Embedding ML algorithms onto LPWAN sensors for compressed communications

Antoine Bernard, Aicha Dridi, Michel Marot, Hossam Afifi, Sandoche Balakrichenan

Embedding ML Algorithms onto LPWAN Sensors for Compressed Communications

Antoine Bernard[1,2], Aicha Dridi[2], Michel Marot[2], Hossam Afifi[2], and Sandoche Balakrichenan[1]

[1]Afnic
firstname.surname@afnic.fr
[2]Samovar, Télécom SudParis, Institut Polytechnique de Paris
firstname.surname@telecom-sudparis.eu

***Abstract*—LPWANs are networks characterized by the scarcity of their radio resources and their limited payload size. To extend the efficiency of the data transmission by decreasing the traffic sent from sensors, this paper proposes a lossy compression method using known ML techniques. We embedded a pre-trained neural network directly on constrained LoRaWAN devices and we tested the trade-off between compression ratio and accuracy of the compression algorithm. This paper studies multiple aspects of the system - energy consumption, error rate due to the lossy compression, compression ratio and the impact of LSTM parameter quantization - to measure the possible strengths and weaknesses of using a dual prediction system in order to reduce transmission costs. Surprisingly, machine learning used in this context does not consume a lot of energy and it even leads to energy saving in the very constrained devices which are the sensors.**

Keywords - Internet of Things; LPWAN; Compression; Neural Networks

I. Introduction

Internet of Things (IoT) includes a wide range of technologies relying on a variety of standards and protocols. Most IoT devices, as well as the network, have constrained capabilities. Low-Power Wide-Area Networks (LPWANs cf. [1]) is one such IoT technology that aims to provide network connectivity to IoT devices distributed over a wide area. Their distinct characteristics, such as their coverage capacity ranging from ten to fifteen kilometers, long battery life with a lifespan of more than ten years, satisfy the requirements of a considerable IoT market [2]. Their cost is low because their unique architecture permits their user to built cheaper infrastructure and because on-air communications use license-free bands or re-use already owned licensed bands.

LPWANs are, by design, highly constrained networks that allow payload size up to a few tens of bytes with a long transmission time. Thus, compressing data traffic is of utmost importance. In IoT data traffic, the header size is large compared to the total transmitted data, especially when using IPv6. Static Context Header Compression (cf. [3]) is a possible way to compress headers, but the payload data themselves remain to be compressed. Long Range Wide Area Network or LoRaWAN [4] is an LPWAN technology. This paper aims to study LoRaWAN traffic compression, precisely its data payload, using a machine learning-based compression scheme.

Many IoT applications consist in monitoring: power grid or water distribution network metering, electric vehicle battery level monitoring, meteorological, temperature, humidity monitoring, etc. In most of these cases, the observed time series are highly correlated and can be forecast easily unless unexpected events occur. Thus, it is not necessary to transmit the data in most cases, but only in case of unexpected events. With a good time-series predictor both on the sensor and on the network backend, the data can be deduced at the backend without any transmission from the sensor. However, if the sensor, using the same predictor, observes that the measured data is different from the predictor's forecast, if it notes it is an unexpected event, then the sensor must send the data. With such a mechanism, we can avoid many transmissions and produce highly compressed traffic.

This paper's goal is precisely to test to what extent such an approach is well suited for IoT and particularly in the case of LoRaWAN. We want to observe the efficiency in terms of network performance (e.g. compression ratio) and power consumption since it is essential to save the batteries of sensors that are expected to have a long life. We also want to test whether our solution is feasible practically by setting up an experiment with actual sensors.

Different kinds of predictors may be envisioned, but machine learning, particularly neural networks, is well suited to model any repeated pattern. Here, we use a Long Short-Term Memory neural model (LSTM, cf. [5]) for two scenarii: power production and consumption metering and cellular base station load monitoring. We trained the neural network model on a powerful computer, and then we injected the trained model into the sensors. We measured the compression ratio and the sensor's electric consumption, taking into account the transmission and the computation cost. We run our experiments with real measured data and LoRaWAN equipment.

This work also presents our thoughts on LSTMs' accuracy and how the device's compression capabilities are impacted by LSTM accuracy.

In part II, the related works are reviewed. Then, in part III we present our experiment setup, tools and software, implementation choices and the walls we encountered. Lastly, in part IV, we present our results and discuss possible improvements to the system. V sums up our experiments and conclusions.

II. Related works

The easiest way to reduce data transmission is to delete redundancies or to round them to near values. Sensors often generate time-correlated data. For example, the temperature may vary slowly. Run-Length Encoding (RLE) takes advantage of the adjacent clustering of symbols that occur in succession. It replaces a "run" of symbols with a tuple that contains the symbol and the number of times it is repeated. The authors of [6] apply Delta Encoding followed by RLE at the end node. In [7] delta compression allows sending only the difference between two consecutive temperature measurements. Data are also usually quantized to round the measures to significant approximations requiring fewer bits for coding. They are often aggregated ([8] or [9]).

