JYU DISSERTATIONS 645

Olli Väänänen

Lightweight Methods to Reduce the Energy Consumption of Wireless Sensor Nodes with Data Compression and Data Fusion

UNIVERSITY OF JYVÄSKYLÄ
FACULTY OF INFORMATION TECHNOLOGY

JYU DISSERTATIONS 645

Olli Väänänen

Lightweight Methods to Reduce the Energy Consumption of Wireless Sensor Nodes with Data Compression and Data Fusion

Esitetään Jyväskylän yliopiston informaatioteknologian tiedekunnan suostumuksella
julkisesti tarkastettavaksi yliopiston Agora-rakennuksen auditoriossa 3
toukokuun 30. päivänä 2023 kello 12.

Academic dissertation to be publicly discussed, by permission of
the Faculty of Information Technology of the University of Jyväskylä,
in building Agora, Auditorium 3, on May 30, 2023 at 12 o'clock noon.

JYVÄSKYLÄN YLIOPISTO
UNIVERSITY OF JYVÄSKYLÄ

JYVÄSKYLÄ 2023

Editors
Marja-Leena Rantalainen
Faculty of Information Technology, University of Jyväskylä
Ville Korkiakangas
Open Science Centre, University of Jyväskylä

ISBN 978-951-39-9570-6 (PDF)
URN:ISBN:978-951-39-9570-6
ISSN 2489-9003

Permanent link to this publication: http://urn.fi/URN:ISBN:978-951-39-9570-6

ABSTRACT

Väänänen, Olli
Lightweight Methods to Reduce the Energy Consumption of Wireless Sensor Nodes with Data Compression and Data Fusion
Jyväskylä: University of Jyväskylä, 2023, 53 p. (+included articles)
(JYU Dissertations
ISSN 2489-9003; 645)
ISBN 978-951-39-9570-6 (PDF)

The Internet of Things (IoT) has become part of everyday life in the last 10 years, and the intense enthusiasm for it has dissipated. Although the term "Internet of Things" is not as present at the moment, its meaning has not disappeared, but rather the reverse. Internet access devices are now ubiquitous, and the number of these devices is still increasing sharply. Each device with an internet connection can be considered an IoT device. Most of these devices include a sensor or sensors and a wireless connection to the internet. Due to the large number of devices and their location everywhere, IoT devices are often battery powered. Battery operation places demands on the power consumption of devices, as replacing or charging batteries is difficult and expensive when there are a large number of devices and when they are located in a wide area. A typical sensor application is a device for monitoring an environment that transmits data measured by sensors wirelessly at regular intervals. The power consumption of such a device should be so low that the device can run on a battery for up to years without replacing or recharging the battery.

This study focused on exploring and developing sensor data compression methods that are as light as possible and suitable for sensor nodes with light computing power. The developed methods are able to compress sensor data in real-time as new measurement values come in. Thus, the amount of data that can be transmitted wirelessly is reduced without sacrificing too much data accuracy. Wireless data transmission is known to be the single largest power consumer in such a sensor node. In addition, by combining other existing data or data that can be openly obtained from the internet, the amount of data measured by IoT devices can be reduced. It is possible to lengthen the measurement interval or reduce the number of sensor nodes themselves.

In this study, compression methods based on linear regression were developed, especially for compressing data for measuring environmental quantities. The methods developed proved to be simple, lightweight, and well suited for use in sensor nodes. The methods were shown to allow for a clear reduction in the energy consumption of the sensor node and thus an increase in its lifetime.

Keywords: Internet of Things, sensor data, compression algorithms, embedded systems, edge computing

TIIVISTELMÄ (ABSTRACT IN FINNISH)

Väänänen, Olli
Kevyet menetelmät langattomien anturisolmujen energiankulutuksen vähentämiseksi datan pakkaamisen ja datojen yhdistämisen avulla
Jyväskylä: Jyväskylän yliopisto, 2023, 53 p. (+artikkelit)
(JYU Dissertations
ISSN 2489-9003; 645)
ISBN 978-951-39-9570-6 (PDF)

Esineiden internetistä on tullut osa jokapäiväistä elämää kymmenen viime vuoden aikana, mutta samalla suurin innostus aiheeseen on laantunut. Vaikka termi esineiden internet ei ole yhtä paljon pinnalla, sen merkitys ei ole kadonnut mihinkään, vaan päinvastoin. Internet-laitteita on nyt kaikkialla, ja määrä kasvaa edelleen jyrkästi. Jokainen laite, jolla on Internet-yhteys, voidaan laskea kuuluvaksi esineiden internet -laitteisiin. Suurin osa näistä laitteista sisältää anturin tai antureita ja langattoman yhteyden Internetiin. Johtuen laitteiden suuresta määrästä ja niiden sijainnista kaikkialla, esineiden internet -laitteet ovat usein akkukäyttöisiä. Akun käyttö asettaa vaatimuksia laitteiden energiankulutukselle, koska akkujen vaihtaminen tai lataaminen on vaikeaa ja kallista, jos laitteita on paljon ja ne sijaitsevat laajalla alueella. Tyypillinen anturisovellus on ympäristön seurantaan tarkoitettu laite, joka lähettää antureiden mittaamia tietoja langattomasti säännöllisin väliajoin. Tällaisen laitteen energiankulutuksen tulisi olla niin alhainen, että laite voi toimia akulla jopa vuosia vaihtamatta tai lataamatta akkua.

Tässä tutkimuksessa keskityttiin tutkimaan ja kehittämään anturidatan pakkausmenetelmiä, jotka ovat mahdollisimman kevyitä ja soveltuvat alhaisen laskentatehon omaaviin anturisolmuihin. Kehitetyt menetelmät pystyvät pakkaamaan anturidataa reaaliajassa, sitä mukaa kuin uusia mittausarvoja tulee. Siten langattomasti lähetettävän datan määrää on mahdollista vähentää menettämättä kuitenkaan liikaa datan tarkkuutta. Langattoman tiedonsiirron tiedetään olevan suurin yksittäinen energiankuluttaja tällaisessa anturisolmussa. Lisäksi yhdistämällä muita olemassa olevia tietoja tai avoimesti Internetistä saatavia tietoja, esineiden internet -laitteiden mittaaman datan määrää voidaan vähentää. Mittausväliä on mahdollista pidentää tai itse anturisolmujen määrää vähentää.

Tutkimuksessa kehitettiin lineaariseen regressioon perustuvia pakkausmenetelmiä, erityisesti ympäristösuureiden mittausdatalle. Kehitetyt menetelmät osoittautuivat yksinkertaisiksi, kevyiksi ja soveltuivat hyvin käytettäväksi anturisolmuissa. Menetelmien osoitettiin mahdollistavan anturisolmun energiankulutuksen selkeän vähenemisen ja siten sen käyttöiän pidentämisen.

Avainsanat: esineiden internet, anturidata, pakkausalgoritmit, sulautetut järjestelmät, reunalaskenta

Author Olli Väänänen
Faculty of Information Technology
University of Jyväskylä
Finland
ORCID: 0000-0002-7211-7668

Supervisor Timo Hämäläinen
Faculty of Information Technology
University of Jyväskylä
Finland

Reviewers Pekka Toivanen
School of Computing
University of Eastern Finland
Finland

Andrey Garnaev
WINLAB
Rutgers University
USA

Opponent Susanna Pirttikangas
Faculty of Information Technology and Electrical Engineering
University of Oulu
Finland

ACKNOWLEDGEMENTS

When I started this journey to reach PhD, I didn't have a clear vision of what my dissertation would be dealing. I just thought it would be something related to the Internet of Things and embedded systems. The general topic was then found quite quickly when studying the latest research and articles related to the Internet of Things. All in all, the whole journey began slowly, then there was a time of rapid progress, until the last year and a half, things were completed slowly. But at the end, I was in no hurry to end my PhD journey. I didn't sacrifice any holidays or nights because of this, but a lot of other free time; yes, in the evenings and on weekends.

First of all, I would like to express my sincere gratitude to my supervisor, Prof. Timo Hämäläinen. He was very supportive and gave me valuable guidance, but at the same time he trusted me and let me find my own path in the field of research.

I would like to thank the Jamk University of Applied Sciences for the opportunity and support to publish my research articles related to this dissertation. I would also like to thank Ms. Tuula Kotikoski for proofreading most of the research articles in this dissertation in English.

I would like to say special thanks to my reviewers: Prof. Andrey Garnaev from Rutgers University in USA and Prof. Pekka Toivanen from the University of Eastern Finland in Finland as well as the opponent Dos. Susanna Pirttikangas from the University of Oulu in Finland. Many thanks to them for their time and efforts, and valuable comments.

I also send sincere thanks to my parents, family, and friends for their continuing support and for not asking too much about the progress of my studies.

Jyväskylä 3.5.2023
Olli Väänänen

FIGURES

CONTENTS

LIST OF INCLUDED ARTICLES

I **O. Väänänen** and T. Hämäläinen, ”Requirements for Energy Efficient Edge Computing: A Survey,” in *The 18th International Conference on Next Generation Wired/Wireless Advanced Networks and Systems NEW2AN 2018,* St. Petersburg, Russia, Aug. 2018, doi: https://doi.org/10.1007/978-3-030-01168-0_1

II **O. Väänänen**, J. Hautamäki and T. Hämäläinen, "Predictive pumping based on sensor data and weather forecast," *2019 IEEE Sensors Applications Symposium (SAS)*, Sophia Antipolis, France, 2019, pp. 1–5, doi: https://doi.org/10.1109/SAS.2019.8706018

III **O. Väänänen** and T. Hämäläinen, ”Compression Methods for Microclimate Data Based on Linear Approximation of Sensor Data," *The 19th International Conference on Next Generation Wired/Wireless Advanced Networks and Systems NEW2AN 2019,* St. Petersburg, Russia, Aug. 2019, doi: https://doi.org/10.1007/978-3-030-30859-9_3

IV **O. Väänänen**, M. Zolotukhin, and T. Hämäläinen, "Linear Approximation Based Compression Algorithms Efficiency to Compress Environmental Data Sets," In: Barolli, L., Amato, F., Moscato, F., Enokido, T., Takizawa, M. (eds) Web, Artificial Intelligence and Network Applications. WAINA 2020. Advances in Intelligent Systems and Computing, vol. 1150. Springer, Cham. doi: https://doi.org/10.1007/978-3-030-44038-1_11

V **O. Väänänen** and T. Hämäläinen, "Sensor Data Stream on-line Compression with Linearity-based Methods," *2020 IEEE International Conference on Smart Computing (SMARTCOMP)*, Bologna, Italy, 2020, pp. 220-225, doi: https://doi.org/10.1109/SMARTCOMP50058.2020.00049

VI **O. Väänänen** and T. Hämäläinen, "LoRa-Based Sensor Node Energy Consumption with Data Compression," *2021 IEEE International Workshop on Metrology for Industry 4.0 & IoT (MetroInd4.0&IoT)*, Rome, Italy, 2021, pp. 6-11, doi: https://doi.org/10.1109/MetroInd4.0IoT51437.2021.9488434

VII **O. Väänänen** and T. Hämäläinen, ”Efficiency of temporal sensor data compression methods to reduce LoRa-based sensor node energy consumption," in *Sensor Review*, vol. 42, no. 5, pp. 503–516, 2022, doi: https://doi.org/10.1108/SR-10-2021-0360

VIII **O. Väänänen** and T. Hämäläinen, ”Linearity-based Sensor Data Online Compression Methods for Environmental Applications," *6th Conference on Cloud and Internet of Things (CIoT)*, Lisbon, Portugal, 2023, pp. 149-156, doi: https://doi.org/10.1109/CIoT57267.2023.10084892

1 INTRODUCTION

The increase in Internet of Things (IoT) devices has brought the edge computing paradigm to focus in recent years. Estimates of the number of IoT devices connected to the network vary significantly. In 2021, 35 billion IoT devices were estimated to have been installed worldwide, and in 2025, more than 75 billion devices will be installed in total [1]. There is the other estimation that in 2025, 37 billion industrial IoT devices will be installed in total, but this number does not include consumer IoT devices [2]. These estimations seem to be slightly exaggerated as there are new, more modest estimations. In [3], it was estimated that there were 7.6 billion active IoT devices at the end of 2019, and this number will increase to 24.1 billion by 2030. In 2021, there were 12.2 billion connected IoT devices globally, and it was estimated to increase to 14.4 billion devices by the end of 2022 [4]. The same source estimated that in 2025, there would be 27 billion connected IoT devices globally. Overall, it is difficult to estimate the actual number of IoT devices, but the general view is that the number of IoT devices will increase significantly in the following years. This means a significant rise in market size. The IoT market size is estimated to be worth USD 384.7 billion in 2021, and it will increase to 2,464.26 billion by 2029 [5].

Most IoT devices are connected wirelessly to the network; thus, the need for reliable wireless connections is crucial. Numerous wireless technologies and protocols for use in IoT devices are available, but the drawback of wireless connections is the significant need for electrical energy for data transmission. Wireless communication is known as the major energy consumer in wireless sensor networks [6]. IoT devices are usually battery powered with limited energy resources. Many wireless technologies have been developed, particularly with low energy consumption in mind. Suitable low-energy-consumption wireless technologies for IoT devices include LoRa and SigFox [7]. Minimizing the wireless sensor node or IoT device energy consumption requires paying attention to every stage of device design and operation [6].

Edge computing is an effective and significant method to solve the problems and challenges of a large number of IoT devices produce for the reliability of wireless connections. Edge computing can also help minimize the energy consumption of IoT devices, thus increasing the lifetime of IoT devices. This is due to the reduction of the data needed to be transmitted via wireless connection. Edge computing does not mean an automatic savings in energy consumption, but when it is properly used, it is possible to achieve a reduction in energy consumption. Edge computing as a term is rather unclear, and it is sometimes described as the same as, or at least very close to, fog computing. Many sources define the term edge as an IoT end device, such as a sensor node, and the term fog as the next level from the end device in a hierarchical placement. This 'next level' refers to, for example, network devices such as gateways, base stations, and routers [8,9]. Edge computing means that at least part of the calculations and decision making takes place in the edge device; thus, not all raw data necessarily need to be transmitted via the network. This type of approach reduces the amount of data required for transmission via a wireless connection, thus also helping to reduce energy consumption in the end device. However, edge computing may require more calculations in the edge device, thus increasing energy consumption in some cases.

The largest energy consumer in wireless edge devices is the radio transmitter when it is in idle mode or transmitting [10]. The edge computing paradigm has been proven to be a suitable solution for maximizing the lifetime of battery powered IoT end devices [11]. IoT sensor nodes are usually widespread in the environment, and the number of nodes can be large. Thus, it is not suitable to power nodes from a public electricity network; instead, the nodes must be powered by batteries or through energy harvesting. This type of situation is quite typical, for example, in different agricultural applications in which IoT nodes are located in the countryside and, possibly, in fields. The replacement of empty batteries for numerous IoT nodes incurs a significant cost. Therefore, edge computing, when used correctly, is a cost-effective solution. A simple edge computing method to reduce the energy consumption of an IoT sensor node is to compress the node's raw sensor data and thus reduce the overall amount of data needed to transmit over a wireless connection [12]. It can also reduce the number of transmission cycles. Thus, the radio transmitter can be in sleep mode for a longer time.

Edge computing also significantly contributes to ensuring security and protecting privacy. If the raw data are sensitive, decision making at the edge removes the need to send sensitive data via the public internet. Such sensitive sensor data can be personal health data. However, it also gives rise to new security concerns. IoT edge devices can be connected to a large number of other IoT devices through traditional sensor networks. These IoT devices can be heterogeneous in nature, have limited resources, and utilize different routing protocols [13,14]. Some of these devices can have security vulnerabilities, and it is difficult to guarantee the security of all these devices.

One significant benefit of edge computing is shorter latency [13]. Latency is also more predictable and stable if the calculations and decision making are performed in close proximity to the edge device. Some safety-critical applications require short latency, which can be achieved with edge computing. Solutions that are safety critical and require short latency include various smart traffic applications, such as connected vehicles and even autonomous vehicles, in the future.

2 RELATED WORK

In this chapter, a review is given to provide readers with some background material on the methods used to reduce the sensor node's energy consumption, mainly with simple compression methods.

2.1 Methods to Reduce the IoT Sensor Node's Energy Consumption

Maximizing the IoT sensor node lifetime requires minimizing energy consumption in every operating phase of the device and choosing optimal hardware and network solutions. A typical wireless IoT sensor node architecture can be divided into the following subsystems: computing system (microcontroller/microprocessor, MCU), wireless communication subsystem (radio), sensing subsystem, and power supply subsystem [6].

The power performance of different MCUs has been studied widely, and several wireless communication technologies for constrained IoT devices have been developed, such as NB-IoT, LoRa, and SigFox [6,7,15]. Many IoT wireless networks operate at a sub-1 GHz frequency to achieve broader network coverage and low power consumption [16]. At sub-1 GHz frequencies, the signal is less sensitive to obstacles such as walls and buildings. These network technologies specialized for the IoT are called low-power wide-area networks (LPWANs), and they utilize star topology [17,18]. LPWANs allow communication over a kilometer distance with low energy consumption, but they have the disadvantage of a low data rate [18]. Therefore, these technologies are suitable for sensing devices that are located in a wide area and are battery powered.

A sensing subsystem is a vital part of an IoT device [19]. An IoT sensor node can include several sensors that all consume energy. This energy consumption can become a significant problem for IoT devices that have limited energy resources. Two possible solutions for this energy consumption problem are energy harvesting and energy saving [19]. As energy harvesting is rather low and weak power supply, it requires different energy-saving techniques. Energy saving techniques attempts to optimize the energy consumption of IoT devices. These methods can be utilized at different levels of device operation [19]. As the number of sensors increases in one IoT device, it results in a significant increase in energy consumption. Thus, one method to reduce sensors' energy consumption is to measure periodically and to set sensors in sleep mode between measurement periods, if possible.

A power supply subsystem requires the use of energy-efficient components for energy saving [20]. Energy-efficient components and designs use switched power supplies instead of linear regulators. Switched power supplies are more challenging to design because they can cause electromagnetic disturbances in the device itself and in the environment [21].

