bormann et al., "Concise Binary Object Representation (CBOR)," RFC 8949, Dec. 2020. [Online]. Available: https://www.rfc-editor.org/info/rfc8949
[12] M. M. Grau et al., "On-device training of machine learning models on microcontrollers with a look at federated learning," in Proceedings ACM GoodIT, 2021, p. 198–203.
[13] K. Kopparapu et al., "Tinyfedtl: Federated transfer learning on tiny devices," arXiv preprint arXiv:2110.01107, 2021.
[14] P. Pinyoanuntapong et al., "Edgeml: Towards network-accelerated federated learning over wireless edge," Computer Networks, vol. 219, p. 109396, 2022.
[15] B. McMahan et al., "Communication-efficient learning of deep networks from decentralized data," in Artificial intelligence and statistics. PMLR, 2017, pp. 1273–1282.
[16] J. Konečnỳ et al., "Federated learning: Strategies for improving communication efficiency," arXiv preprint arXiv:1610.05492, 2016.
[17] P. Kairouz et al., "Advances and open problems in federated learning," Foundations and Trends® in Machine Learning, vol. 14, no. 1–2, pp. 1–210, 2021.
[18] "Ieee standard for low-rate wireless networks," IEEE Std 802.15.4-2020, pp. 1–800, 2020.
[19] B. Moran et al., "A Firmware Update Architecture for Internet of Things," RFC 9019, Apr. 2021. [Online]. Available: https://www.rfc-editor.org/info/rfc9019
[20] C. F. Jennings et al., "Sensor Measurement Lists (SenML)," RFC 8428, Aug. 2018. [Online]. Available: https://www.rfc-editor.org/info/rfc8428
[21] H. Birkholz et al., "Concise Data Definition Language (CDDL): A Notational Convention to Express Concise Binary Object Representation (CBOR) and JSON Data Structures," RFC 8610, Jun. 2019. [Online]. Available: https://www.rfc-editor.org/info/rfc8610
[22] P. Thubert, "An Architecture for IPv6 over the Time-Slotted Channel Hopping Mode of IEEE 802.15.4 (6TiSCH)," RFC 9030, May 2021. [Online]. Available: https://www.rfc-editor.org/info/rfc9030
[23] M. Veillette et al., "CoAP Management Interface (CORECONF)," Internet Engineering Task Force, Internet-Draft draft-ietf-core-comi-12, Mar. 2023. [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-core-comi/12/
[24] C. Bormann, "Concise Binary Object Representation (CBOR) Sequences," RFC 8742, Feb. 2020. [Online]. Available: https://www.rfc-editor.org/info/rfc8742

[25] Y. LeCun et al., “Gradient-based learning applied to document recognition,” Proceedings of the IEEE, vol. 86, no. 11, pp. 2278–2324, 1998.