Approximating the measurements reduces their size also [10]. One can compress signals by approximating them with auxiliary, more simple functions. Lightweight Temporal Compression (LTC) [11] is an energy-efficient lossy compression algorithm that maintains a memory usage and per-sample computational cost in O(1). LTC estimates data points using a piece-wise linear function that guarantees an upper bound on the maximum absolute error between the reconstructed signal and the original one while maintaining a memory usage and per-sample latency in O(1).

Classical compression approaches based on dictionaries or entropy coding have been adapted to IoT, like [12] where a specific dictionary is created for different kinds of data depending on their change frequency. Transform methods are classical tools for compressing data but may be CPU resource consuming. [13] evaluates several lossy compression algorithms for efficiently storing weather sensor data based on the encoding of temporal changes and three signal transformation algorithms on spatial data. Specifically, they evaluate reconstructed weather sensor data fidelity using Discrete Cosine Transform, Fast Walsh-Hadamard Transform and Discrete Wavelet Transform (and also Lossy Delta Encoding). The objective is to provide useful information for minimizing data reconstruction errors, and more importantly, make sure they are within a tolerable range. Chebyshev compression is considered in [14] and [15].

Compressed sensing is a new technique. As stated in [16], in a series of pioneering works by Candes ([17], [18], [19]) and their co-authors, it was shown that when a signal has a sparse representation in a known basis, one can vastly reduce the number of samples that are required—below the Nyquist rate and still be able to recover the signal (under appropriate conditions) perfectly. This framework suggests compressing the data while sensing it; hence the name compressed sensing. Nevertheless, on the one hand, compressed sensing reduces the number of measurements and the sampling rate. However, on the other hand, it increases the computational complexity of the signal recovery ([20]). The signal is recovered approximately by solving a convex relaxation of a non-convex optimization problem. [21] proposes a unified approach for compression and authentication of smart-meter reading in advanced metering infrastructure. In [22] an algorithm is designed which combines the accuracy of standard lossless compression with the efficiency of a compressive sensing framework. It balances each technique's trade-off and optimally selects the best compression mode by minimizing reconstruction errors, given the sensor node battery state.

Recently, Neural network-based techniques entered the landscape of IoT data compression techniques. In [23], data are compressed by their regression curve obtained from a neural network. In [24], biomedical signals are compressed using autoencoders. These neural networks are three-stage networks whose input and output dimensions are the same, while the hidden stage has a smaller dimension. Thus, the first stage's output has a reduced dimension compared to the input and constitutes the compressed data.

Another part of transmission compression is header compression, recent work from IETF develops possible ways to improve payload efficiency by compressing packet headers such as ROHC [25], or SCHC [3].

Prediction methods are also used. Neural networks are known as universal function approximators with the capability to learn arbitrarily complex mappings, and in practice, show excellent performance in prediction tasks. Thus, the authors of [26] train a Recurrent Neural Network predictor followed by encoding with a traditional arithmetic coder block using the probabilities generated by the trained neural network. The decompression is performed symmetrically and requires the trained model for arithmetic decoding. In [27] a prediction scheme is implemented on cluster nodes and cluster heads to reduce data transmission. If the measured data corresponds to the predicted one, it has not to be transmitted. Neural networks (NNs) and Long Short-Term Memory networks (LSTMs) are proposed to perform predictions. We decided to push the subject further by implementing the algorithm directly on the sensors instead of relying on simulation. Our Proof-Of-Concept experiment aims to back these simulations or disprove them should the system prove unreliable.

On the subject of traffic data prediction, some papers propose to use a similar approach using large LSTMs such as [28]. Their multi-feature approach allows them to correlate data and obtain interesting results regarding traffic prediction. We hope to obtain similar results with our curves as we work with network traffic. However, our approach differs in our decision to focus on the effect of predictions on transmission: improvements to LSTM capabilities are out of our paper's scope.

This approach was also tested with energy production forecasting. LSTMs are presented as a possible candidate for energy production forecasting in [29]; the solution seems adaptative enough for our approach to be reliable enough when using LSTM as a forecasting tool.