Therefore, energy consumption is affected by the physical sensors, central processing unit hardware, wireless network, hardware platform and all its solutions, and computation model [19]. Examples of software solutions for reducing energy consumption include the efficient use of MCU operating modes, such as sleep mode, and pre-processing data to reduce the need for transmission. Data pre-processing can be in the lightest mode only to filter erroneous or redundant data. Typically, data pre-processing includes data compression and some encryption. Data compression is used to reduce the amount of data needed to transmit; thus, it can reduce the energy consumption of the IoT device [22]. Data encryption is essential for security and privacy, but it usually requires more complex calculations, thus deriving higher energy consumption. The increase in energy consumption due to encryption is dependent on the type of encryption algorithm used. There are available encryption schemes that do not significantly affect end-node energy consumption [23,24].

Compression techniques can be broadly categorized into two categories: those that result in loss of data (lossy) and those that do not (lossless) [25]. Lossless compression does not lose any information. The original data can be recovered but at the cost of a lower Compression Ratio (CR) and more complex calculations. Thus, lossy compression algorithms are recommended for sensor data because sensor data are not accurate and sensor nodes are computationally constrained [26]. Lossy compression methods can be divided into transform-based and time domain methods. Well-known transform-based lossy compression methods include Discrete Cosine Transform (DCT), Discrete Fourier Transform (DFT), and Discrete Wavelet Transform (DWT) [27,28]. Time domain methods include data linearity-based compression techniques, such as Lightweight Temporal Compression (LTC), Piecewise Constant Approximation (PCA), Adaptive PCA (APCA), and PieceWise Linear Histogram (PWLH) [29,30].

Prediction-based methods have been widely used and have proven effective for image compression [31]. Prediction-based methods for sensor data have been also proposed [32,33]. There are both lossy and lossless methods. In [34], a lossy predictive coding -based compression algorithm for real-time data has been proposed and verified to be efficient. Although these prediction-based methods are effective, they are more complex than linearity-based methods. Therefore, they are not the subject of this dissertation.

2.2 Lightweight Temporal Compression Methods for Sensor Data

Various data compression methods have been developed for decades. Some compression algorithms are suitable for environmental data and some for physiological signals in wireless body area networks (WBAN) [35,36]. Some compression algorithms are more complex than others, and there is a significant difference in their CRs. The ability of compression methods to compress data also depends on the type of data it is used to compress [37,38]. Because IoT sensor nodes are often computationally limited and have limited energy resources, the compression methods used must be computationally light. Most compression algorithms are unsuitable for constrained sensor nodes with limited energy resources [39]. In addition, many healthcare applications in WBAN require real-time compression of one sample at a time [40]. The methods presented in this study are simple and suitable for constrained sensor nodes.

Typically, sensors measure environmental magnitudes such as temperature, humidity, air pressure, and solar radiation. These types of environmental magnitudes behave linearly on a short time scale [29]. There are simple and light compression algorithms suitable for this type of data. Most simple algorithms are based on data linearity. For example, LTC is an efficient method for compressing environmental data. It adapts and finds linear sections from consecutively measured values with a certain allowed error bound [29]. Thus, LTC and most other simple methods are called lossy methods. Lossy methods lose some information, and the reconstructed data after compression differ from the original data. The error bound represents the maximum allowable loss. Many simple linearity-based algorithms use linear regression or other simple methods to find the best-suited linear sections from measured data [30]. These linearity-based lossy compression methods are known as time-domain compression algorithms. The other category of compression algorithms is transform-based algorithms. In these algorithms, data are transformed into a different domain in which the data characteristics are represented by a limited number of coefficients obtained using a different transform method [37]. Well-known transform-based methods include DFT and DCT.

The effectiveness of a compression algorithm is evaluated based on three metrics:

1. Compression Ratio (CR),
2. Computational Complexity, and
3. Root Mean Square Error (RMSE) [30].

Most methods presented in the literature are suitable for retrospectively compressing data. If the dataset is already available, it is possible to find the best possible solution for compressing the dataset. For example, linearity-based methods can be used to find the best-suited, linearly behaving periods in a dataset to fit different linear models. This type of approach is not suitable for compressing real-time sensor data streams. If the sensor data stream is required to compress in real-time, then the inherent latency of the compression algorithm is a significant parameter [41].

Methods that are not suitable for compressing sensor data streams in real-time mode can be used in periodic sensor networks (PSN), in which data are always sent periodically to the sink [42,43]. The sending frequency describes the inherent latency of a PSN. In addition, there is always time required for algorithm computation and transmission delay. The data collected between the sending periods can be treated as any dataset. The compression algorithm can be used to compress the dataset, and only the compressed data are sent to the sink. This type of method can significantly reduce the amount of data but does not decrease the required transmission periods. Transmission periods are described in PSN, and they also describe the latency and suitability for near-real-time applications. Thus, the ability of the algorithm to reduce energy consumption in the PSN sensor node is limited to reducing the amount of data needed to send in one sending period.

Simple linearity-based methods include PCA, poor man's compression, APCA, PWLH, slide filter (SF), and LTC [29,30,44]. In the descriptions of the compression algorithms, the original sensor data are presented as data pairs

$$S = \langle (v_1, t_1), (v_2, t_2), \ldots, (v_n, t_n) \rangle,$$

where v_i is the measured sensor value (data point), and t_i is the timestamp. Normally, the sensor data are gathered with a constant measurement frequency (measurement interval, Δt), creating a dataset represented by S.

The methods PCA, APCA, PWLH, and LTC are briefly described next.

Piecewise Constant Approximation

In PCA, the entire dataset, S, is divided into fixed-length segments. This method is similar to piecewise aggregate approximation. The algorithm was explained in [30]. The length of each segment is w. The algorithm compares the largest and smallest values in each segment. If the difference between the largest and smallest values is smaller than two error bounds ($2 \cdot \varepsilon$), then all the data points in that segment are presented with one value, which is the midpoint between the highest and smallest values,

$$c_i = \frac{(v_{\max} + v_{\min})}{2}$$

Therefore, each raw data value is within an error bound distance from the compressed value. If the data values do not fit the error bound limits, then the original data values remain in the compression output. The compressed data are presented as

$$\mathrm{PCA}(S) = \langle (c_1, f_1), (c_2, f_2), \ldots, (c_k, f_k) \rangle,$$

where f_i is a Boolean value indicating that the value c_i is a constant value (compressed) or a raw value (not compressed due to values falling out from error bound limits) [30].

An example of PCA algorithm results is shown in Figure 1. The data are real temperature measurement data with a 10-min measurement interval. The temperature is measured in degrees Celsius, with a resolution of 0.1 degrees. The raw data contain 30 temperature sensor values (blue circles). The error bound used in PCA compression is 0.5 °C, and the length of segment w is 5 measurement values. In Figure 1, the solid constant five-measurement long lines present the compressed constant value. Thus, the first five original values (timestamps 1–5) are presented with one value, which is −7.1 °C. From 6 to 10, the constant value is −7.85°. The values between 11 and 15 vary more than two error bounds in total. Therefore, these values do not fit in the constant segment and are presented as original values. In this example, the original dataset contained 30 temperature values, and with a PCA algorithm with a 0.5 °C error bound, the compressed dataset contains 14 values and 14 Boolean f_i values. In addition, the compressed dataset can also include the timestamps, but it is also possible to derive the timestamps from the compressed data, as the length of the constant segments is known.

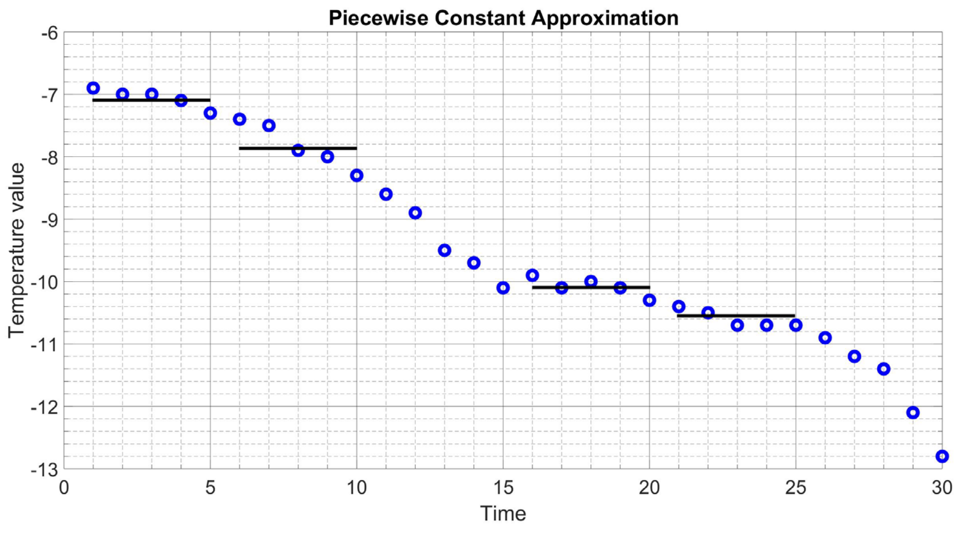


FIGURE 1 PCA for the test dataset

The PCA algorithm is simple but also limited in its compression efficiency. This algorithm is suitable if the entire dataset or at least one fixed-segment length is available. Segment length describes the algorithm’s inherent latency and theoretical maximum CR. The PCA algorithm is appropriate for data that are almost constant or change only slightly. If the values change rapidly in one direction, as is often the case with ambient temperature, the method is not effective. The theoretical maximum for CR is w because if every value in the dataset fits into one of the fixed linear sections (length of w), then n/w linear sections exist (n is the number of data points in the original dataset), and thus the number of values is the same in the compressed dataset. Subsequently,

$$\mathrm{CR} = \frac{n}{\frac{n}{w}} = w.$$

In addition, n/w Boolean values are required.

The PCA algorithm has variations, such as the cache filter, poor man's compression (mean), and poor man's compression (mid-range), which is the same compression algorithm as the PCA [45].

Adaptive Piecewise Constant Approximation

APCA is an advanced version of PCA. In some sources, this method is called PCA. However, in this study, PCA has fixed-length constant segments, as presented previously, while in APCA, the window size w varies to achieve the best possible CR [46]. Thus, APCA is rather similar to PCA.

The APCA algorithm starts from the first value in the dataset and continues to subsequent values one by one continuously, comparing the values with each other as long as $v_{\max} - v_{\min} \leq 2\varepsilon$. When the difference between the largest and smallest values exceeds the error bound requirements, the last value is removed from the constant linear segment. The constant linear segment value is calculated as in PCA as the middle value between the largest and smallest values. For the next segment, the APCA algorithm begins the comparison again, starting from the value that falls out of the previous linear segment [30].

The results of the APCA (with an error bound $\varepsilon = \pm 0.5$ ℃) for the test dataset are shown in Figure 2. The test dataset is the same as that used for PCA. The original dataset of 30 values is reduced to five constant values with varying lengths as a result of the compression.

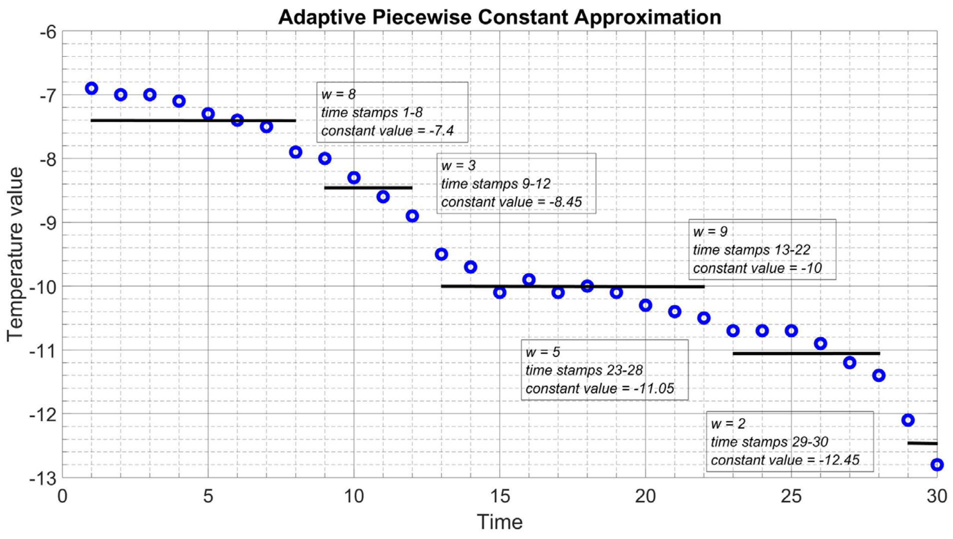


FIGURE 2 APCA for the test dataset

Similar to PCA, APCA also works well if the values remain stable, but when the values change periodically in one direction, the compression efficiency is limited. As a result of the APCA compression algorithm, the original dataset is presented as a sequence of k data segments presented as a pair of values c_i

(constant value) and τ_i (timestamp of the last data point in the segment). The compressed data are presented as

$$\text{APCA}(S) = \langle(c_1, \tau_1), (c_2, \tau_2), \dots, (c_k, \tau_k)\rangle.$$

Each constant-value segment varies in length, and the size of the segment can be as short as one data point [30,46]. Owing to the unpredictable length of the segments, the inherent latency of the APCA algorithm is unknown and depends on the dataset characteristics. If the data do not vary significantly and remain for long periods within the error bound requirement, the CR will be high; however, as a drawback, the latency will also be high [44]. APCA is unsuitable for real-time applications due to its unpredictable latency.

The APCA algorithm is a simple method and is thus well suited for constrained sensor nodes. Nevertheless, its effectiveness in compressing data is limited, and there are more effective but slightly more complex algorithms available. In APCA, a single segment costs $b_s + b_{tp}$ space to store, where b_s is the size of the constant value (individual data points in the original data are of the same size), and b_{tp} is the size of the timestamp. The CR in the total size is

$$\text{CR}_{\text{size}} = \frac{n(b_s + b_{tp})}{k(b_s + b_{tp})},$$

where n is the number of data points in the original dataset, and k is the number of constant segments after compression [46]. Thus, the overall CR in the data points is $\text{CR} = \frac{n}{k}$.

Piecewise Linear Histogram

PWLH is similar to APCA, but the linear segments do not need to have constant values. The algorithm finds the best-fit linear segments from the original data that meet the error bound requirements. The PWLH algorithm uses linear regression to find the best-fit linear segment starting from data point 1 and continuing as long as the difference between each individual data point remains within the error bound from the regression line. In the compressed dataset, each linear line is presented with three parameters: c_i^b, which is the beginning value of the linear segment; c_i^e, which is the end value of the segment; and τ_i, which is the timestamp of the end point of the linear segment. Thus, the compressed dataset is presented as

$$\text{PWLH}(S) = \langle(c_1^b, c_1^e, \tau_1), (c_2^b, c_2^e, \tau_2), \dots, (c_k^b, c_k^e, \tau_k)\rangle,$$

see [30,47].

In Figure 3, the PWLH algorithm is used for the same test data as for PCA in Figure 1 and for APCA in Figure 2. The solid black lines represent the linear segments when the error bound is 0.5 °C. The raw dataset can be modeled with two linear lines.

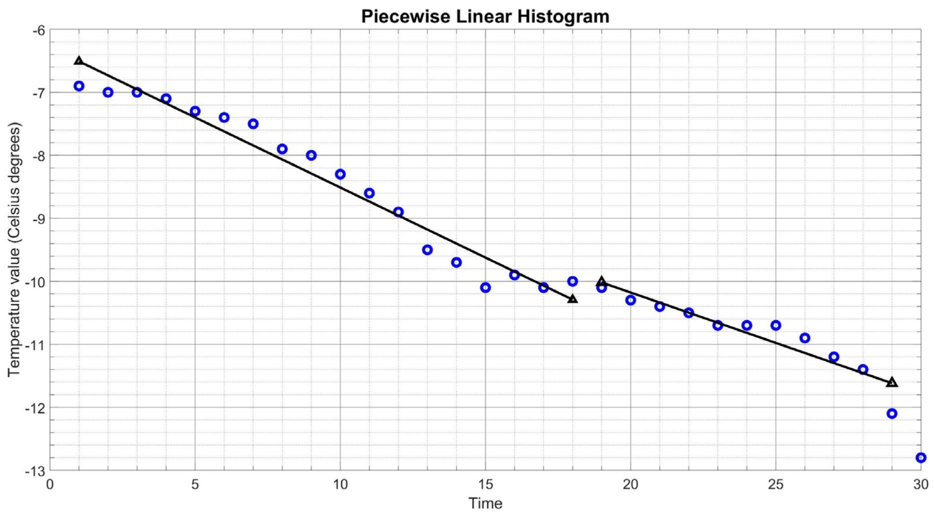


FIGURE 3 PWLH for the test dataset

The linear regression line must be calculated from three values to as many values as there are in each final linear segment. Many calculations are required if the linear segment is long. The computational complexity is not as low as that of other simple methods. In addition, the inherent latency becomes high and unpredictable when the linear segments are long. Unpredictable and long inherent latency makes the algorithm inappropriate for use in online streaming and in real-time or near-real-time applications. The basic idea of finding the best-fit linear line to represent a dataset and maximize the CR has been used in many linearity-based compression algorithms. Examples of slight variations of the idea are bounded-error piecewise linear approximation, SF, swing filter, and swing-RR [30,45,48,49].

Lightweight Temporal Compression

LTC is a well-known simple compression algorithm. It was first presented in [29], but a similar algorithm, Fan, was actually presented previously in [50] for electrocardiogram (ECG) data. LTC is a powerful compression algorithm, especially for environmental data that behave quite linearly when the observation time window is short. The CR depends on the data properties and the error bound used. LTC can achieve a CR as high as 20 when compressing environmental temperature data [29].