Thus, there are many compression techniques appeared for many years. Nevertheless, if the "classical" methods may present efficient compression ratios, they do not avoid transmitting data. Actually, periodically a sensor senses data, may compress it and then send the compressed payload. Neverthe-

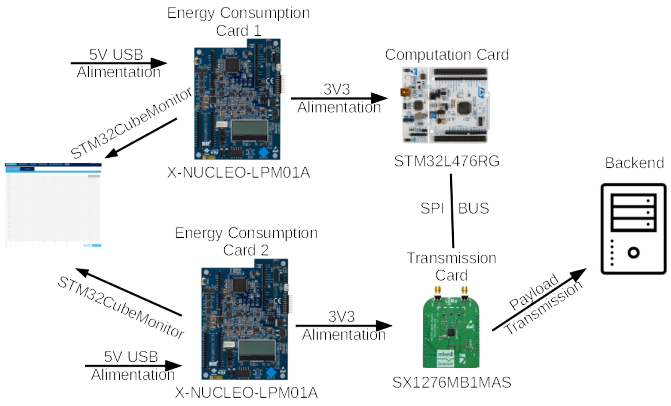

Figure 1: Experimental testbed

less, compressed data payload (plus header) are still sent. New neural network-based techniques appeared, and they avoid sending data at all in some situations where the prediction is good, but to our knowledge, they have not been tested with actual data and on real equipment. The goal of our test is precisely to propose a validation in real conditions.

For this paper, the prediction algorithms used rely on [30]. We aim to have a reliable data prediction based on an LTSM neural network and run the predictor on both the sensor and the network infrastructure. As we can expect prediction performance reliable within a 10% Mean Absolute Percentage Error (MAPE), we might expect a complementary compression coefficient up to 90% for our experiment. Such a compression ratio would allow us to build sensor networks where each device consumes less bandwidth, thus improving the scalability of LPWANs solutions further.

With our experiment, we test various neural network size and transmission threshold to measure how these parameters might influence the compression ratio of our device's transmissions.

III. Experiment

We embarked an LSTM algorithm similar to the ones studied in [27] and [30] onto an electronic device as a way to test the experimental feasibility of these solutions. These articles propose to use LSTMs that are sufficiently simple to be implemented and embarked onto devices.

The devices used for this experiment are STM32L476 [31] as measurement and calculation device which generates prediction data and compares it to measurements, Semtech's SX1276MB1MAS [32] as LoRa transmission board, and STM32 Nucleo Expansion Board [33] to measure the energy consumption of LoRaWAN transmissions. The data sets we used are occupancy data of cellular base stations for the 1st data set and power consumption of a smart building for the 2nd one, in function of the time. We used the 1st data set for all experiments except for those presented Figure 4.

A. Hand coding the neural network

Initially, we thought of basing our experiment on EdgeImpulse [34]. EdgeImpulse is an easy-to-use and well-documented framework to generate machine learning models from actual sensor data and automatically embed them onto the device. EdgeImpulse was a strong candidate to support our experiments on sensors. Unfortunately, the framework offered from EdgeImpulse did not support LSTM functionalities, thus was not adapted to our use case.

By studying the source code of the program generated by EdgeImpulse, we constated that the basic library used to execute Machine Learning algorithm was the TensorFlow Lite [35] library. So, we built an MBED OS firmware that embarks the TensorFlow Lite exported neural network and transmits data based on the predictions but could not exploit the strength of the TensorFlow Lite for microcontrollers library as the LSTM operations are not supported yet.[1]

So, we trained our LSTM network with TensorFlow [36] and exported the LSTM weights and parameters necessary to our implementations (weights, bias and hidden layers' state). We ported the Deep LSTM code successfully onto basic STM32 boards. We injected the parameters into our own implementation of the LSTM network, developed in C, and embarked the LSTM network directly on the sensor. Our implementation is available following [37]. This implementation was built thanks to the paper from Christopher Olah [38], and the following tutorial on weights and parameters extraction [39].

B. Dual prediction with LSTM

The usual approach with LSTM is to use measured values as input to the system and obtain a predicted value based on these measurements, as proposed in [40]. A modified LSTM architecture is proposed hereafter as this approach does not fit with dual prediction since the backend (i.e. the network side receiving the traffic) does not have access to the sensor's measurements. Our experiment relies on a dual prediction of data to reduce transmission. It relies on two types of data: On the one hand, we have measured values from sensors, and on the other, we have calculated (i.e. predicted) values. Our LSTM needs to keep calculating based on calculated values on both the backend and the sensor as long as transmissions are unnecessary. Actually, as long as no data is transmitted, the backend has only the calculated values, not the measured ones. Thus, the backend must run the LSTM by re-injecting these calculated data into the LSTM, and, consequently, the device must do the same to check to which extent the data predicted by the backend is far or near the data just measured. When transmissions occur, we can recalibrate the LSTM and use the new transmitted value as a new baseline for calculations.

The firmware we developed is then flashed onto an STM32L476 card to exploit the capability of LSTM combined

[1]We discussed this issue with contributors from both EdgeImpulse and TensorFlow Lite for microcontrollers and will do our best to carry on with this work and use it to contribute to the support of LSTM capabilities on TensorFlow Lite.

with LoRaWAN transmissions. This algorithm, built on MBED OS, uses an LSTM Neural Network to predict a theoretical value at a given time. It compares these theoretical values to experimental measurements at the corresponding time. We define a threshold that determines a transmission policy: if the experimental measurements differ from the predicted value within a given margin, the transmission is not realised. However, if the experimental measurements are too far from the predictions, the data is sent over the air from our LoRaWAN device to our LoRaWAN backend. This threshold might be fixed as in Figure 5, or we might study the effect of changing the value of the threshold through simulations such as with Figure 3 which studies the compression ratio one can expect with this system when picking various threshold values. Such data transmission policy may allow us to reduce the band usage for our device.

We plug our transmission card (SX1276MB1MAS) into an STM 32 Nucleo Expansion Board to monitor its energy consumption using STM32CubeMonitor [41]. We do the same with our STM32L476 card in order to study the overcost of running the LSTM.

On the backend side, we monitor data reception and aggregate the data predicted from the neural network on the infrastructure side with the data received from the sensors, allowing us to plot on a graph the combination of the actual measured data, for which we consider that the information is 100% reliable, and the predicted data which was inferred by the neural network and not disproved by sensor transmission (which is reliable up to a certain threshold). The effect of this threshold will also be studied. In this experiment, our LoRaWAN gateway is accessible through SF7 communications and is a part of TheThingsNetwork [42] community LoRaWAN Network. Figure 1 sums up our experimental setup and illustrates the various hardware components we use.

Alongside this real experimental setup, we studied the effect of changing the system's variables through extensive simulations allowing us to shorten the experimental exploration, find interesting parameters for our embedded experiment and confront simulation results to experiments.

We also studied the system's reliability. We defined the system's reliability as the MAPE of the data perceived by the backend compared to the real values. This reliability study aims to study the consequences of a bounded lossy compression on the values obtained at the system's output. Our compression is lossy because the data recorded by the back-end is not the measured one but the predicted one as long as the predicted one is within the tolerated threshold interval, and thus defining a threshold means we study the trade-off between accepting a given error on our data and improving the compression ratio. Our compression is bounded because we set it back to the actual data if the loss exceeds the given threshold.

With this experiment, we aim to evaluate:

- the energy cost added by the ML-based compression scheme at the device side;
- the energy saved on the transmission card thanks to data prediction;
- the compression ratio expected with regards to a given neural network size and data prediction threshold;
- the bounded loss introduced by the solution compared to transmitting all values;
- the impact of the quantization of the weights of the neural network predictor, by running these experiments with quantized parameters instead of floating-point numbers to improve neural network complexity, memory size and finally energy efficiency.

IV. Discussion

A. Energy

Table I: Comparison of the mean energy consumption of the calculation card and its variance, with and without LSTM-based compression (in Watts)

With Machine Learning		Without Machine Learning	
Mean value (W)	Variance	Mean value (W)	Variance
$6.31 * 10^{-4}$	$7.57 * 10^{-5}$	$7.76 * 10^{-4}$	$7.61 * 10^{-5}$

Table II: Comparison of the mean energy consumption of the transmission card and its variance, with and without LSTM-based compression (in Watts)

With Machine Learning		Without Machine Learning	
Mean value (W)	Variance	Mean value (W)	Variance
$5.48 * 10^{-4}$	$4.10 * 10^{-5}$	$9.87 * 10^{-4}$	$7.12 * 10^{-5}$

The comparison of the consumption of our cards (Table I & II) shows a save of around 40% on transmission cards. Considering IoT systems similar to our own with measurements stating that calculation and transmission consume about the same, this would represent a save of around 20% on both transmission and battery life. Considering the card we are using, an LR6 battery with a 1200mAh charge would power our device for around ten months without embedded machine learning and about a year with machine learning.

Figure 2 presents the energy passing through both our network card and calculation card, measured using STM32CubeMonitor which permits us to measure and log instantaneous consumption for our device, in function of the time. The device life-cycle follows a two-step routine. Most of the device's life is spent in a sleeping state with low energy consumption. Here in our illustration, the device's sleeping state is around 9s long to respect LoRaWAN duty cycle. The device will, exceptionally or regularly, transmit data based on its measurements. Transmitting is the other step in the routine. Transmissions translate in power consumption as three transmission spikes corresponding to data emission and the opening of two LoRaWAN listening windows. A residual energy consumption about 4.10^{-4}W can be noticed for the calculation card, while it is about 5.10^{-5}W for the transmission card. The calculation card embarks a dedicated OS which requires more permanent consumption. Irregular energy spikes