The functionality of the LTC algorithm is illustrated in Figure 4. The linear model starts with the first measured value ($v_1 = 20.0, t_1 = 1$) as a starting point (the first value in the compressed dataset, $\langle c_1, \tau_1 \rangle$). When the next value ($v_2 = 20.3, t_2 = 2$) is received, the limit lines (upper and lower dashed lines) are drawn to the new value of $\pm\varepsilon$ (20.8 and 19.8 at t_2, as the error bound ε is 0.5 °C) (Figure 4 (a)). When the next value is received, the limit lines are tightened if the value extremes with error bounds ($v_i \pm \varepsilon$) are located inside the limit lines (Figure 4 (b) and (c)). If the new value with error bound extremes falls outside the limit lines, then the linear section ends in the timestamp of the previous value with the value at the midpoint of the limit lines (Figure 4 (d); red solid line end point in timestamp 4). The algorithm then starts again, with the compressed value (c_2, τ_2) in timestamp 4 as the starting point. The compressed dataset is

$$\mathrm{LTC}(S) = \langle (c_1, \tau_1), (c_2, \tau_2), \dots, (c_k, \tau_k) \rangle.$$

In Figure 4, the first two data pairs in the compressed dataset are (20, 1) and (20.65, 4).

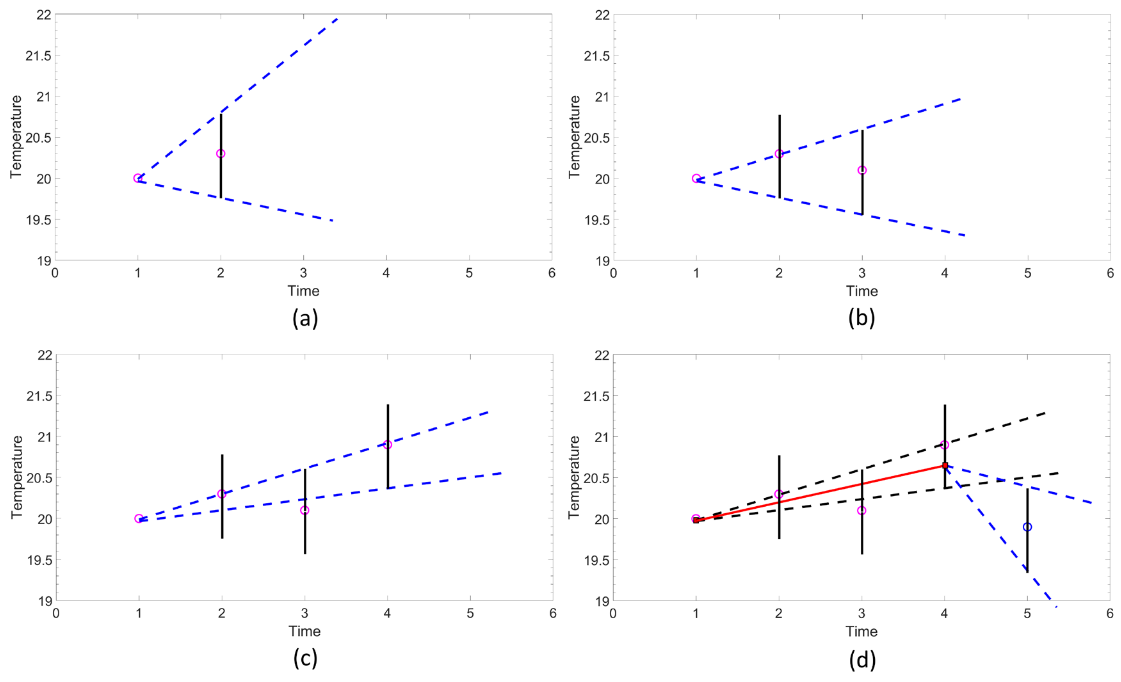


FIGURE 4 LTC algorithm

The efficiency of the LTC algorithm is shown in Figure 5, in which the LTC algorithm is used to compress the test dataset (temperature in degrees Celsius). With an error bound $\varepsilon = \pm 1.0$ °C the entire test dataset fits into the linear section (red solid line). If the error bound is lowered to ± 0.5 °C, the test dataset is presented with four linear lines (black dashed lines). Therefore, the compressed dataset has five value pairs.

The LTC has unpredictable latency, and it depends on each linear section length. The higher the CR, the longer the latency. If the data behave linearly, a long latency is derived. When the new linear segment begins, the starting point is known, but the direction of the following values remains unknown until the linear segment ends, and the end point is stored in the compressed dataset.

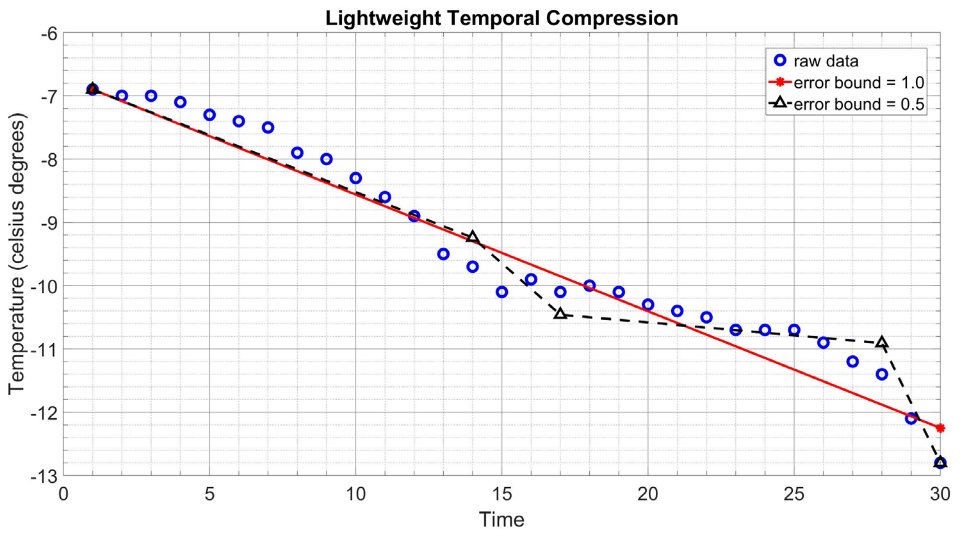


FIGURE 5 LTC for the test dataset

Some slight variations in the original LTC algorithm have been developed. In [51], a slight modification of the LTC algorithm was used. In this modified version, the out-of-bound (first value capped off from the linear segment) value becomes the starting point for a new linear segment. This modified version is not as compression efficient as the original LTC because the end point of the linear segment is not the starting point of the next segment, as it is in the original version. Other variations include adaptive lightweight temporal compression [52], refined lightweight temporal compression (RLTC) [53], multidimensional extension of the LTC method [54], direct lightweight temporal compression (DLTC) [55], and DFan [56]. These modified versions were developed to either minimize the data reconstruction error or improve compression efficiency. For example, DLTC can improve the data reconstruction error, but it is achieved at the cost of a lower CR. RLTC can improve the CR but at the cost of a more complex algorithm.

3 MOTIVATION AND RESEARCH PROBLEM

As the use of IoT is evolving quickly, the energy efficiency of IoT devices has become a significant factor. As the number of IoT devices can be very large, the cost of one IoT device cannot be high. Thus, the technologies, components, and solutions used in most IoT devices need to be simple and cost effective. IoT devices typically refer to battery powered wireless devices, including sensors. For cost efficiency and low power consumption, the microcontrollers used are simple and computationally constrained. Moreover, memory resources are limited. Sensor nodes are usually located in places where electricity grids are not available. Thus, the devices are usually battery powered, and energy consumption needs to be minimized to lengthen the lifetime of the devices. Changing the empty batteries of numerous IoT devices located in a wide area incurs a significant cost.

The hardware technology solutions used need to be optimized for low energy consumption. For example, specific low-power wireless connections developed for the IoT are available. The use of software solutions is as important as the use of optimal hardware solutions. With suitable software solutions, it is possible to achieve significant energy savings on IoT sensor devices. Developing lightweight sensor data compression algorithms can help reduce sensor node energy consumption. Compression algorithms help minimize the amount of data required to send over a wireless connection to the network. This can reduce either the number of transmitting periods or the amount of data transmitted in one transmission. These compression algorithms need to be appropriate for compressing the data achieved from sensors, and the compression should be done in online mode for sensor data stream. Compression algorithms should be effective for compression and computationally light to be suitable for computationally constrained sensor nodes. CR, inherent latency, and reconstruction error are important aspects of compression algorithms. The efficiency of compression algorithms in compressing sensor data depends on the type of data. For example, different environmental magnitudes behave differently from ECG data, and some compression algorithms are more effective for certain types of data than others. There is no general lightweight compression algorithm that suits every type of data, application, and platform. The compression algorithm used is a compromise between CR for certain types of data, inherent latency, and data accuracy. Moreover, computational complexity is a concern.

In this study, simple, lightweight, and easy-to-implement sensor data compression algorithms, mainly for compressing environmental magnitudes, are evaluated and developed. Compression algorithms have been evaluated and developed to minimize inherent latency and to efficiently compress specifically environmental magnitudes that behave linearly in a short time window. Algorithms are lightweight and easy to implement in constrained sensor nodes. The effect on the sensor node's energy consumption is evaluated using real experiments.

Compression algorithms have been developed for decades, but most are not suitable for compressing data in online mode. Many algorithms are also complex and computationally heavy. Simple and light sensor data compression algorithms have not been focused on in recent years.

Using compression algorithms in the sensor node itself can be considered edge computing. In edge computing, at least some of the calculations are made in close proximity to the data source, such as a sensor. Thus, the compression of sensor data can be considered edge computing. Sensors are fundamental components of IoT devices, as the whole concept of the IoT is related to the data. However, achieving data with sensors and sensor nodes is not always the most efficient and cost-effective method. It can be useful to enrich the data obtained from sensors with other available data. Other data can be open data available on the internet. Weather data are common open data available on the internet to be used in environmental solutions. Combining open data with sensor data can be more effective than relying on sensor data alone. It can also be used to reduce the number of IoT sensor nodes or sensors in one IoT node. Weather data can be weather observations or weather forecast data. Combining open data to enrich measured sensor data is also evaluated in this study.

Overall, the main idea of this study is to find simple solutions to reduce and minimize the energy consumption of an IoT sensor node using lightweight and easy-to-implement solutions. Methods should be suitable for use in constrained IoT sensing devices with a wireless network connection.

4 OVERVIEW OF INCLUDED ARTICLES

Article I is a general overview of the different aspects of energy-efficient edge computing. It describes the main subjects related to the topic and guides the overall studies covered in the other articles. It also serves as a general introduction to the topics of this dissertation and describes the importance of the topics covered in the other articles. All the other articles deal with specific sensor data analysis topics. Article II focuses on utilizing sensor data, together with other available data, to make decisions. In this case, other data are the different local weather observation data and weather forecast data. The other data used to enrich the measured sensor data can be open data available from the internet, other data that can be restricted, or company confidential data. Articles III–VIII deal with linearity-based compression algorithms and their utilization and performance to compress sensor data efficiently and with low energy consumption. The main idea of Articles III–VIII is to compress environmental sensor data efficiently in real-time using simple and computationally light compression algorithms. One of the main targets is to reduce energy consumption with sensor data compression and to lengthen the lifetime of battery powered IoT sensor nodes. The algorithms' inherent latency is also a significant concern.

Article II presents how sensor data can be combined with other available data. The other data are open data available from the internet and more restricted data that are not openly available for everyone. Other data have been used to enrich the data gathered with sensors. Not all information needs to be measured individually, but data already available from other sources can be used. It may also lighten the need for accurate and continuous sensor measurement and thus can reduce energy consumption overall, even though energy consumption was not a concern in the pilot case presented in the article.

Articles III and V present new versions of linearity-based compression algorithms suitable for sensor nodes. Methods are developed specially for compressing sensor data streams in online mode. Article V uses the results of Article III, and the main consideration is to minimize the algorithms' inherent latency. The algorithms developed in Article III are further developed in Article

V. Article IV also uses the results obtained in Article III. The algorithms developed in Article III are evaluated in Article IV using multiple datasets with varying error bounds to determine whether there is a correlation between the characteristics of the dataset and the CR obtained using a particular linearity-based compression algorithm. The correlation found in Article IV can be used to evaluate the performance of a certain compression algorithm to compress a dataset with certain characteristics. In Articles VI and VII, the compression algorithms developed in Articles III and V are implemented on a sensor node that utilizes a LoRa network to transmit the data to the cloud. The effect of the algorithms on the energy consumption of the sensor node is evaluated and tested. In Article VIII, the real-time linear regression-based temporal compression (RT-LRbTC) algorithm presented in Article V is developed further. Thus, two new versions of the algorithm are developed, and their performance is evaluated.

4.1 Article I: Requirements for Energy Efficient Edge Computing: A Survey

O. Väänänen and T. Hämäläinen, "Requirements for Energy Efficient Edge Computing: A Survey," in *The 18th International Conference on Next Generation Wired/Wireless Advanced Networks and Systems NEW2AN 2018.* St. Petersburg, Russia, Aug. 2018, doi: https://doi.org/10.1007/978-3-030-01168-0_1.

Overview of Article I

Article I surveys publications on different edge computing methods from the perspective of energy consumption. The purpose of Article I is to go through the current situation of the different methods for reducing overall energy consumption in IoT end devices or in the so-called edge devices. It deals with the fundamentals of energy-efficient edge computing and the many different aspects of edge devices and edge computing. Different challenges, benefits, and advantages are discussed in the article.

The article discusses the different aspects of the following terms: IoT, edge computing, and fog computing. The benefits and challenges between edge/fog computing and cloud computing are compared. According to the literature, cloud computing can cause challenges in different latency critical IoT applications [57]. Applications that are sensitive to latency include smart traffic and intelligent transportation systems, autonomous vehicles, and virtual and augmented reality applications. Various applications related to safety-critical and sensitive data may not rely on cloud connection. Cloud computing cannot meet the quality of service requirements for latency and safety-critical systems. In smart transportation and vehicle systems, it is possible to use vehicle-to-vehicle connections or data achieved from vehicles to avoid collisions. As such an analysis is time and latency critical, it is impossible to rely on cloud computing. These decisions must be made either locally or in close proximity at a Cloudlet [58]. An overview of the general IoT architecture is illustrated in Figure 6 [59].

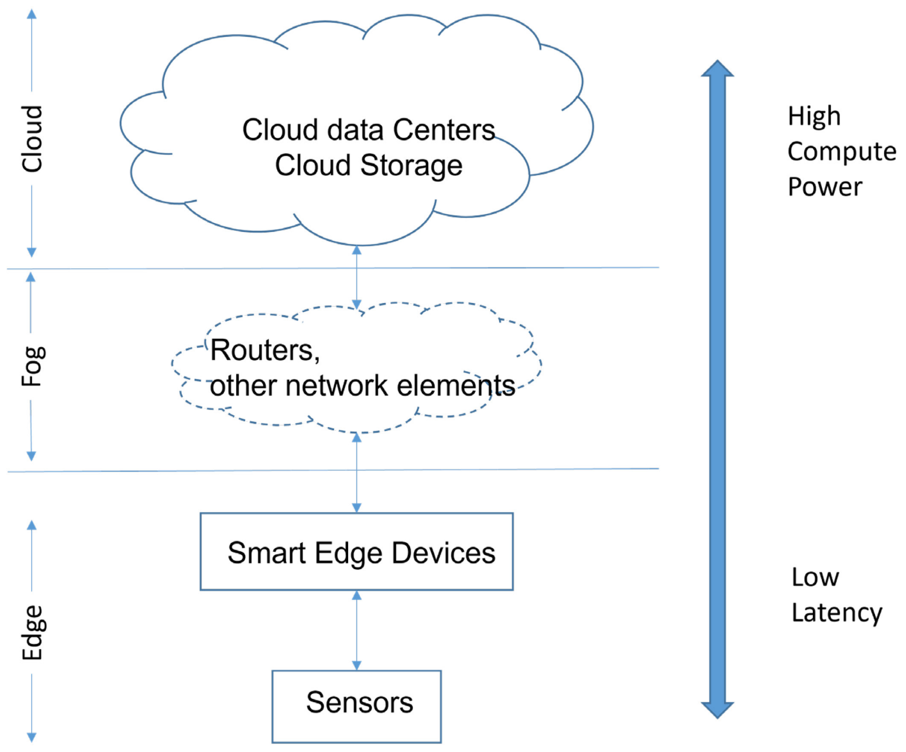


FIGURE 6 Edge and fog architecture in IoT [59].

The edge refers to the close proximity of the sensor and the sensor node. An edge device usually refers to the sensor node itself. A sensor node is commonly wirelessly connected to the network gateway, and it includes a microcontroller, which is constrained and limited by memory. As the number of edge devices can be large, edge devices are usually battery powered. This nature of the IoT sensor node pushes for the requirements for the energy efficiency of these end nodes. Sending data wirelessly is recognized as the most energy-intensive task for wireless IoT devices. Changing the batteries of a large number of IoT nodes can be costly. Thus, maximizing battery lifetime can be a cost-effective solution.

The term fog refers to the next level in a network hierarchy, as shown in Figure 6. Different network elements, such as gateways and routers, can be described as fog devices. Typically, fog devices are computationally more powerful than edge devices. As they are normally mains-powered devices, energy efficiency is not so much a critical aspect in fog devices.

One great challenge in the utilization of edge and fog computing is the heterogeneous nature of devices. It is difficult to develop generic solutions to be used in different edge devices, which can have different operating systems, wireless connection protocols and technologies, and hardware platforms. Many simple sensor nodes are based on simple embedded microcontrollers, while some edge devices may be based on Linux-based single-board computers, such as Raspberry Pi. In addition to different platforms, applications also vary; thus, there are no generic solutions available for different IoT devices. For more powerful single-board computers, it is possible to use virtualization or a container-based approach. This kind of approach makes it possible to run the same applications with different platforms and operating systems.

Various methods of reducing the energy consumption of wireless sensor networks are effective, as sending data wirelessly is the most energy-consuming activity of a wireless node. Article I lists many energy-efficient routing algorithms used in traditional wireless sensor networks. These routing algorithms are not suitable for IoT sensor networks. In IoT sensor networks, the end nodes can be distributed in a large area; thus, the nodes are not wirelessly connected to each other, such as in a traditional sensor network. That kind of IoT sensor network utilizes a star-type protocol in which the nodes are directly connected to the base station (gateway), which can be located at a long distance. To solve this challenge, several IoT network technologies have been developed, such as LoRa, SigFox, and NB-IoT. All of these have long ranges, use a star topology, and have low energy consumption. As a drawback to low energy consumption, the data rate is limited in these wireless technologies but sufficient for transmitting sensor data.

As wireless transmission and receiving are the most energy-intensive activities in a wireless sensor node, reducing the transmitted data is an effective method to reduce energy consumption in the IoT node and thus lengthen the lifetime of the battery powered device. Article I lists and surveys several data compression methods suitable for constrained IoT sensor nodes. Data compression needs to be conducted on the edge. As IoT sensor nodes have a simple microcontroller with limited computational power and limited memory, these impose requirements for the compression algorithm used. The compression algorithm should be able to compress the sensor data stream in online mode. The compression methods listed in Article I are lossy methods. The methods can be categorized into two groups: time domain and transform domain. The effectiveness of the compression method to compress data depends on the characteristics of the data. Therefore, there is no universal compression algorithm suitable and effective for every data type. LTC is a popular light and effective time domain compression algorithm for sensor data. It is especially suitable for environmental data compression, such as temperature and humidity.

Article I also briefly discusses the different IoT protocols. The typical communication data protocols are MQTT, CoAP, XMPP, and AMQP. MQTT and CoAP are the most appropriate for constrained IoT devices.

As IoT devices are connected to the internet, security is an important issue. The data transmitted via the internet can be sensible and need to be kept in secret. One solution is to encrypt transmitted data. The encryption method should be light because IoT devices are computationally constrained. Lightweight encryption algorithms have been developed and are available. A potential lightweight asymmetric encryption algorithm called Aaβ is presented in [60], and it is compared with the traditional RSA encryption algorithm in [61].

Author's Contribution to Article I

Olli Väänänen proposed the general topic, and Article I served as a survey to find more detailed topics for further research. Olli Väänänen wrote and revised the manuscript. Timo Hämäläinen supervised the study and revised the manuscript.

4.2 Article II: Predictive pumping based on sensor data and weather forecast

O. Väänänen, J. Hautamäki and T. Hämäläinen, "Predictive pumping based on sensor data and weather forecast," *2019 IEEE Sensors Applications Symposium (SAS)*, Sophia Antipolis, France, 2019, pp. 1–5, doi: https://doi.org/10.1109/SAS.2019.8706018.

Overview of Article II

An IoT-based system for controlling pumping from a water reservoir while drying peat bog has been developed. The IoT system that controls pumping makes decisions based on several sensors and other data. The data used in decision making are local water-level data, local rainfall measurements, and weather forecasts obtained from the Finnish Meteorological Institute (FMI) open data. The water pump is driven by a frequency converter, which can control the pump's rotational speed in stepless mode. The IoT system is used to control the frequency converter in such a way that pumping can be started before the water level in a reservoir reaches the trigger level. In Article II, the sensor data (water level) have been enriched with other data. Other data that have been used are open data (weather forecasts) and local weather observation data from separate weather services (not open data).

Drying peat bog is important in peat production. Water is directed from the peat bog through ditches to the local water reservoir. The water is then pumped from the reservoir to the filtering field. The filtering field is a large area with small trees, plants, and other vegetation. Water is filtered through this field before it enters nature and water bodies. Water needs to be filtered because otherwise, it will be contaminated with solids and other substances, which can weaken the water quality in water bodies. The filtering field filters more effectively if the water flow is even. Traditionally, the pumping period starts when the water level reaches a certain level in the water reservoir and stops when the water level reaches the lower limit level. Pumping occurs at nominal speed, thus causing the water rush in, bursting toward the filtering field. This burst can be seen as an uneven water flow after the field. Between pumping periods, the water flowing to the bodies of water can be minor.

The overall pump control system is presented in Figure 7. A water level transmitter has a traditional 4–20 mA current signal, which is used in automation technology due to its robustness against electromagnetic interference. This mA signal is converted to a digital I2C-bus signal, which can be connected to the

Raspberry Pi platform. The Raspberry Pi sends the water level information continuously through 3G/4G connection to the ThingsBoard.io IoT platform, which operates in the cloud server. ThingsBoard.io is an open-source IoT platform that can be used to visualize different IoT data and simple data analysis. The local weather station sends the data to the Weatherlink cloud server. The ThingsBoard.io platform obtains local weather data from Weatherlink. The weather forecast is obtained from the FMI open data service. The FMI updates the weather forecast every 6 h. This study is interested only in rainfall forecasts.

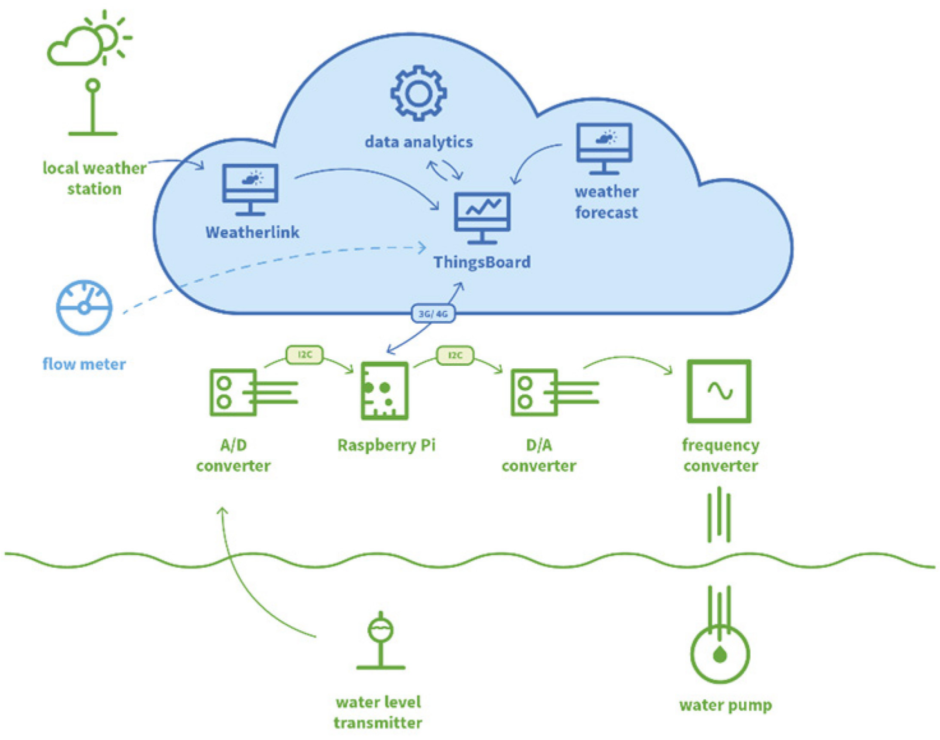


FIGURE 7 The IoT-based pumping control system.

The algorithm developed for controlling the frequency converter and pump is simple. The algorithm inputs are the rainfall forecast, the last hour of accumulated rainfall, and the actual water level. The overall system creates a manipulated current signal (4–20 mA) to control the frequency converter. If the water level is low or if no rainfall has been detected during the last hour and no rainfall is predicted, then the created current signal is the same received from the water level transmitter. If the water level is over a certain trigger level, and if the local weather station has detected enough rain during the last hour and/or the weather forecast predicts rainfall during the next hour, the current signal value increases, and the pumping may start in anticipation if the current signal created is sufficiently high.

One challenge is the uncertainty of weather forecasts. The retrospect between the real rainfall data from the local weather station and the weather forecast data demonstrates a low correlation. Thus, the algorithm may predict pumping due to predicted rainfall and start pumping even if rain has never started.

Results of Article II

In the pilot case, the algorithm's parameters were the best estimates because no previous measurement data were available. The algorithm parameters were tuned during the pilot in autumn 2018. The system proved to be a potential one to even out the pumping, but the weather conditions during the piloting period were not normal. Rainfall was minimal during that autumn, and tuning the algorithm further was not possible. The basic operation of the IoT control system functioned well, and with fine tuning of the algorithm's parameters, the system could be used to even out pumping when the rain had just started or when rain was coming. The system utilizes local sensor data and open data from the internet and combines the data with the algorithm to make decisions.

Author's Contribution to Article II

Olli Väänänen developed the system-level functionality of the IoT pumping system, developed the control algorithm, participated in constructing the IoT system, participated in analyzing the results, and wrote and revised the manuscript. Jari Hautamäki participated in developing the IoT pumping system, writing, and revising the manuscript, and supervising the pilot case. Timo Hämäläinen supervised the entire study and revised the manuscript.

4.3 Article III: Compression Methods for Microclimate Data Based on Linear Approximation of Sensor Data

O. Väänänen and T. Hämäläinen, "Compression Methods for Microclimate Data Based on Linear Approximation of Sensor Data," *The 19th International Conference on Next Generation Wired/Wireless Advanced Networks and Systems NEW2AN 2019.* St. Petersburg, Russia, Aug. 2019, doi: https://doi.org/10.1007/978-3-030-30859-9_3.

Overview of Article III

This article evaluates the ability of different simple sensor data compression algorithms to compress environmental sensor data and to further develop certain compression algorithms. Using computationally light compression algorithms to compress the sensor data stream in an edge device, it is possible to reduce the number of radio transmitting periods. The radio transmitter is the most energy-intensive component of a wireless sensor node while transmitting. With the right compression algorithm with an acceptable reconstruction error, the energy consumption of the end device can be reduced, and data accuracy can be maintained at an acceptable level at the same time.

Linear regression-based compression algorithms have been developed and presented in the literature [37,45,62,63]. Most of the compression algorithms presented in the literature are suitable for compressing a certain dataset that is already available. In this study, the main idea is to compress sensor data value by value when new data values are available. One well-known and suitable algorithm for compressing the sensor data stream is LTC. LTC was first introduced in [29], but a similar compression algorithm was already presented several years earlier called Fan for ECG data [50]. In Article III, the authors present a new version of the linear regression-based algorithm called linear regression-based temporal compression (LRbTC) and its modification, called modified linear regression-based temporal compression (M-LRbTC). These regression-based algorithms were developed to compress sensor data stream value by value in online mode. These algorithms are also simple and computationally light.

The ability of LTC and M-LRbTC algorithms to compress real environmental sensor data is evaluated and compared with the DCT method. The microclimate datasets used are from the FMI open data service. The datasets are from the whole-year 2018 values for temperature, humidity, and wind speed from the Salla Naruska measurement station. The measurement interval in the datasets is 10 minutes. Datasets with 1-h measurement intervals are tested for comparison. The algorithms' ability to compress datasets is evaluated using MATLAB simulation. Even though full datasets are available, data compression is performed value by value as the data are measured in online mode. The ability of compression algorithms to compress data is evaluated using CR, with different reconstruction errors (error bound) allowed.

Results of Article III

LTC proved to be the most effective compression algorithm among the tested ones for environmental data. The developed M-LRbTC algorithm was more effective than DCT when the DCT had a window size of 5 values. The M-LRbTC benefits from increasing the *N* value from 3 to 4 or 5 but falls behind the LTC significantly. For the tested temperature data with a 10-min measurement interval and error bound $\varepsilon = 0.5$ °C, the CR values were 3.9–4.8 for M-LRbTC and 9.5 for LTC. For the air pressure data with a 10-min interval and error bound $\varepsilon = 0.5$ hPa, the CR values were 8.94–10.75 for M-LRbTC and 28.22 for LTC. For the wind speed data with 10-min average values and error bound $\varepsilon = 0.5$ m/s, the CR values were 2.62–3.18 for M-LRbTC and 5.09 for LTC. Thus, the CR values were significantly higher for air pressure data than for the other data types because air pressure data behave more linearly with the chosen error bound.

In Article III, the idea of modifying the M-LRbTC further to be more suitable for real-time sensor stream compression is presented. The latency of the algorithm could be improved by sending the regression line coefficients with a timestamp, not the actual values. The receiver knows that the future values continue to follow the regression line with the allowed reconstruction error as long as the new regression line parameters are not received. This kind of modified version is developed, evaluated, and presented in Article V.

Author's Contribution to Article III

Olli Väänänen proposed the topic, developed the new algorithms, programmed the algorithms using MATLAB, evaluated the algorithms, and wrote and revised the manuscript. Timo Hämäläinen supervised the study and revised the manuscript.

4.4 Article IV: Linear Approximation Based Compression Algorithms Efficiency to Compress Environmental Data Sets

O. Väänänen, M. Zolotukhin, and T. Hämäläinen, "Linear Approximation Based Compression Algorithms Efficiency to Compress Environmental Data Sets," In: Barolli, L., Amato, F., Moscato, F., Enokido, T., Takizawa, M. (eds.) Web, Artificial Intelligence and Network Applications. WAINA 2020. Advances in Intelligent Systems and Computing, vol. 1150. Springer, Cham. doi: https://doi.org/10.1007/978-3-030-44038-1_11.

Overview of Article IV

In Article IV, the effect of dataset characteristics on linearity-based algorithms' ability to compress effectively is evaluated. The ability of different compression algorithms to compress data depends on the data characteristics. For example, environmental microclimate data differ greatly from ECG data, which have periodical behavior. Data from an acceleration sensor in moving vehicles (cars, motorcycles, etc.) or in wearable well-being devices can be unpredictable and have sudden changes. There is no compression algorithm that suits every type of data.

In this article, the effect of dataset characteristics on the ability of various linearity-based compression algorithms to compress data is evaluated. The idea is to examine whether the dataset characteristics can be used to estimate the CR achieved with a certain compression algorithm. Even though the sensor values are not known in advance, the environmental values of a certain geological location behave similarly year by year. Thus, it is possible to expect that future values have behaviors similar to history values.

The evaluated linearity-based algorithms are LTC and LRbTC (i.e., M-LRbTC, which is presented in Article III) with N = 3, 4, and 5. DCT is also evaluated as a comparison. The CR of the compression algorithms is evaluated for different dataset characteristics. The datasets used are temperature, air pressure, and wind speed data. The datasets used are taken from the FMI open data service. All environmental magnitudes measured are from the 2018 Salla Naruska measurement station data. The whole-year 2018 datasets with a 10-min measurement interval are used, and they are divided into monthly datasets with 10-min, 20-min, 30-min, 40-min, 50-min, and 1-h measurement intervals. A whole-year dataset is also used with the same measurement intervals. In total, 78 different datasets are used for each environmental magnitude.

The dataset characteristics evaluated are the average absolute change between consecutive measurements (AC) and the standard deviation of the change between consecutive measurements (SD). The error bounds used are 0.5 °C for the temperature datasets, 0.5 hPa for the air pressure datasets, and 0.5 m/s for the wind speed datasets. The wind speed datasets are 10-min average values.

The compression algorithms are programmed using MATLAB, and both AC and SD values are evaluated for each dataset. The correlation between the CR and the AC and SD values is examined from the graphs, and the trend lines are determined using MATLAB. The trend line chosen is visually the best-fit polynomial regression line. The ability of the polynomial regression line to estimate the CR for the dataset with a certain AC or SD is estimated using the norm of the residual value, which is the measure of the goodness of the line fit for the data values.

Results of Article IV

Correlation was clearly seen in the graphs between the AC and SD values with CR for all types of environmental data tested. Generally, the AC value estimated the CR better than the SD value for all types of data tested. This was visually observed in the graphs and in the norm of the residual values. For the temperature data, the best-fit trend lines were eighth-degree polynomials; for the air pressure data, the best-fit trend lines were fourth-degree polynomials; and for wind speed data, the best-fit trend lines were also fourth-degree polynomials. The best-fit trend lines were chosen by visually estimating the fit together with the norm of the residual values.

Article IV shows that it is possible to predict the CR of the selected compression algorithm for data with certain AC and SD values. The correlation was better between AC and CR than between SD and CR. The AC value is also easier to calculate than the SD value. The results of Article IV can be used to select a suitable compression algorithm and to estimate the lifetime of a battery powered sensor node using a certain compression algorithm. The strongest correlation between the quality factors and CR was observed in the wind speed data and the worst in the air pressure data. The CR was the highest for air pressure data and the lowest for wind speed data with each compression algorithm. These results are for certain error bounds selected.

Author’s Contribution to Article IV

Olli Väänänen proposed the topic, conducted the evaluations using MATLAB, evaluated the results, and wrote and revised the manuscript. Mikhail Zolotukhin revised and recommended improvements to the manuscript and its structure. Timo Hämäläinen supervised the study and revised the manuscript.

4.5 Article V: Sensor Data Stream On-line Compression with Linearity-based Methods

O. Väänänen and T. Hämäläinen, "Sensor Data Stream on-line Compression with Linearity-based Methods," *2020 IEEE International Conference on Smart Computing (SMARTCOMP).* Bologna, Italy, 2020, pp. 220–225, doi: https://doi.org/10.1109/SMARTCOMP50058.2020.00049.

Overview of Article V

The linearity-based compression algorithms presented in the literature are best suited for datasets that are already available or for sensor data systems that are not latency-critical. In Article V, the compression algorithms are compared by inherent latency and CR. A new version of LRbTC, called RT-LRbTC, is presented. It is a variation of the M-LRbTC algorithm that significantly improves latency. Therefore, it is more suitable for applications that have latency requirements.

Some temporal compression algorithms, such as PCA, APCA, and PWLH, have been presented and explained. These algorithms are simple but do not have predictable latency. The linear regression-based algorithms (i.e., LRbTC and M-LRbTC) presented in previous articles are also explained, and their inherent latency is described. LTC is used for comparison, but it has the drawback of having a non-predictable latency, which is long if the CR is high.

The short latency and predictability of RT-LRbTC are based on calculating the new regression line from the sensor values already available. The RT-LRbTC algorithm uses the three last available values ($N = 3$) to calculate the regression line. M-LRbTC needs to wait for new measurement values when a new regression line is required, adding to the inherent latency. The output of the RT-LRbTC algorithm is the parameters of the regression line (slope and base) and the timestamp of the starting point of the line. When the line ends (value fall off the regression line by more than one error bound), the new line parameters are immediately calculated and stored or transmitted to the sink. Thus, only one transmission cycle per regression line is required, compared with the two transmitting periods needed in the M-LRbTC algorithm.