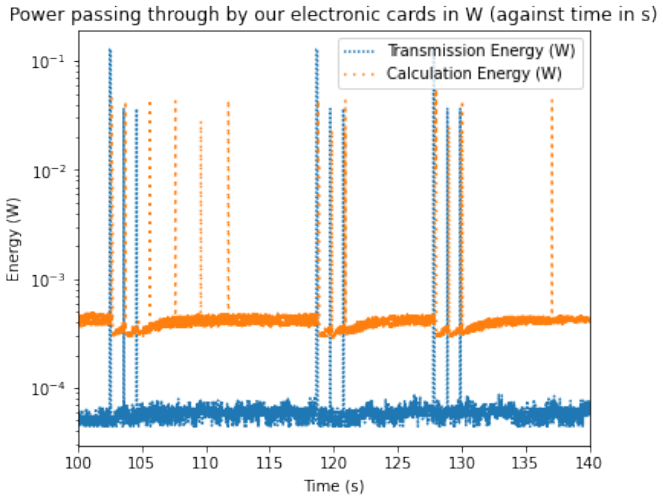

Figure 2: Energy (in W) passing through the calculation card and the transmission card (Sample)

can be observed for the calculation card, which are due to OS eventing. Our Machine Learning algorithm directly results into transmission spikes. Except for the operations realised by the OS on the calculation card, no transmission means no power spike, which leads to less power consumption as a whole, a result that can be observed on the transmission card's power consumption. With regular transmissions we would observe regular power spikes but with our method these spikes are completely cut off.

B. Compression and Mean Absolute Percentage Error

Figure 3 presents the MAPE and the compression ratio we can expect with regards to the size of the neural network and the decision threshold. We observe that, as expected, the compression ratio improves with the threshold but that the MAPE worsens. The consequences of the number of hidden layers is not significant.

Experimenting with different data-sets (Figure 4) show how the performances of the neural network in its prediction greatly influence the quality of the compression scheme. With a better overall MAPE, one might achieve around 60% compression accepting as little as 1% error in its transmissions.

The questions that come with these curves need to be addressed directly by the user. A user with concern with precision will prefer a lower MAPE, thus obtaining a lower compression ratio. If a user accepts a 1% error on its global data, setting its transmission threshold around 8%, He would end up with a 30% compression ratio for our first data-set and 85% compression ratio for the second one. Accepting more errors would permit compression ratios up to 90%. We note that the compression ratio is low with a strict threshold, but remember that contrary to classical compression methods where at least a header is sent, no packet is sent at all with our method when we compress. Thus, for a strict threshold of 10%, we decrease the overall traffic by 1 packet over 5 (20%) while keeping a global error on our overall data around 2%.

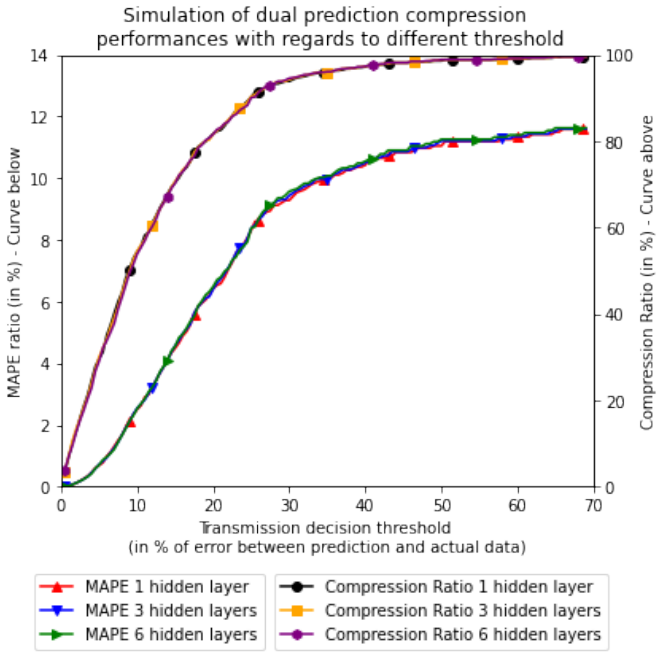

Figure 3: Compression ratio and mean absolute percentage error with regards to neural network size and precision threshold

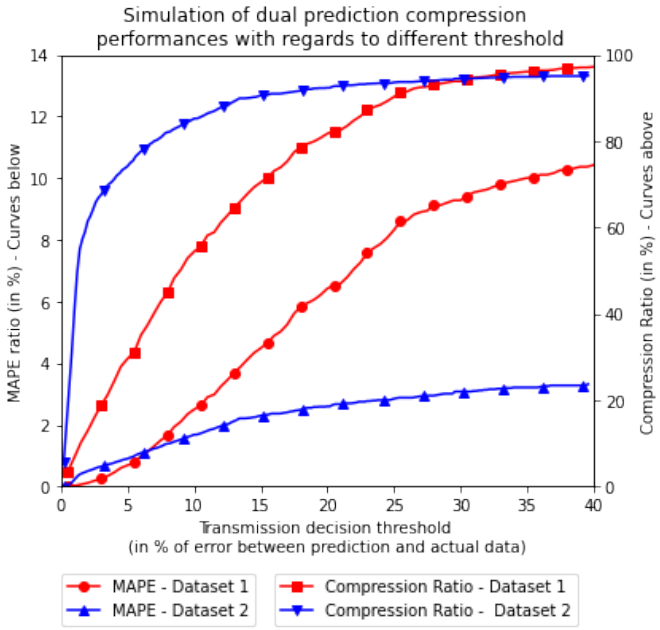

Figure 4: Compression ratio and mean absolute percentage error with regards to neural network size and precision threshold