In Article V, the performance of the selected algorithms is evaluated by compressing real environmental datasets. The magnitudes of the datasets are temperature, air pressure, and wind speed. The CRs of the algorithms are evaluated using MATLAB with different error bounds. The error bounds are 0.1–2.0 °C for the temperature dataset, 0.1–2.0 hPa for the air pressure dataset, and 0.1–2.0 m/s for the wind speed dataset.

Results of Article V

The compression efficiency of RT-LRbTC was lower than that of M-LRbTC because the compressed data had more regression lines. However, RT-LRbTC benefited from the fact that only one transmitting period was required for each regression line, so its overall efficiency was better than that of the M-LRbTC algorithm. LTC was used for comparison, but it is not suitable for compressing the online sensor data stream because it has non-predictable latency. The latency is also long if the algorithm compresses the data stream efficiently. LTC had superior CR for every dataset tested. RT-LRbTC proved to be an efficient compression algorithm with a short and fixed latency, thus making it suitable for solutions that have requirements for latency.

RT-LRbTC was originally developed to achieve short inherent latency. RT-LRbTC has the shortest latency of the presented algorithms. Its latency in the linear section and when calculating the new regression line is one measurement interval long (Δt). The RT-LRbTC algorithm is simple and computationally light, suitable for compressing environmental sensor data streams in online mode.

Author's Contribution to Article V

Olli Väänänen proposed the topic, developed a new version of the algorithm, conducted the evaluations using MATLAB, examined the results, and wrote and revised the manuscript. Timo Hämäläinen supervised the study and revised the manuscript.

4.6 Article VI: LoRa-Based Sensor Node Energy Consumption with Data Compression

O. Väänänen and T. Hämäläinen, "LoRa-Based Sensor Node Energy Consumption with Data Compression," *2021 IEEE International Workshop on Metrology for Industry 4.0 & IoT (MetroInd4.0&IoT)*, Rome, Italy, 2021, pp. 6–11, doi: https://doi.org/10.1109/MetroInd4.0IoT51437.2021.9488434.

Overview of Article VI

In Article VI, linear regression-based compression methods are implemented on an embedded sensor board, which has LoRa connectivity to the network. The embedded board is an Arduino MKR WAN 1310, which has an Arm Cortex-M0+-based microcontroller and a Murata LoRa module. A DHT22 temperature and humidity sensor is connected to the board. The LoRa network used is a commercial LoRa network in Finland operated by Digita. The compression algorithms implemented on the embedded board are LTC and RT-LRbTC. RT-LRbTC is tested with two versions, one with $N = 3$ and the other with $N = 4$. The embedded board is programmed to measure the temperature value with a DHT22 sensor periodically and to apply the chosen compression algorithm to the sensor data stream value by value when new values are measured. If the data need to be transmitted after the measurement and compression period, the LoRa connection is used to transmit the compressed data value to the network. Between the measurement and compression periods, the device is programmed to go in deep sleep mode.

The energy consumption of the embedded module is measured in different operational phases using a digital multimeter (DMM) and a digital oscilloscope.

The DMM is used to measure current consumption in deep sleep mode. The supply voltage of the battery is simultaneously measured with an oscilloscope. These measured values are used to calculate the power consumption in deep sleep mode. Current consumption during active periods cannot be measured with a DMM because the current value changes during the active period, depending on the operation phase of the board. The current consumption and power consumption are measured with and without a compression algorithm to determine whether the algorithm calculations have any effect on power consumption. The different operational and measurement scenarios are the deep sleep phase, the sensor measurement and algorithm calculations phase, and the LoRa transmission phase. Energy consumption of the sensor measurement and algorithm calculation phase is measured using the current probe of the oscilloscope. The current probe is set to measure the current in the battery wire, and the other oscilloscope channel is used to measure the supply voltage in the board's battery connector with a voltage probe. Energy consumption is measured using the oscilloscope's math functions to calculate the overall energy consumed in the oscilloscope screen time. The energy consumption of deep sleep is also measured with the same setup and subtracted from the total energy consumption, resulting the measurement and algorithm phase's energy consumption only. The energy consumption of the LoRa transmission is measured similarly, except that the previously measured measurement and energy consumption of the algorithm are also subtracted from the result, giving only the energy consumption of the LoRa transmission. The energy consumption of LoRa transmission consists of the energy consumption of the transmitting uplink and the receiving downlink. As the LoRa transmitting setup is not adjusted, the uplink uses only the Spreading Factor 10, and the downlink varies between the Spreading Factor 9 and 12. Energy consumption is measured in both cases.

Total energy consumption comprises the energy used in deep sleep mode, the energy used for measurement and algorithm processing, and the energy consumed during LoRa transmission. The measurement and algorithm phase appears at regular intervals. The LoRa transmission phase appears when there is a need to send the results of the compression to the sink. Thus, the algorithm's CR affects when and how often the transmission phase appears. The total energy consumption is approximated using the following equation:

$$W_{\mathrm{tot}} = P_{\mathrm{ds}} t_x + \frac{t_x}{\Delta t} W_M + \frac{t_x}{\mathrm{CR}\Delta t} W_s,$$

where P_{ds} is the deep sleep power consumption, t_x is the overall time, Δt is the measurement interval, W_M is the measurement and algorithm phase energy consumption, W_s is the LoRa transmission event energy consumption, and CR is the CR of the algorithm implemented for the measured data (approximation).

If the overall energy available is known, such as battery capacity, then it is possible to estimate the device lifetime by solving the overall time t_x from the previous equation.

Results of Article VI

Every measurement was repeated at least 10 times to improve the reliability of the measurement results; thus, the average values were used. Deep sleep current consumption was 117 µA, which is not a low value for a modern microcontroller sensor board. As the supply voltage was 3.99 V, the power consumption in deep sleep was 0.46683 mW. The energy consumption of the measurement and algorithm phase varied approximately at 4.6–4.8 mWs, depending on the compression algorithm implemented with or without an algorithm. Energy consumption was the lowest without an algorithm implemented, but the measurement inaccuracy for the low-level currents using an oscilloscope current probe was so low that the results were close to each other. Thus, it was impossible to draw a difference between the results. The difference in the results between the different algorithms and those without the algorithm was negligible. The LoRa transmission energy consumption varied slightly between the compression algorithm implemented with and without the algorithm. Using an uplink SF10 and downlink SF9, the total energy consumption was the lowest for LTC (61.68 mWs) and the highest for RT-LRbTC, N = 4 (73.14 mWs), but the difference between the algorithms was, again, negligible. Using the uplink SF10 and the downlink SF12, the lowest value was found in LTC (108.80 mWs) and the highest in RT-LRbTC, N = 3 (113.40 mWs). The results varied from measurement to measurement more than the differences between the algorithms and those with no algorithm implemented. Therefore, the calculations of the algorithm did not show any significant effect on energy consumption.

For example, Article VI presents a scenario with a 2,000 mAh battery lifetime with a measurement interval of 10 min. The results showed that RT-LRbTC could extend the device's lifetime by 25.2%. With a rather long measurement interval, the device's deep sleep energy consumption becomes determining for the device lifetime.

Author's Contribution to Article VI

Olli Väänänen proposed the topic, implemented the algorithms for the embedded sensor board, conducted the measurements, analyzed the results, and wrote and revised the manuscript. Timo Hämäläinen supervised the study and revised the manuscript.

4.7 Article VII: Efficiency of temporal sensor data compression methods to reduce LoRa-based sensor node energy consumption

O. Väänänen and T. Hämäläinen, "Efficiency of temporal sensor data compression methods to reduce LoRa-based sensor node energy consumption," *Sensor Review*, Vol. 42 No. 5, pp. 503–516, 2022, doi: https://doi.org/10.1108/SR-10-2021-0360.

Overview of Article VII

Article VII is based on Article VI and is a comprehensive study of the topic. The measurement setup is improved to be more accurate, and different LoRa setup variations are evaluated and measured. This study evaluates the possible reduction in the overall energy consumption of the LoRa sensor node using linearity-based compression algorithms. By compressing sensor data in online mode, LoRa transmission periods can be reduced, thus decreasing overall energy consumption.

The same compression algorithms as in Article VI are implemented on an Arduino MKR WAN 1310 evaluation board. The implemented compression algorithms are LTC and RT-LRbTC, with $N = 3$ and 4, respectively. A situation without a compression algorithm implemented is used for comparison. The Arduino MKR WAN 1310 has LoRa-connectivity and uses a lithium–polymer battery as power supply. The DHT22 sensor is used to measure the ambient temperature. The test setup is illustrated in Figure 8.

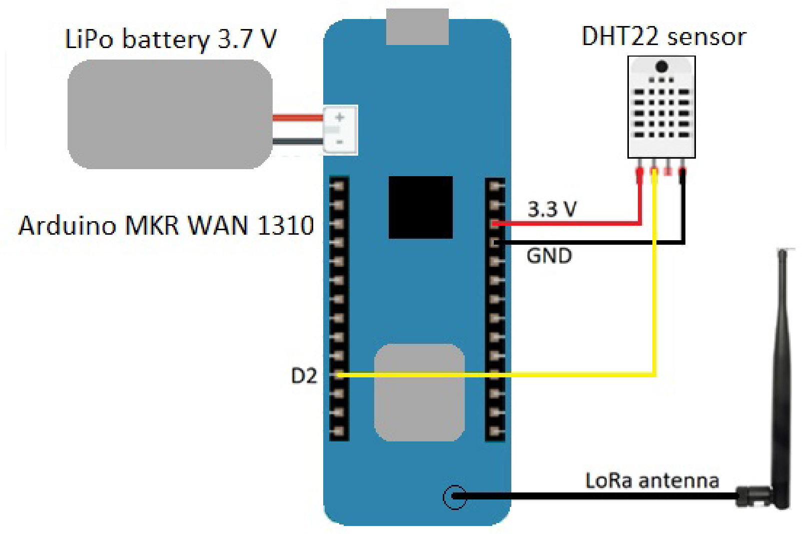


FIGURE 8 Arduino MKR WAN 1310 setup.

The active mode current consumption is measured using a shunt resistor method in which a resistor of a known value is in a series in supply. By measuring the voltage across the resistor, it is possible to calculate the current from Ohm's law equation, $I = U/R$. The voltage across the shunt resistor can be measured with an oscilloscope using a differential voltage probe or two normal voltage probes. In this study, the resistor's voltage is measured with two active voltage probes. The shunt resistor is 10 Ω. The shunt resistor measurement is more accurate than the current probe measurement used in Article VI. The shunt resistor measurement circuit is presented in Figure 9. The current can be obtained from the measured values as follows:

$$I = \frac{V_1 - V_2}{R}$$

The power consumption can be calculated from the current value if the supply voltage is known. Supply voltage is measured with an oscilloscope passive probe. The power consumption of the device is as follows:

$$P = V_{\text{supply}} I$$

V_{supply} is the same as V_2 but measured separately in this study.

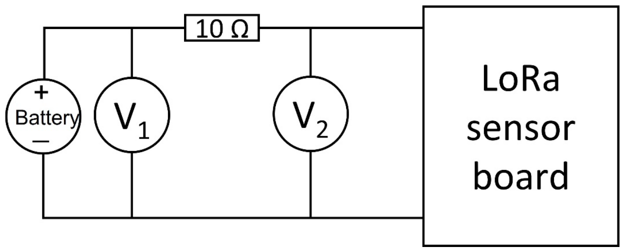


FIGURE 9 Shunt resistor circuit.

Deep sleep current consumption is measured with a DMM because it is more accurate for µA level measurements than the shunt resistor method. The sensor measurement process and the algorithm's energy consumption with and without the implemented algorithms are measured using the shunt resistor method. The energy consumption of the LoRa transmission is measured with every Spreading Factor (SF7–SF12) setting using the shunt resistor setup.

Results of Article VII

The deep sleep power consumption P_{ds} was 0.46683 mW, similar to Article VI. The sensor measurement and the algorithm's energy consumption varied between 4.88 mWs and 4.98 mWs (average values). The difference between the algorithms and no algorithm implemented was negligible. Thus, the compression algorithm calculations were not adding the energy consumption. The LoRa transmission energy consumption depended on the SF used. The energy consumption in the transmission was the highest when both the uplink and downlink used SF12, and the downlink was received. Energy consumption was the lowest with the uplink SF7, and the downlink was not received.

The average power consumption of the LoRa sensor node was estimated from the measured results and with certain CRs for each compression algorithm. The CRs of the compression algorithms were taken from Article V. CR = 10 was used for LTC and CR = 6 for the RT-LRbTC algorithms. The downlink was not received every time, even though it was sent from the network side. Receiving the downlink message consumes more energy, but it is the normal scenario and should be used when estimating the device's power consumption and overall lifetime. The average power consumption is the highest when both the uplink and the received downlink use SF12. The power consumption reduction with the compression algorithm is the highest with the uplink SF12 and downlink SF12. With the uplink SF7 and the received downlink SF9, the difference in power consumption between the algorithms and no implemented algorithms is small.

The lifetime of the LoRa sensor node was evaluated with a 2,000 mAh battery. In the worst-case scenario, the device always transmits with an SF12 setting, and the SF12 downlink message is received. The device lifetime without a compression algorithm implemented was 343 days if the measurement interval was 10 min. If the compression algorithm were implemented, then the lifetime would be 562 days with RT-LRbTC and 596 days with the LTC algorithm. If the device was transmitted with a lower SF, the lifetime would increase, and the effect of the algorithms would not be as significant.

The study shows that simple compression algorithm calculations do not add to the device's energy consumption but can significantly decrease overall energy consumption by reducing wireless transmission periods. With a long measurement interval, deep sleep consumption has become the determining factor for device lifetime.

Author's Contribution to Article VII

Olli Väänänen proposed the topic, implemented the algorithms on an embedded sensor board, performed the measurements, analyzed the results, and wrote and revised the manuscript. Timo Hämäläinen supervised the study and revised the manuscript.

4.8 Article VIII: Linearity-based Sensor Data Online Compression Methods for Environmental Applications

O. Väänänen and T. Hämäläinen, "Linearity-based Sensor Data Online Compression Methods for Environmental Applications," *6th Conference on Cloud and Internet of Things (CIoT),* Lisbon, Portugal, 2023, pp. 149-156, doi: https://doi.org/10.1109/CIoT57267.2023.10084892.

Overview of Article VIII

In this study, some well-known computationally lightweight temporal compression algorithms are evaluated in terms of the CR and reconstruction error quality metrics. The evaluated and tested algorithms are LTC, M-LRbTC with different N values, RT-LRbTC, and two new versions derived from RT-LRbTC. The new versions, RT-LRbTC-2Δt and RT-WLRbTC-2Δt, are presented in this article. Special attention is given to the inherent latency of the algorithms and the suitability of algorithms to compress sensor data in online mode. The algorithm efficiency is evaluated using the CR. The reconstruction error is evaluated using the RMSE values. The performance of the algorithms is tested using real environmental data from two different weather stations: one located in Lapland, Finland, and the other in the southernmost part of Finland. The environmental datasets are the temperature, air pressure and wind speed datasets for the whole year of 2019. All datasets are measured at 10-min measurement intervals. The results show that the method with the best compression efficiency also has the highest reconstruction error, with a certain error bound. Moreover, the algorithm with the best CR (LTC) has the most unpredictable and the longest inherent latency; thus, it is not suitable for online sensor data stream compression. The two new algorithms developed and presented improve the compression efficiency compared with the original RT-LRbTC but at the cost of a longer but still predictable inherent latency. They provide a usable compromise between their CR and inherent latency compared with the original RT-LRbTC algorithm.

RT-LRbTC-2Δ*t* is similar to RT-LRbTC, except that the new regression line is calculated with one value from a previous linear line (the last value that fits in the previous line) and two new values. Thus, waiting for one measurement interval is required to obtain the second new value, adding inherent latency to 2Δ*t* in total. The main idea is to wait for one new value, as it is expected to predict future values better than already available previous values and to maintain the inherent latency in a reasonably short time.

The RT-WLRbTC-2Δ*t* algorithm is similar to RT-LRbTC-2Δ*t*, but it utilizes a weighted linear regression instead of a normal linear regression. Certain values are weighted in a weighted linear regression. In this algorithm, the last value is used twice, giving it a double weight. Again, the main idea is the expectation that the latest value predicts future values better than the already available previous values.

Results of Article VIII

LTC had the best compression efficiency, but because of its unpredictable latency, it was not suitable for compressing sensor data streams if there were requirements for inherent latency. It also had the highest *RMSE* values for a certain error bound.

Two new algorithm versions, RT-LRbTC-2Δ*t* and RT-WLRbTC-2Δ*t*, presented better performance than RT-LRbTC, but the improvement was not significant, at least for the datasets used. Using the weighted linear regression did not lead to any better performance compared with the normal linear regression. The newly developed algorithms benefited from the fact that they required only one transmitting period for each linear regression line. Overall, the results for the linear regression-based algorithms were similar.

The performance of the M-LRbTC algorithm was improved by increasing the *N* from 3 to 4 or 5. This led to more complex calculations and added inherent latency when calculating the new regression line.

The RMSE values were similar for each linear regression-based algorithm. Overall, all the compression algorithms showed the best performance for air pressure data. The air pressure dataset characteristics predicted this because the average change (AC) and standard deviation (SD) values were smaller than those for the temperature and wind speed data.

The original RT-LRbTC had the shortest latency. The new algorithms had higher inherent latency but slightly better performance for the datasets used. Thus, the two new algorithms can be used as a compromise between short latency and compression performance.

Author's Contribution to Article VIII

Olli Väänänen proposed the topic, developed new versions of the algorithm, conducted the evaluations using MATLAB, evaluated the results, and wrote and revised the manuscript. Timo Hämäläinen supervised the study and revised the manuscript.

5 DISCUSSION AND CONCLUSIONS

This study proves that it is possible to significantly reduce IoT sensor node energy consumption using simple edge software solutions. The compression algorithms developed are based on an old idea about the linear behavior of environmental magnitudes. The algorithms developed attempt to minimize the inherent latency to be suitable for online compression and near-real-time applications.