C. Backend considerations

Figure 5 presents a comparison between the measured time-series as transmitted without ML and the calculated time-series improved with transmissions. The Calculated Data curve

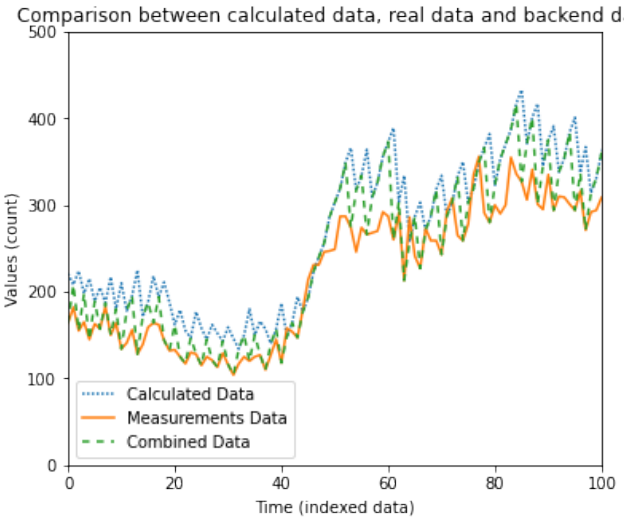

Figure 5: Comparison sample between calculated data, reference data and data perceived by the backend

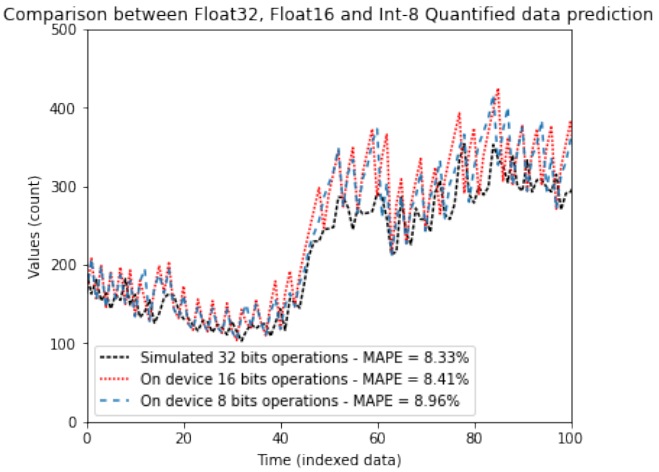

Figure 6: Float32, Float16 and Int-8 Quantized LSTM forecasting

consists solely of data calculated by the LSTM neural network. The Measurements Data curve corresponds to our LSTM target data, and its value end up transmitted should the calculated data end up being too far from the measurements. Finally, the combined data is the data curve as seen on our backend-side: the calculated data improved by the measured transmission should the two curves differ above a given threshold.

D. Quantization

Figure 6 presents our backend-side time-series as a function of the time index, combining calculated data and received measurements. The three curves on this figure differ by the number of bits necessary to code the LSTM weights, hidden layer parameters, cell state and input data. The dotted blue curve is obtained by running the dual prediction algorithm in a Python simulated environment and operating with 32-bits-encoded floats. The plain orange curve is obtained by running the dual prediction algorithm directly on the device with an LSTM operating with 16-bits-encoded floats. And the dash-dot green curve is obtained by running the dual prediction algorithm directly on the device with an LSTM operating with quantized parameters encoded on 8-bits integers.

Measurements of the compression ratio for the three above curves show no significant degradation between the three systems (the compression ratio almost does not change and remains around 70% for the three curves). 8-bit quantization is a well-documented solution to reduce the operations' complexity while working with neural networks on constrained devices. This also proves to be confirmed with our implementation of LSTM. Efficient quantization is essential when working with Neural Networks; it reduces the complexity of the operations, which might, in turn, allow for savings in processing power and battery life. Further works would be necessary on the efficiency of quantization once the LSTM operation will be ported to the TensorFlow Lite for microcontrollers library.

V. Conclusion

We built an experimental testbed to check the capabilities of on-boarding LSTM algorithm on-sensor to forecast data, achieve dual prediction, and eventually compress data traffic and save energy. A deep LSTM algorithm was developed and integrated into a small, constrained hardware to obtain these results, its source code is accessible following [37]. Our findings show that it can efficiently minimize the traffic while preventing non-relevant transmissions to occur with a significant impact on energy consumption. We observed the impact of the neural network size and the decision threshold on the compression ratio and the MAPE. Our system allows efficient compression while keeping the user within a reasonable error margin. It can be customized depending on precision and compression trade-off requirements. We also check the impact of the quantization of the LSTM parameters because of device constraints and also to decrease the complexity of the algorithm. We observed no significant degradation in the system when using 8-bit quantization.