In Article V, a new version of the LRbTC algorithm called RT-LRbTC is presented. The RT-LRbTC algorithm has an inherent latency of one measurement interval (Δt) long and demonstrates similar compression efficiency as the other linear regression-based compression algorithms. LTC has a superior CR, but due to its unpredictable latency, it is not suitable for near-real-time applications that require more constant and known latency.

Linear regression-based compression algorithms are also called lossy algorithms, and their compression efficiency is related to the requirement of accepted reconstruction error. A typical reconstruction error (error bound) accepted for temperature data can be 0.5 °C, for example. With an error bound $\varepsilon = \pm 0.5$ °C, RT-LRbTC obtains a CR of up to 7.3 for the Hanko Tulliniemi temperature dataset, as shown in Article VIII. Data were measured at 10-min measurement intervals. Thus, the number of transmission periods can be reduced to 1/7 from a raw data situation.

Different variations of linear regression-based compression algorithms are easy to develop, but their performance does not change much. In compressing sensor data in online mode, it is not possible to anticipate the future values and thus the compression algorithm's performance with certainty. However, as environmental magnitudes behave typically for certain geological places, the previous datasets can be used to predict different algorithm behaviors for future values. These behaviors are utilized and evaluated in Article IV.

RT-LRbTC has the shortest inherent latency of the evaluated and developed compression algorithms. Its effectivity to lengthen the typical IoT sensor node lifetime is evaluated in Articles VI and VII. In Article VI, the node lifetime is lengthened from 491 days (no compression) to 616 days with RT-LRbTC, when the board is powered by a 2,000 mAh battery. The result will be even more significant if the hardware platform used is more energy efficient. The Arduino MKR WAN 1310 LoRa platform has a deep sleep power consumption of 0.467 mW, which is not very low. The measured deep sleep current consumption of 116 μA is not very low for a modern microcontroller-based platform. As shown in Article VII, in the worst-case scenario (LoRa using SF12 for both uplink and downlink), the node lifetime was lengthened from 343 days (no compression implemented) to 562 days (RT-LRbTC implemented) using a 2,000 mAh battery.

The other developed variations of the LRbTC algorithm are RT-LRbTC-2Δ*t* and RT-WLRbTC-2Δ*t*. RT-LRbTC-2Δ*t* is similar to RT-LRbTC, but it has an inherent latency of two measurement intervals. When the previous linear line ends, it waits for one measurement interval to obtain a new measurement value and then uses the *N* latest values to calculate the regression line. The idea behind this version is that if the direction of the values changes and the line ends, then the latest values that are already out of the previous line can predict future values better. This same idea is taken further in the RT-WLRbTC-2Δ*t* algorithm, which is similar to RT-LRbTC-2Δ*t* but utilizes a weighted linear regression instead of a traditional linear regression. In a weighted linear regression, the last measured value is weighted to create a greater impact on the linear regression calculation. RT-LRbTC-2Δ*t* and RT-WLRbTC-2Δ*t* demonstrated some improvements in the CR compared with the RT-LRbTC algorithm for the temperature, air pressure, and wind speed datasets. The weighted linear regression-based variation did not show any better performance than the traditional linear regression-based version.

Overall, this study shows that it is possible to achieve significant energy savings with simple compression algorithms in the wireless sensor node, together with the efficient use of sleep modes. Simple compression algorithms based on linear regression can obtain good CRs for environmental sensor data. There are more efficient compression algorithms available, but these methods can be more complex or may not have predictable latency. Simple linear regression-based compression algorithms did not show any rise in energy consumption due to the calculations in the experiment with a sensor node. Moreover, if the previous history datasets are available for certain geological locations for environmental magnitudes, these datasets can be used to predict certain compression algorithm performances to compress the same magnitudes with the same measurement setup.

YHTEENVETO (SUMMARY IN FINNISH)

Tässä tutkimuksessa kehitettiin kevyitä anturidatan pakkausmenetelmiä, jotka soveltuvat yksinkertaisiin ja alhaisen laskentatehon omaaviin langatonta Internet-yhteyttä käyttäviin anturisolmuihin. Tällaiset anturisolmut ovat usein akkukäyttöisiä, ja siten energian kulutuksen minimointi on tärkeää laitteen toimintaiän pidentämiseksi. Akkujen vaihtaminen tai lataaminen on merkittävä kustannus, jos anturisolmujen määrä on suuri ja ne sijaitsevat laajalla alueella.

Väitöskirjassa tutustuttiin ensin tyypillisiin energiatehokkaan reunalaskennan menetelmiin ja niiden vaatimuksiin. Näiden selvitysten pohjalta on keskitytty pääasiassa kehittämään pakkausalgoritmeja, jotka pystyvät pakkaamaan anturidataa reaaliajassa sitä mukaa kuin uusia mittausarvoja tulee. Tutkimuksen tuloksena on esitelty useita versioita lineaariseen regressioon perustuvista pakkausalgoritmeista, joiden lähtökohtana on ollut algoritmin aiheuttaman viiveen vakiointi ja pienentäminen.

Tutkimuksessa on lisäksi testattu datan ominaisuuksien vaikutusta kehitettyjen pakkausalgoritmien pakkaustehokkuuteen. Löydettyjen vaikutusten avulla on mahdollista ennustaa algoritmin pakkaustehokkuutta tietynlaisen datan pakkaamiseen. Ympäristösuureet kuten lämpötila, ilmanpaine ja tuulennopeus käyttäytyvät tietylle mittauspaikalle tyypillisellä tavalla vuodesta toiseen, jolloin historiadataa voidaan käyttää ennustamaan algoritmin kykyä tulevan mittausdatan tehokkaaseen pakkaamiseen.

Kehitettyjä algoritmeja on testattu pääasiassa simuloimalla, mutta myös implementoimalla algoritmeja sulautettuun anturisolmuun. Anturisolmu hyödynsi langatonta LoRaWAN-verkkoa anturidatan lähettämiseen Internetiin. Anturisolmun energiankulutusta mitattiin sekä hyödyntäen pakkausalgoritmeja että ilman niitä. Mittaukset osoittivat kehitettyjen pakkausalgoritmien hyödyntämisen vähentävän anturisolmun energiankulutusta merkittävästi ja siten akkukäytössä pidentävän sen toimintaikää huomattavasti.

REFERENCES

[1] G. D. Maayan, The IoT Rundown For 2020: Stats, Risks, and Solutions [Online], January 2020, available: https://securitytoday.com/Articles/2020/01/13/The-IoT-Rundown-for-2020.aspx?Page=1

[2] S. Woodford, Industrial Revolution 4.0 - The Future of IIoT [Online], November 2020, available: https://www.juniperresearch.com/document-library/white-papers/industrial-revolution-4-the-future-of-iiot

[3] Transforma Insights, Global IoT market will grow to 24.1 billion devices in 2030, generating $1.5 trillion annual revenue [Online], May 2020, available: https://transformainsights.com/news/iot-market-24-billion-usd15-trillion-revenue-2030

[4] IoT Analytics, State of IoT 2022: Number of connected IoT devices growing 18% to 14.4 billion globally [Online], May 2022, available: https://iot-analytics.com/number-connected-iot-devices/

[5] Fortune Business Insight, Internet of Things (IoT) Market Size, Share & COVID-19 Impact Analysis, By Component, By End-use Industry, and Regional Forecast, 2022–2029 [Online], March 2022, available: https://www.fortunebusinessinsights.com/industry-reports/internet-of-things-iot-market-100307

[6] V. Raghunathan, C. Schurgers, Sung Park and M. B. Srivastava, "Energy-aware wireless microsensor networks," in *IEEE Signal Processing Magazine*, vol. 19, no. 2, pp. 40–50, March 2002, doi: https://doi.org/10.1109/79.985679

[7] É. Morin, M. Maman, R. Guizzetti and A. Duda, "Comparison of the Device Lifetime in Wireless Networks for the Internet of Things," in *IEEE Access*, vol. 5, pp. 7097–7114, 2017, doi: https://doi.org/10.1109/ACCESS.2017.2688279

[8] Y. Ai, M. Peng, and K. Zhang, "Edge computing technologies for Internet of Things: a primer," *Digital Communications and Networks*, vol. 4, issue 2, pages 77–86, 2018, doi: https://doi.org/10.1016/j.dcan.2017.07.001

[9] W. Shi, J. Cao, Q. Zhang, Y. Li and L. Xu, "Edge computing: vision and challenges," in *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, Oct. 2016, doi: https://doi.org/10.1109/JIOT.2016.2579198

[10] D. C. Harrison, D. Burmester, W. K. G. Seah, and R. Rayudu, "Busting myths of energy models for wireless sensor networks," in *Electronics Letters*, vol. 52, no. 16, pp. 1412–1414, 2016, doi: https://doi.org/10.1049/el.2016.1591

[11] J. Mocnej, M. Miškuf, P. Papcun, and I. Zolotová, "Impact of Edge Computing Paradigm on Energy Consumption in IoT," in *IFAC PapersOnLine*, 51(6), pp. 162–167, 2018, doi: https://doi.org/10.1016/j.ifacol.2018.07.147

[12] D. I. Săcăleanu, R. Popescu, I. P. Manciu and L. A. Perişoară, "Data Compression in Wireless Sensor Nodes with LoRa," *2018 10th International*

Conference on Electronics, Computers and Artificial Intelligence (ECAI), Iasi, Romania, 2018, pp. 1–4, doi: https://doi.org/10.1109/ECAI.2018.8679003

[13] A. M. Alwakeel, "An Overview of Fog Computing and Edge Computing Security and Privacy Issues," *Sensors (Basel Switzerland)*, vol. 21, (24), pp. 8226, 2021, doi: https://doi.org/10.3390/s21248226

[14] S. Oh, Y. Seo, E. Lee and Y. Kim, "A Comprehensive Survey on Security and Privacy for Electronic Health Data," *International Journal of Environmental Research and Public Health*, vol. 18, (18), pp. 9668, 2021, doi: https://doi.org/10.3390/ijerph18189668

[15] L. Bao, L. Wei, C. Jiang, W. Miao, B. Guo, W. Li, X. Cheng, R. Liu and J. Zou, "Coverage Analysis on NB-IoT and LoRa in Power Wireless Private Network," *Procedia computer science*, vol. 131, pp. 1032–1038. 2018, doi: https://doi.org/10.1016/j.procs.2018.04.252

[16] Z. Yang, A. Ghubaish, D. Unal and R. Jain, "Factors Affecting the Performance of Sub-1 GHz IoT Wireless Networks," *Wireless Communications and Mobile Computing*, vol. 2021, 2021, doi: https://doi.org/10.1155/2021/8870222

[17] J. P. Queralta, T. Gia, Z. Zou, H. Tenhunen and T. Westerlund, "Comparative Study of LPWAN Technologies on Unlicensed Bands for M2M Communication in the IoT: Beyond LoRa and LoRaWAN," *Procedia computer science*, vol. 155, pp. 343–350. 2019, doi: https://doi.org/10.1016/j.procs.2019.08.049

[18] B. Foubert and N. Mitton, "Long-Range Wireless Radio Technologies: A Survey," *Future Internet*, vol. 12, no. 1, pp. 13, 2020, doi: https://doi.org/10.3390/fi12010013

[19] Y. Liang, X. Wang, Z. Yu, B. Guo, X. Zheng, and S. Samtani, "Energy-efficient Collaborative Sensing: Learning the Latent Correlations of Heterogeneous Sensors," *ACM Transactions on Sensor Networks*. vol. 17, no. 3, pp. 1–28, 2021, doi: https://doi.org/10.1145/3448416

[20] M. Szymczyk and P. Augustyniak, "Selected Energy Consumption Aspects of Sensor Data Transmission in Distributed Multi-Microcontroller Embedded Systems," *Electronics (Basel)*, vol. 11, no. 6, pp. 848, 2022, doi: https://doi.org/10.3390/electronics11060848.

[21] A. Subramanian and U. Govindarajan, "Analysis and mitigation of EMI in DC–DC converters using QR interaction," *IET Circuits, Devices & Systems*, vol. 11, pp. 371–380, 2017, doi: https://doi.org/10.1049/iet-cds.2016.0288

[22] M. Mishra, S. G. Gourab and X. Gui, "Investigation of Energy Cost of Data Compression Algorithms in WSN for IoT Applications," *Sensors (Basel, Switzerland)*, vol. 22, no. 19, pp. 7685, 2022, doi: https://doi.org/10.3390/s22197685

[23] I. Froiz-Míguez, T. M. Fernández-Caramés, P. Fraga-Lamas and L. Castedo, "Design, Implementation and Practical Evaluation of an IoT Home Automation System for Fog Computing Applications Based on

MQTT and ZigBee-WiFi Sensor Nodes," *Sensors (Basel, Switzerland)*, vol. 18, no. 8, pp. 2660, 2018, doi: https://doi.org/10.3390/s18082660

[24] N. Abosata, S. Al-Rubaye and G. Inalhan, "Lightweight Payload Encryption-Based Authentication Scheme for Advanced Metering Infrastructure Sensor Networks," *Sensors (Basel, Switzerland)*, vol. 22, no. 2, pp. 534, 2022. doi: https://doi.org/10.3390/s22020534

[25] U. Jayasankar, V. Thirumal and D. Ponnurangam, "A survey on data compression techniques: From the perspective of data quality, coding schemes, data type and applications," *Journal of King Saud University. Computer and information sciences*, vol. 33, no. 2, pp. 119–140, 2021, doi: https://doi.org/10.1016/j.jksuci.2018.05.006

[26] C. Chen, L. Zhang and R. L. K. Tiong, "A new lossy compression algorithm for wireless sensor networks using Bayesian predictive coding," *Wireless Networks*, vol. 26, no. 8, pp. 5981–5995, 2020, doi: https://doi.org/10.1007/s11276-020-02425-w

[27] A. Moon, J. Park and Y. J. Song, "Prediction of Compression Ratio for Transform-based Lossy Compression in Time-series Datasets," *2022 24th International Conference on Advanced Communication Technology (ICACT)*, 2022, pp. 142–146, doi: https://doi.org/10.23919/ICACT53585.2022.9728954

[28] H. Wu, M. Suo, J. Wang, P. Mohapatra and J. Cao, "A Holistic Approach to Reconstruct Data in Ocean Sensor Network Using Compression Sensing," in *IEEE Access*, vol. 6, pp. 280–286, 2018, doi: https://doi.org/10.1109/ACCESS.2017.2753240

[29] T. Schoellhammer, B. Greenstein, E. Osterweil, M. Wimbrow and D. Estrin, "Lightweight temporal compression of microclimate datasets [wireless sensor networks]," *29th Annual IEEE International Conference on Local Computer Networks*, Tampa, FL, USA, 2004, pp. 516–524, doi: https://doi.org/10.1109/LCN.2004.72

[30] N. Q. V. Hung, H. Jeung and K. Aberer, "An Evaluation of Model-Based Approaches to Sensor Data Compression," in *IEEE Transactions on Knowledge and Data Engineering*, vol. 25, no. 11, pp. 2434–2447, Nov. 2013, doi: https://doi.org/10.1109/TKDE.2012.237

[31] M. A. Rahman, M. Hamada, "A prediction-based lossless image compression procedure using dimension reduction and Huffman coding," in *Multimedia Tools Applications,* vol. 82, pp. 4081–4105, 2023, doi: https://doi.org/10.1007/s11042-022-13283-3

[32] C. Chen, L. Zhang and R. L. K. Tiong, "A new lossy compression algorithm for wireless sensor networks using Bayesian predictive coding," in *Wireless Networks,* vol. 26, pp. 5981–5995, 2020, doi: https://doi.org/10.1007/s11276-020-02425-w

[33] P. Chakraborty, C. Tharini, "Integration of Prediction Based Hybrid Compression in Distributed Sensor Network," in *Wireless Personal*

Communications, vol. 122, pp. 229–241, 2022, doi: https://doi.org/10.1007/s11277-021-08896-0

[34] Z. Yan, J. Wang, L. Sheng and Z. Yang, “An effective compression algorithm for real-time transmission data using predictive coding with mixed models of LSTM and XGBoost,” in *Neurocomputing,* vol. 462, pp. 247-259, 2021, doi: https://doi.org/10.1016/j.neucom.2021.07.071

[35] M. Borova, M. Prauzek, J. Konecny and K. Gaiova, “Environmental WSN Edge Computing Concept by Wavelet Transform Data Compression in a Sensor Node,” *IFAC-PapersOnLine*, vol. 52, issue 27, pp. 246–251, 2019 ISSN 2405-8963, doi: https://doi.org/10.1016/j.ifacol.2019.12.646

[36] C. Passos, C. Pedroso, A. Batista, M. Nogueira and A. Santos, "GROWN: Local Data Compression in Real- Time To Support Energy Efficiency in WBAN," *2020 IEEE Latin-American Conference on Communications (LATINCOM)*, Santo Domingo, Dominican Republic, 2020, pp. 1–6, doi: https://doi.org/10.1109/LATINCOM50620.2020.9282319

[37] T. Bose, S. Bandyopadhyay, S. Kumar, A. Bhattacharyya and A. Pal, "Signal Characteristics on Sensor Data Compression in IoT -An Investigation," *2016 13th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*, London, UK, 2016, pp. 1–6, doi: https://doi.org/10.1109/SAHCN.2016.7733016

[38] A. Moon, J. Kim, J. Zhang and S. W. Son, “Evaluating fidelity of lossy compression on spatiotemporal data from an IoT enabled smart farm,” *Computers and electronics in agriculture*, vol. 154, pp. 304–313, 2018, doi: https://doi.org/10.1016/j.compag.2018.08.045

[39] C. Sadler and M. Martonosi, “Data compression algorithms for energy-constrained devices in delay tolerant networks,” *In Proceedings of the 4th international conference on Embedded networked sensor systems (SenSys '06),* Boulder, Colorado, USA, 2006, doi: https://doi.org/10.1145/1182807.1182834

[40] K. L. Ketshabetswe, A. M. Zungeru, B. Mtengi, C. K. Lebekwe and S. R. S. Prabaharan, "Data Compression Algorithms for Wireless Sensor Networks: A Review and Comparison," in *IEEE Access*, vol. 9, pp. 136872–136891, 2021, doi: https://doi.org/10.1109/ACCESS.2021.3116311

[41] G. Giorgi, "A Combined Approach for Real-Time Data Compression in Wireless Body Sensor Networks," in *IEEE Sensors Journal*, vol. 17, no. 18, pp. 6129–6135, 15 Sept. 15, 2017, doi: https://doi.org/10.1109/JSEN.2017.2736249

[42] A. K. M. Al-Qurabat and A. K. Idrees, “Two level data aggregation protocol for prolonging lifetime of periodic sensor networks,” in *Wireless Networks,* vol. 25, no. 6, pp. 3623–3641, 2019, doi: https://doi.org/10.1007/s11276-019-01957-0

[43] A. Makhoul, H. Harb, and D. Laiymani, “Residual energy-based adaptive data collection approach for periodic sensor networks,” *Ad Hoc Networks*,

vol. 35, pp. 149–160, 2015, ISSN 1570-8705, doi: https://doi.org/10.1016/j.adhoc.2015.08.009.