Compressing data with this kind of Bounded Lossy Compression allows to expend battery lifetime depending on the accepted margin of error. Our results show an excellent compression ratio compared to the state of the art. Note that our scheme avoids sending any data while classical compression mechanism at least send a frame header each time a compressed data is sent. Moreover, an extensive energy consumption study proves that our algorithm saves an important energy ratio that can be used in further communication.

Further approach would consist of selecting multiple features on multiple values to attain more precision in the calculation with a more complex recalibration. Works are carried by contributing to the actual TensorFlow Lite community in order to propose a more complete port of the LSTM libraries from the global TensorFlow project to the TensorFlow Lite for microcontrollers community.

VI. Acknowledgement

This work benefited from the support of the Energy4Climate Interdisciplinary Center (E4C) of IP Paris and Ecole des Ponts ParisTech. It was supported by 3rd Programme d'Investissements d'Avenir [ANR-18-EUR-0006-02]. This work was partly financed by the French National Research Agency through the CIFRE program [2018/0668]. Our thanks to our colleagues Dr. Ghalid Abib and Florian Grante from Telecom SudParis and to Alexandre Abadie from INRIA for their advice regarding a few of the tools and hardware used in this paper. Our thanks to our colleagues Pr. Monique Becker from Telecom SudParis and Benoît Ampeau from Afnic for their reviews and advices.

References

[1] Farrell, S., Ed. "Low-Power Wide Area Network (LPWAN) Overview", RFC 8376. *IETF lpwan working group*, May 2018. https://www.rfc-editor.org/info/rfc8376.

[2] LPWAN market to reach $65 billion by 2025, 2019. https://www.smart-energy.com/industry-sectors/business-finance-regulation/lpwan-market-to-reach-65-billion-by-2025/.

[3] Minaburo, A., Toutain, L., Gomez, C., Barthel, D., and JC. Zúñiga. "SCHC: Generic Framework for Static Context Header Compression and Fragmentation", RFC 8724. *IETF lpwan working group*, April 2020. https://www.rfc-editor.org/info/rfc8724.

[4] LoRa Alliance, Inc. LoRaWAN Specifications, Accessed Oct 2019. https://lora-alliance.org/about-lorawan.

[5] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.

[6] A. K. M. Al-Qurabat, C. Abou Jaoude, and A. K. Idrees. Two tier data reduction technique for reducing data transmission in iot sensors. In *2019 15th International Wireless Communications Mobile Computing Conference (IWCMC)*, pages 168–173, June 2019.

[7] B. R. Stojkoska and Z. Nikolovski. Data compression for energy efficient iot solutions. In *2017 25th Telecommunication Forum (TELFOR)*, pages 1–4, Nov 2017.

[8] T. Boshita, H. Suzuki, and Y. Matsumoto. Compression method of position information for iot-based bus location system using lorawan. In *2018 Eleventh International Conference on Mobile Computing and Ubiquitous Network (ICMU)*, pages 1–2, Oct 2018.

[9] X. Zhao, V. Sadhu, and D. Pompili. Analog signal compression and multiplexing techniques for healthcare internet of things. In *2017 IEEE 14th International Conference on Mobile Ad Hoc and Sensor Systems (MASS)*, pages 398–406, Oct 2017.

[10] V. Alieksieiev. One approach of approximation for incoming data stream in iot based monitoring system. In *2018 IEEE Second International Conference on Data Stream Mining Processing (DSMP)*, pages 94–97, Aug 2018.

[11] T. Schoellhammer, B. Greenstein, E. Osterweil, M. Wimbrow, and D. Estrin. Lightweight temporal compression of microclimate datasets [wireless sensor networks]. In *29th Annual IEEE International Conference on Local Computer Networks*, pages 516–524, Nov 2004.

[12] K. MATSUDA and M. KUBOTA. Compound compression method for gathering traffic of iot/cps data. In *2019 IEEE 5th World Forum on Internet of Things (WF-IoT)*, pages 761–766, April 2019.

[13] A. Moon, J. Kim, J. Zhang, and S. W. Son. Lossy compression on iot big data by exploiting spatiotemporal correlation. In *2017 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–7, Sep. 2017.

[14] A. Ukil, S. Bandyopadhyay, A. Sinha, and A. Pal. Adaptive sensor data compression in iot systems: Sensor data analytics based approach. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5515–5519, April 2015.

[15] A. Ukil, S. Bandyopadhyay, and A. Pal. Iot data compression: Sensor-agnostic approach. In *2015 Data Compression Conference*, pages 303–312, April 2015.

[16] E. Zisselman, Amir Adler, and M. Elad. *Compressed Learning for Image Classification: A Deep Neural Network Approach*, volume 19, pages 3 – 17. Elsevier, 10/2018 2018.