[44] A. Mahbub, F. Haque, H. Bashar and M. Rezwanul Huq, “Improved Piecewise Constant Approximation Method for Compressing Data Streams” *2019 1st International Conference on Advances in Science, Engineering and Robotics Technology (ICASERT)*, Dhaka, Bangladesh, 2019, pp. 1–6. doi: https://doi.org/10.1109/ICASERT.2019.8934460

[45] Aggarwal, C.C. Managing and Mining Sensor Data. Springer, Boston, 2013, doi: https://doi.org/10.1007/978-1-4614-6309-2

[46] I. Lazaridis and S. Mehrotra, "Capturing sensor-generated time series with quality guarantees," *Proceedings 19th International Conference on Data Engineering (Cat. No.03CH37405)*, Bangalore, India, 2003, pp. 429–440, doi: https://doi.org/10.1109/ICDE.2003.1260811

[47] C. Buragohain, N. Shrivastava and S. Suri, "Space Efficient Streaming Algorithms for the Maximum Error Histogram," *2007 IEEE 23rd International Conference on Data Engineering*, Istanbul, Turkey, 2007, pp. 1026–1035, doi: https://doi.org/10.1109/ICDE.2007.368961

[48] H. Elmeleegy, A. K. Elmagarmid, E. Cecchet, W. G. Aref and W. Zwaenepoel, “Online piece-wise linear approximation of numerical streams with precision guarantees,” *Proceedings of the VLDB Endowment*, vol. 2, no. 1, pp. 145–156, August 2009, doi: https://doi.org/10.14778/1687627.1687645

[49] J.-W. Lin, S.-w. Liao and F.-Y. Leu, “Sensor Data Compression Using Bounded Error Piecewise Linear Approximation with Resolution Reduction,” *Energies (Basel)*, vol. 12, no. 13, p. 2523, 2019, doi: https://doi.org/10.3390/en12132523

[50] S. M. S. Jalaleddine, C. G. Hutchens, R. D. Strattan and W. A. Coberly, "ECG data compression techniques-a unified approach," in *IEEE Transactions on Biomedical Engineering*, vol. 37, no. 4, pp. 329–343, April 1990, doi: https://doi.org/10.1109/10.52340

[51] D. Parker, M. Stojanovic, and C. Yu, “Exploiting temporal and spatial correlation in wireless sensor networks,” *2013 Asilomar Conference on Signals, Systems and Computers*, Pacific Grove, CA, 2013, pp. 442–446, doi: https://doi.org/10.1109/ACSSC.2013.6810315

[52] J. Azar, A. Makhoul, R. Darazi, J. Demerjian and R. Couturier, “On the performance of resource-aware compression techniques for vital signs data in wireless body sensor networks,” *2018 IEEE Middle East and North Africa Communications Conference (MENACOMM)*, Jounieh, Lebanon, 2018, pp. 1–6, doi: https://doi.org/10.1109/MENACOMM.2018.8371032

[53] O. Sarbishei, “Refined Lightweight Temporal Compression for Energy-Efficient Sensor Data Streaming,” *2019 IEEE 5th World Forum on Internet of Things (WF-IoT)*, Limerick, Ireland, 2019, pp. 550–553, doi: https://doi.org/10.1109/WF-IoT.2019.8767351

[54] B. Li, O. Sarbishei, H. Nourani, and T. Glatard, “A multi-dimensional extension of the Lightweight Temporal Compression method,” *2018 IEEE*

International Conference on Big Data (Big Data), Seattle, WA, USA, 2018, pp. 2918–2923, doi: https://doi.org/10.1109/BigData.2018.8621946

[55] L. Klus, R. Klus, E. S. Lohan, C. Granell, J. Talvitie, M. Valkama and J. Nurmi, “Direct Lightweight Temporal Compression for Wearable Sensor Data,” In *IEEE Sensors Letters*, vol. 5, no. 2, pp. 1–4, Feb. 2021, doi: https://doi.org/10.1109/LSENS.2021.3051809

[56] S. Lu, Q. Xia, X. Tang, X. Zhang, Y. Lu and J. She, "A Reliable Data Compression Scheme in Sensor-Cloud Systems Based on Edge Computing," in *IEEE Access*, vol. 9, pp. 49007–49015, 2021, doi: https://doi.org/10.1109/ACCESS.2021.3068753

[57] W. Shi, J. Cao, Q. Zhang, Y. Li and L. Xu, “Edge computing: vision and challenges,” in *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016, doi: https://doi.org/10.1109/JIOT.2016.2579198

[58] M. Satyanarayanan, “The emergence of edge computing,” in *Computer*, vol. 50, no. 1, pp. 30–39, 2017, doi: https://doi.org/10.1109/MC.2017.9

[59] E. Yigitoglu, M. Mohamed, L. Liu and H. Ludwig, “Foggy: a framework for continuous automated IoT application deployment in fog computing,” in *2017 IEEE International Conference on AI and Mobile Services (AIMS)*, Honolulu, HI, USA, 2017, pp. 38–45, doi: https://doi.org/10.1109/AIMS.2017.14

[60] M. R. K. Ariffin, M. A. Asbullah, N. A. Abu and Z. Mahad, “A new efficient asymmetric cryptosystem based on the integer factorization problem of N = P^2.q,” in *Malaysian Journal of Mathematical Sciences*, vol. 7(S), pp. 19–37, 2012, Special Issue 3rd International Conference on Cryptography and Computer Security 2012.

[61] S. F. S. Adnan, M. A. M. Isa and H. Hashim, “Energy analysis of the AAb lightweight asymmetric encryption scheme on an embedded device,” in *2016 IEEE Industrial Electronics and Applications Conference (IEACon)*, Kota Kinabalu, Malaysia, pp. 116–122, 2016, doi: https://doi.org/10.1109/IEACON.2016.8067366

[62] B. Ying, "An energy-efficient compression algorithm for spatial data in wireless sensor networks," *2016 18th International Conference on Advanced Communication Technology (ICACT)*, PyeongChang, Korea (South), 2016, pp. 161–164, doi: https://doi.org/10.1109/ICACT.2016.7423312

[63] S. A. Fallah, M. Arioua, A. El Oualkadi and J. El Asri, "On the performance of piecewise linear approximation techniques in WSNs," *2018 International Conference on Advanced Communication Technologies and Networking (CommNet)*, 2018, pp. 1–6, doi: https://doi.org/10.1109/COMMNET.2018.8360262

ORIGINAL PAPERS

I

REQUIREMENTS FOR ENERGY EFFICIENT EDGE COMPUTING: A SURVEY

by

Olli Väänänen & Timo Hämäläinen, 2018

Proceedings of the 18th International Conference on Next Generation Wired/Wireless Advanced Networks and Systems NEW2AN 2018

https://doi.org/10.1007/978-3-030-01168-0_1

Requirements for Energy Efficient Edge Computing: A Survey

Olli Väänänen[1](✉) and Timo Hämäläinen[2]

[1] Industrial Engineering, School of Technology, JAMK University of Applied Sciences, Jyväskylä, Finland
olli.vaananen@jamk.fi

[2] Department of Mathematical Information Technology, University of Jyväskylä, Jyväskylä, Finland
timo.t.hamalainen@jyu.fi

Abstract. Internet of Things is evolving heavily in these times. One of the major obstacle is energy consumption in the IoT devices (sensor nodes and wireless gateways). The IoT devices are often battery powered wireless devices and thus reducing the energy consumption in these devices is essential to lengthen the lifetime of the device without battery change. It is possible to lengthen battery lifetime by efficient but lightweight sensor data analysis in close proximity of the sensor. Performing part of the sensor data analysis in the end device can reduce the amount of data needed to transmit wirelessly. Transmitting data wirelessly is very energy consuming task. At the same time, the privacy and security should not be compromised. It requires effective but computationally lightweight encryption schemes. This survey goes thru many aspects to consider in edge and fog devices to minimize energy consumption and thus lengthen the device and the network lifetime.

Keywords: IoT, Edge Computing, Fog Computing, sensor data compression.

1 Introduction

The Internet of Things (IoT) has been in focus on recent years. There are already billions of devices connected to the Internet and the amount of the Internet connected things is estimated to grow exponentially in these years [1, 2]. There are forecasts that by 2020 there will be more than 50 billion devices connected to the Internet [3]. These connected devises and things are very heterogeneous and require very different and application specific solutions and approaches. [1] The IoT as a concept was first introduced in 1999 by Kevin Ashton and it was related to the devices connected to the Internet via RFID connection. [1] The term IoT was mainly forgotten for years after that but it was reinvented some years ago. The exact definition of the IoT is still not described clearly, [1] but the technologies, solutions and the use of the IoT is all the time emerging.

There are already solutions of the IoT in use but the real success of the IoT depends on the standardization, which allows the compatibility, interoperability, reliability and effectiveness of the IoT solutions. The IoT devices and things should be able to autonomously communicate with other devices or things and connect data to the Cloud. The IoT describes the next generation of the Internet, where physical things are connected to the Internet and can be identified and accessed via Internet. [1]

There are presented and used many solutions and techniques to save energy in the IoT devices. These methods are mainly based on reducing wireless broadcasting because it is more energy consuming to broadcast data than pre-analyze it in close proximity of the source (sensor). [4] The IoT sensor data need to be compressed efficiently to reduce and minimize the cost of broadcast and storage [5]. At the same time, many IoT devices are battery powered wireless devices. Thus, these IoT devices can be located in places where changing the battery might be impossible or at least battery replacement cost is one of the most critical source of cost in this kind of devices. [2] These devices are often very limited in computing power. So often, it is the case that it is possible to perform only very light analysis of the collected data in locally. In addition, the IoT itself is very constrained in terms of bandwidth, energy and storage. [5, 6]

The IoT systems and the whole IoT sector is very heterogeneous. The things vary a lot and may move geographically and they need to interact with other things and Cloud systems in real-time mode. When designing the IoT systems it should be taken account scalability and interoperability of the heterogeneous devices. Design of the IoT applications and systems require involvement of many factors like networking, communication, business models and processes, and security. The IoT architecture should be very adaptive to make IoT devices to interact with other devices and with the Internet. [1]

2 Definition of Edge and Fog

The term Fog Computing was introduced by Flavio Bonomi in 2012. [7, 8] It refers to dispersed Cloud computing which is vital in several applications where the IoT devices collect data in the local network and the actions required from analyzed data take place in the same local network. [9] In that kind of case, it is not efficient to send all the data to centralized Cloud to be analyzed. It is not even possible to send data to the Cloud for analysis in many latency critical applications. The term Edge Computing means that computing happens in close proximity of data sources in the edge of the network. In many cases the terms Edge Computing and Fog Computing are interchangeable. But it can be defined that Edge refers more to the device side very close to data sources and Fog refers more infrastructure side like gateways and routers. [10]

Cloud service providers locate their data centers often in rural areas to minimize costs. This lead to high latencies because customers are often located far from data centers. [11] Many IoT applications require very short response times, some create a large amount of data that can be heavy for network and some applications are involved with sensitive private data. Cloud computing cannot reply all these requirements so the Edge Computing is one answer for these challenges. [10] Latency critical applications are for example many intelligent transportation and traffic systems, autonomous vehicles, virtual reality (VR) and augmented reality (AR) applications. [7] Also many safety critical applications cannot rely on the connection to the Cloud. For example, vehicle-to-vehicle connection or data from vehicles can be used to avoid collision, but that analysis need to be done locally or in very close proximity located Cloudlet [7]. The Cloudlet means smaller size local datacenter. Safety critical systems are also very common in industrial automations systems. These kind of applications cannot tolerate possible Cloud outages and they often need low and predictable latency [7, 11]. This kind of new Fog Computing paradigm is not a replacement of the centralized Cloud. These concepts are more complementary to each other. [9, 11] In some applications the Cloud is not even possible to be used; this kind of situation happens for example in the modern aircraft. The modern aircraft can generate nearly half a terabyte of data from its sensors in one flight. [7] This amount of data cannot be sent to the Cloud for real time analysis from the middle of the ocean. Only possibility is to analyze the data locally and then perhaps download the raw data after flight for further analysis that can be executed in the Cloud. Even in ground level, the current wireless networks will be challenged with the amount of data that the huge amount of devices will produce in the near future [10]. Most of the data produced by the IoT devices will be analyzed locally in the Edge devices and will never be transmitted to the Cloud. [10]

In Fig. 1 is illustrated the basic architecture of the IoT infrastructure including Edge and Fog devices. The difference between the Edge and the Fog devices is not always as clear as presented in Fig 1.

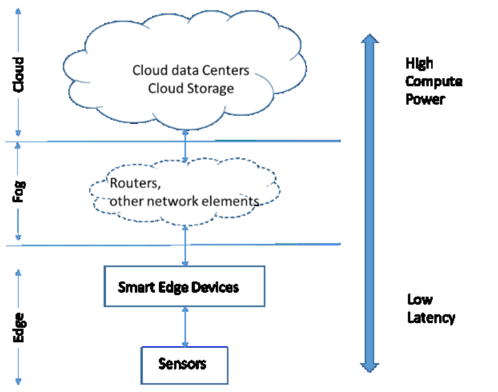


Fig. 1. Edge and Fog architecture in IoT. [12]

Fog and Edge devices can be efficient data servers, routers, gateways, any kind of embedded systems or even end node like vehicles or sensors with some computational capability. [11] The Edge devices can be small-embedded devices with very energy efficient and limited micro controller or more capable single board Linux-computer like Raspberry PI. In Fig 1. typically sensors are small wireless sensor tags and Smart Edge Devices are gateways for sensors. Smart Edge Device (gateway) is connected to the Internet via wireless or wired connection. Edge and Fog devices are very heterogeneous in nature with different hardware architectures and they run various different Operating Systems (OS). There are also available numerous different wireless access technologies and sensor network topologies [11]. This heterogeneous nature of Edge and Fog devices and systems avoid developing generic and easily adaptable solutions for Edge and Fog analytics. It is predicted that the Edge Computing could have as big impact in society as Cloud Computing has [10].

3 Benefits of the Edge Computing

While the Cloud Computing is very efficient method for data processing having a huge amount of computing power, [10] the Cloud Computing cannot meet and ensure the Quality of Service (QoS) in the IoT due to unstable latency and possible outages in the network connection and the Cloud servers. Fog or Edge Computing is an answer for the problem. In the Edge Computing the majority of the computing is carried out in close proximity of the data source. There are researches done that proof the Edge Computing reduction in response times and in energy consumption. By doing part of computation and analysis in the Edge reduce the needed wireless connection bandwidth. For example, photos can be compressed in the Edge before transmitting to the cloud. [10] Even if most of the data analysis is done in the Cloud, it is recommended to do some preprocessing for sensor data in the Edge before uploading it to the Cloud. In minimum this kind of preprocessing can be only filtering erroneous sensor data. More advanced preprocessing can mean different compression methods like sending only the information of the variation/alteration of the sensor values and not absolute values. This kind of preprocessing can reduce significantly the amount of data needed for upload data in the Cloud [10].

Security and privacy critical application can also benefit from the Edge/Fog Computing approach where the original raw and sensitive data is not sent to the centralized Cloud thru public Internet. [7] Data sent to the Cloud can be denatured data; for example, in images the faces can be blurred. [7] Applications producing very sensitive and private data are for example different healthcare applications.

Also home automation systems sending information to the Cloud could include some private sensitive data. For example, information of the water and electricity usage could easily tell if the house is vacant or not. If the computation is kept in close proximity of this data (in the Edge), it could be decent solution to keep sensitive data in private. [10] But if this home automation application is connected to the Internet, this sensitive data could be reachable for inappropriate quarters. So the cybersecurity is vital for all IoT applications whether the sensitive data is transferred to the Cloud or not.

4 Edge and Fog Computing Challenges

Fog and Edge devices are very heterogeneous. [11] It is difficult to design easily adaptable and generic solutions for the Edge Computing. Most applications are individual and cannot utilize generic computational, data aggregation and data analysis methods. There are different hardware platforms and different operational systems. Hardware platforms can vary from very simple micro-controller based platform with very limited memory to single board Linux-computer like Raspberry PI that is rather powerful platform. Virtualization is one way to handle multiplatform and multi-OS challenge.

One possibility towards generic solutions to be used in different and computationally restricted platforms is a container-based approach. Container-based virtualization can be considered as a lightweight virtualization solution. Because of lightweight nature, the containers can run in computationally limited IoT-platform like Raspberry PI. [13, 14] Containers could be used in the different platforms to perform same tasks. Anyway, these platforms could not be very limited basic embedded micro-controller based platforms, but require more computational power and generic operating system (OS) like Linux.

In [15], has been tested the ARM-based Single Board Computers with Docker containers and compared the overall efficiency in power consumption to the native executions. The performance evaluation showed almost negligible impact with container virtualization compared to native executions.