[17] E. J. Candès. Compressive sampling. In *International Congress of Mathematicians, ICM 2006*, volume 3, pages 1433–1452, 2006. Cited By :2334.

[18] E. J. Candes and M. B. Wakin. An introduction to compressive sampling. *IEEE Signal Processing Magazine*, 25(2):21–30, March 2008.

[19] D. L. Donoho. Compressed sensing. *IEEE Transactions on Information Theory*, 52(4):1289–1306, April 2006.

[20] T. Vlašić, J. Ivanković, A. Tafro, and D. Seršić. Spline-like chebyshev polynomial representation for compressed sensing. In *2019 11th International Symposium on Image and Signal Processing and Analysis (ISPA)*, pages 135–140, Sep. 2019.

[21] Y. Lee, E. Hwang, and J. Choi. A unified approach for compression and authentication of smart meter reading in ami. *IEEE Access*, 7:34383–34394, 2019.

[22] S. Kartakis, M. M. Jevric, G. Tzagkarakis, and J. A. Mccann. Energy-based adaptive compression in water network control systems. In *2016 International Workshop on Cyber-physical Systems for Smart Water Networks (CySWater)*, pages 43–48, April 2016.

[23] J. Park, H. Park, and Y. Choi. Data compression and prediction using machine learning for industrial iot. In *2018 International Conference on Information Networking (ICOIN)*, pages 818–820, Jan 2018.

[24] D. Del Testa and M. Rossi. Lightweight lossy compression of biometric patterns via denoising autoencoders. *IEEE Signal Processing Letters*, 22(12):2304–2308, Dec 2015.

[25] Bormann, C., Burmeister, C., Degermark, M., Fukushima, H., Hannu, H., Jonsson, L-E., Hakenberg, R., Koren, T., Le, K., Liu, Z., Martensson, A., Miyazaki, A., Svanbro, K., Wiebke, T., Yoshimura, T., and H. Zheng. "RObust Header Compression (ROHC): Framework and four profiles: RTP, UDP, ESP, and uncompressed", RFC 3095. *IETF lpwan working group*, July 2001. https://www.rfc-editor.org/info/rfc3095.

[26] M. Goyal, K. Tatwawadi, S. Chandak, and I. Ochoa. Deepzip: Lossless data compression using recurrent neural networks. In *2019 Data Compression Conference (DCC)*, pages 575–575, March 2019.

[27] A. Jarwan, A. Sabbah, and M. Ibnkahla. Data Transmission Reduction Schemes in WSNs for Efficient IoT Systems. *IEEE Journal on Selected Areas in Communications*, 37(6):1307–1324, June 2019.

[28] Zheng Zhao, Weihai Chen, Xingming Wu, Peter CY Chen, and Jingmeng Liu. LSTM network: a deep learning approach for short-term traffic forecast. *IET Intelligent Transport Systems*, 11(2):68–75, 2017.

[29] A. Gensler, J. Henze, B. Sick, and N. Raabe. Deep Learning for solar power forecasting — An approach using AutoEncoder and LSTM Neural Networks. In *2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 002858–002865, 2016.

[30] Aicha Dridi, Hatem Khedher, Hassine Moungla, and Hossam Afifi. An Artificial Intelligence Approach for Time Series Next Generation Applications. pages 1–6, 06 2020.

[31] STM32L476, 2021. https://www.st.com/en/microcontrollers-microprocessors/stm32l476rg.html.

[32] Semtech's SX1276MB1MAS, 2021. https://www.semtech.com/products/wireless-rf/lora-transceivers/sx1276mb1mas.

[33] STM32 Nucleo Expansion Board, 2021. https://www.st.com/content/st_com/en/products/evaluation-tools/product-evaluation-tools/stm32-nucleo-expansion-boards/x-nucleo-lpm01a.html.

[34] EdgeImpulse, 2020. https://edgeimpulse.com/.

[35] TensorFlow Lite, 2020. https://www.tensorflow.org/lite/.

[36] TensorFlow, 2020. https://www.tensorflow.org/.

[37] C Implementation of Simple LSTM network, 2021. https://github.com/Bernard-A/LSTM-by-Hand-FairyOnIce/blob/main/CPP/main.cpp.

[38] Understanding LSTM Netwoks, August 2, 2015. https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[39] Extract weights from Keras's LSTM and calcualte hidden and cell states, February 19, 2018. https://fairyonice.github.io/Extract-weights-from-Keras's-LSTM-and-calcualte-hidden-and-cell-states.html.

[40] Hasim Sak, Andrew W Senior, and Françoise Beaufays. Long short-term memory recurrent neural network architectures for large scale acoustic modeling. 2014.

[41] STM32 Cube Monitor Desktop Application, 2021. https://www.st.com/en/development-tools/stm32cubemonitor.html.

[42] The Things Network, 2021. https://www.thethingsnetwork.org/.