4.1 Methods for Reducing Energy Consumption in Wireless Sensor Networks

Several energy-efficient routing algorithms have been proposed for wireless sensor networks (WSN) but they are mostly not suitable for the IoT. Current IoT devices are mostly static and follow tree-based structure. [16] Dynamic routings developed for WSN architectures are not suitable for the IoT. The IoT network is often a complex large scale network and dynamic routing is difficult to be used effectively in this kind of network. [17]

The Low-Energy Adaptive Clustering Hierarchy (LEACH) protocol utilizes several methods and techniques to reduce energy consumption in WSN. [18] LEACH is the most popular routing algorithms used in WSNs [19]. There are several variations and further developments of LEACH protocol like LEACH-C and ENHANCED LEACH for example [16, 20]. Weight energy efficient clustering (WEEC) is an extended version of LEACH. In WEEC the energy efficiency optimization is done by cluster head (CH) selection procedure. Every node in the sensor network can be elected as a cluster head. WEEC is a single-hop routing protocol. [19]

In [16], the authors have presented a cluster head selection for energy optimization (CHSEO) algorithm to reduce the overall energy consumption in the IoT network. The CHSEO algorithm is based on selecting the optimal cluster head of the sensor nodes to reduce overall energy consumption. Hierarchical IoT sensor node framework is composed of different node types. Sensor node is sensing, aggregating and forwarding data, Relay node is receiving the data from sensor nodes and transmit it to the cluster head. Cluster head collects, aggregates and transmit the data to the base station. Base station collects, aggregates, analyses and process the data. The CHSEO algorithm was proved to have better performance than traditional WSN mechanism in energy consumption and network lifetime.

Other example of hierarchical network architecture to reduce IoT network energy consumption is presented in [17]. It is based on hierarchical relay node placement with energy efficient routing mechanism. Ad Hoc On-Demand Distance Vector (AODV) routing protocol has been used. This proposed network architecture gives balanced energy consumption and thus better network lifetime. [17]

Modern long-range low-power IoT networks (NB-IoT, LoRa, SigFox) have star topology, so intelligent routing algorithms are out of the question. [21] In these technologies, the ultra-low energy consumption has been achieved by using very limited bandwidth and/or intelligent modulation.

4.2 Data Compression Methods in Edge Device: Lossy and Lossless Methods

In the IoT, huge amount of sensors are generating data and that data should be stored and processed with minimal loss of information. Sensor data compression is not a new discipline and several different compression algorithms are presented. [5] There are also very energy efficient contemporary compression methods for resource constrained IoT-nodes presented [6]. Data aggregation is also related to the data compression. Data aggregation here means for example to combine multiple sensor data and filter the redundant data. Data aggregation in wireless sensor network reduce the amount of data needed to transmit to the base station and thus reduce energy consumption. [18] Most of the compression methods presented for the IoT sensor data compression are lossy compression methods. Lossy methods are more efficient in compression compared to lossless methods. Lossy methods try to identify meaningful data points and discard redundant data. Different compression algorithms perform differently with different types of data sets. Also their computational complexity differs. [5]

Lossy compression methods can be divided in two groups: Time domain and Transform domain. Time domain compression algorithms compress time series data directly without any transformation. Transform domain compression methods transform data into a different domain. Well-known transform domain methods are for example Discrete Fourier Transform (DFT) and Fast Fourier Transform (FFT). [5] Different lossy compression algorithms are listed in Table 1.

Table 1. Lossy Compression Algorithms. [5, 6]

Name of the Algorithm	**Type**
Box-Car	Time Domain
Backward Slope	Time Domain
OSIsoft PI software	Time Domain
Compression extracting major extrema	Time Domain
PLA, PCA	Time Domain
Critical Aperture (CA)	Time Domain
Fractal Resampling (FR)	Time Domain
Lightweight Temporal Compression (LTC)	Time Domain
Fast Fourier Transform (FFT)	Transform Domain
Discrete Cosine Transform (DCT)	Transform Domain
Chebyshev Transform (CH)	Transform Domain

Wavelet Transform (CWT, DWT, WPT)	Transform Domain

In ref. [5] the authors have selected four different lossy compression methods and compared their applicability to different signal characteristics. Compared methods were Critical Aperture (CA), Fractal Resampling (FR), Chebyshev Transform (CH) and Wavelet Packet Decomposition (WPD). Data used for comparison has been diverse publicly available sensor datasets. Comparison has been made by comparing the compression ratio with same Percentage Root mean square deviation (PRD). PRD level used in comparison has been 5 %. Used datasets were different in composition. Some were quasi-periodic (QP), some non-stationary (NS) with sudden transient spikes and some non-stationary (NS) with periodic seasonal components. [5]

As a result, the CH was the most effective method for QP data in terms of compression ratio. For NS with transient spikes data, the CA, FR and WPD were remarkably more effective than CH method. For NS with periodic seasonal data the WPD is the most effective method. [5]

In [5], it is also shown that WPD requires considerably more computational time compared to the other methods. This means a higher energy consumption. In ref. [6] has been introduced lightweight compression algorithm for spatial data which is more energy efficient than wavelet compression. This lightweight compression algorithm can reduce energy consumption to half of the original consumption. This lightweight and energy-efficient compression algorithm is based on a lightweight temporal compression method named LTC [22]. LTC is tunable in accuracy and suitable for the datasets that are largely continuous and slowly changing. LTC is widely used method due to its good compression performance and low computational complexity. [6] LTC also requires very little storage compared to many other compression techniques. LTC is very effective for many environmental type data (temperature, humidity) which are approximately linear in small enough time window. Thus, LTC leverages temporal linearity of environmental data to compress that data. [22]

5 Wireless Technologies for Energy Efficient IoT

For years the main wireless technology for transmitting sensor data with low energy consumption was IEEE 802.15.4 (mostly used protocol is called ZigBee). ZigBee was designed for ultra-low energy consumption and it has been popular in WSNs. [21] IEEE 802.11 (WiFi) has also been available for years but traditionally it has been used for high data rates and it has had rather high energy consumption. To address this energy consumption problem, there is available Power Saving Mode (PSM) in IEEE 802.11. [18] This Power Saving Mode is developed for battery powered mobile devices. IEEE 802.11 was not designed for sensor applications but with PSM it has proofed to be potential alternative for other technologies used for WSNs. In some cases, the IEEE 802.11 PSM can outperform the IEEE 802.15.4 in energy consumption. [23] Bluetooth Low Energy (BLE) is very popular and widely used due to its availability. It is already available in most modern smartphones and it is widely used in wearable devices like heart rate monitors and other monitoring applications.

ZigBee, BLE and WiFi uses the 2.4 GHz ISM frequency band while ZigBee is available also in sub-1 GHz band (868 and 915 MHz). IEEE 802.11ah version address for requirements of the IoT, like increased range, increased reliability and low energy consumption. IEEE 802.11ah is operated in sub-1 GHz range. [21]

Using sub-1 GHz band increases the range and penetration thru obstacles (buildings, constructions). Sub-1 GHz band is also less crowded compared to popular 2.4 GHz band and thus these technologies are less vulnerable for interference. [24]

ZigBee, BLE and WiFi all have rather short range, even if sub-1 GHz band is used (ZigBee and WiFi). As an answer for this limitation there are recent developments in long-range technologies like SigFox and LoRa. These are so called low-power wide-area-networks (LPWAN) [25]. SigFox is an ulta-narrow-band technology and it uses sub-1 GHz band (868 MHz in Europe). Its range is announced to be even up to 40 km. Direct competitor for SigFox is the LoRa. It uses the same frequency band as SigFox but its modulation is based on Chirp Spread Spectrum (CSS). [21] CSS modulation was developed in the 1940's and it is very robust for interference and multipath fading. In CSS modulation the information in spread to different frequency channels and it has noise like properties. [26]

Novel cellular based wireless technology for IoT solutions is Narrow Band-IoT (NB-IoT) which uses narrow bandwidth for lower power consumption. [27] The Third Generation Partnership Project (3GPP) introduced the NB-IoT in LTE Release 13. NB-IoT bandwidth for both uplink and downlink is set to 180 kHz. It is exactly size of one physical resource block (PRB) in LTE standard. [28]

In Table 2 has been combined the main characteristics of the main WSN technologies used in the IoT. LPWAN technologies have long range and very limited data rate. ZigBee, BLE and WiFi have much higher data rate but the range is very limited.

Table 2. Wireless technologies summary for IoT. [1, 23, 24, 26]

Technology	Band	Topology	Announced range	Data rate
802.15.4	2.4 GHz / 0.9 GHz	Meshed	50 m	0.25 Mb/s
BLE	2.4 GHz	Scatternet	10 m	0.125 – 2 Mb/s
802.11 PSM	2.4 GHz	Star	100 m	11 Mb/s
802.11ah	0.9 GHz	Star	100m – 1 km	0.15 – 78 Mb/s
SigFox	0.9 GHz	Star	Up to 40-50 km	100 b/s or 1000 b/s
LoRa	0.9 GHz	Star	Up to 15 km (suburban), 45 km (rural)	0.25 – 50 kb/s
NB-IoT	700-900 MHz	Star	Up to 35 km	20-65 kb/s

As both SigFox and LoRa uses unlicenced ISM band, there is no guarantee for latency. For latency critical applications, the NB-IoT is better choice while SigFox and LoRa are suitable for low-cost projects with wide area coverage [26]. NB-IoT latency is maximum 10 seconds according to the standard, while SigFox and Lora can have latency of 10s of seconds. [27, 28] Lora and SigFox are both very energy efficient technologies with very large range. BLE is also very energy efficient in its range. [21]

6 Energy Efficient IoT Protocols

The most common IoT application protocols are MQTT, CoAP, XMPP and AMQP. MQTT (message queue telemetry transport) and CoAP (constrained application protocol) are designed especially for resource constrained devices like IoT end nodes and gateways. [29, 30]

MQTT protocol is a publish-subscribe messaging protocol with minimal bandwidth requirements. It uses TCP (transmission control protocol) for transport. It is designed to be used in devices with restricted computational power and limited memory. MQTT is considered as a perfect messaging protocol for M2M and IoT applications because of its ability to function within low power, low memory and cheap devices with low bandwidth networks. [29]

CoAP protocol is a request-response protocol but it can function as a publish-subscribe mode too. CoAP uses UDP (user datagram protocol) for transport but it can be used for TCP too. CoAP has a wide acceptance for constrained devices. [30]

In ref. [30] the authors have made comparison and experimental analysis between MQTT and CoAP. As a result they have found that MQTT consumes more bandwidth for transferring same payload than CoAP. But both protocols are efficient in terms of energy consumption.

In ref. [31] have been evaluated the performance, energy efficiency and resource usage of several IoT protocols (MQTT, CoAP, MQTT-SN, WebSocket and TCP). As a result, the authors found that MQTT and CoAP protocols are largely affected by the packet size. In generally CoAP is the most efficient in terms of energy consumption and bandwidth usage. But MQTT protocol is more reliable.

XMPP (extensible messaging and presence protocol) and AMQP (advanced message queuing protocol) are other popular protocols but they require more resources and they are not so suitable for resource constrained devices.

7 Security and Privacy Issues in the Edge

Privacy and security is a very big issue and concern in the IoT systems and applications. In the IoT systems, the end nodes (IoT devices) are connected to the Internet and thus these devices are reachable from all over the Internet. This kind of devices can be for example IP-cameras, health monitors and wearable devices or even WiFi connected toys. These devices can be connected by others if not protected properly.

Ownership of the collected data is other issue to take account. If the data is left on edge device for storage and analysis, then there are no ownership problems as the owner of the device can have all the rights for that data. [10]

Battery powered IoT devices have very limited computational power, so complex encryption techniques require significant amount of computing and thus increase energy consumption. Lightweight encryption algorithms for the IoT devices have been developed.

Encryption scheme can be symmetric or asymmetric and both can be used in the IoT devices. In symmetric encryption scheme only one key is used to encrypt and decrypt the data. Both sender and receiver need to know the same key. In asymmetric encryption scheme two distinct keys are used. One for encrypting and other for decrypting. The advantage here is that the encrypting key can be public key and available to anyone. For asymmetric scheme the key need to be longer than in symmetric scheme to be secure. Thus calculations needed are longer than in symmetric scheme. Famous asymmetric encryption schemes are Rivest, Shamir, Adleman (RSA) scheme and Elliptic Curve Cryptography (ECC). [32]

Several researches have been done to compare ECC and RSA schemes to each other in regarding to encryption/decryption time and key length. The ECC has proved to be more efficient with shorten encryption/decryption time, smaller storage and in generally more energy efficient than RSA. [33]

In ref. [34] the authors have presented lightweight asymmetric encryption scheme called AAβ and in ref. [35] the authors have made comparison in energy consumption between AAβ and RSA. The AAβ outperforms the RSA significantly in encryption and decryption.

8 Conclusions

In this study, a comprehensive study of the energy efficient Edge Computing has been carried out. There are a lot of research published from the different phases and aspects to reduce energy consumption in wireless end devices, but only few of them encompass the subject broadly. Minimizing energy consumption is one of the key aspects to carry out in the IoT device and system development. IoT end devices are often battery powered devices with wireless connection. Thus the computational resources are constrained but at the same time these devices should be able to do pre-processing and analysis for sensor data to reduce transferred data via wireless connection.

Most methods for reducing energy consumption in the IoT devices are concentrated to reduce wireless data transfer. Wireless data transfer is often the most energy consuming operation in the IoT device. In addition, many latency critical applications are pushing the development towards Edge Computing.

At the same time when more and more data analysis is carried out in close proximity of the sensors (in Edge and Fog); there are available several novel wireless technologies to transfer sensor data with low energy consumption. Considering energy consumption in every phase from the sensor to the Internet, it is possible to reduce energy consumption significantly. Many of these techniques are studied in this survey.

References

1. Li, S., Xu, L.D. & Zhao, S.: The internet of things: a survey. In: Springer Information Systems Frontiers, Volume 17, Issue 2, pp 243-259, April 2015.
2. Montori, F., Contigiani, R., Bedogni, L.: Is WiFi suitable for energy efficient IoT deployments? A performance study. In: 2017 IEEE 3rd International Forum on Research and Technologies for Society and Industry (RTSI), Modena, 2017, pp. 1-5.

3. Jayakumar, H., Raha, A., Kim, Y., Sutar, S., Lee, W.S., Raghunathan, V.: Energy-efficient system design for IoT devices. In: 2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC), Macau, 2016, pp. 298-301.
4. Stojkoska, B.R., Nikolovski, Z.: Data compression for energy efficient IoT solutions. In: 2017 25th Telecommunication Forum (TELFOR), Belgrade, 2017, pp. 1-4.
5. Bose, T., Bandyopadhyay, S., Kumar, S., Bhattacharyya, A., Pal, A.: Signal Characteristics on Sensor Data Compression in IoT - An Investigation. In: 2016 IEEE International Conference on Sensing, Communication and Networking (SECON Workshops), London, 2016, pp. 1-6.
6. Ying, B.: An Energy-Efficient Compression Algorithm for Spatial Data in Wireless Sensor Networks. ICACT 2016.
7. Satyanarayanan, M.: The Emergence of Edge Computing. Computer, vol. 50, no. 1, pp. 30-39, Jan. 2017.
8. Bonomi, F., Milito, R., Zhu, J., Addepalli, S.: Fog computing and its role in the internet of things. In: Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing-MCC '12, Helsinki, Finland, 17 August 2012; pp. 13–15.
9. Jalali, F., Khodadustan, S., Gray, C., Hinton, K., Suits, F.: Greening IoT with Fog: A Survey. In: 2017 IEEE International Conference on Edge Computing (EDGE), Honolulu, HI, 2017, pp. 25-31.
10. Shi, W., Cao, J., Zhang, Q., Li, Y., Xu, L.: Edge Computing: Vision and Challenges. IEEE Internet of Things Journal*,* vol. 3, no. 5, pp. 637-646, Oct. 2016.
11. Venkat Narayana Rao, T., Amer Khan, M.D., Maschendra, M., Kiran Kumar, M.: A Paradigm Shift from Cloud to Fog Computing. IJCSET (www.ijcset.net) Vol 5, Issue 11, pp 385-389. November 2015.
12. Yigitoglu, E., Mohamed, M., Liu, L., Ludwig, H.: Foggy: A Framework for Continuous Automated IoT Application Deployment in Fog Computing. In: 2017 IEEE International Conference on AI & Mobile Services (AIMS), Honolulu, HI, 2017, pp. 38-45.
13. Morabito, R.: A performance evaluation of container technologies on Internet of Things devices. In: 2016 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), San Francisco, CA, 2016, pp. 999-1000.
14. Pahl, C., Helmer, S., Miori, L., Sanin, J., Lee, B.: A Container-Based Edge Cloud PaaS Architecture Based on Raspberry Pi Clusters. In: 2016 IEEE 4th International Conference on Future Internet of Things and Cloud Workshops (FiCloudW)*,* Vienna, 2016, pp. 117-124.
15. Morabito, R.: Virtualization on Internet of Things Edge Devices With Container Technologies: A Performance Evaluation. IEEE Access*,* vol. 5, pp. 8835-8850, 2017.
16. Krishna, P.V., Obaidat, M.S., Nagaraju, D., Saritha, V.: CHSEO: An Energy Optimization Approach for Communication in the Internet of Things. In: GLOBECOM 2017 - 2017 IEEE Global Communications Conference, Singapore, 2017, pp. 1-6.
17. Cho, Y., Kim, M., Woo, S.: Energy efficient IoT based on wireless sensor networks. In: 2018 20th International Conference on Advanced Communication Technology (ICACT), Chuncheon-si Gangwon-do, Korea (South), 2018, pp. 294-299.
18. Dargie, W., Poellabauer, C.: Fundamentals of Wireless Sensor Networks, Theory and Practise. Wiley (2010).
19. Bhushan, B., Sahoo, G.: A comprehensive survey of secure and energy efficient routing protocols and data collection approaches in wireless sensor networks. In: 2017 International Conference on Signal Processing and Communication (ICSPC), Coimbatore, 2017, pp. 294-299.

20. Kumar, S., Verma, U.K., Sinha, D.: Performance analysis of LEACH and ENHANCED LEACH in WSN. In: 2016 International Conference on Circuit, Power and Computing Technologies (ICCPCT), Nagercoil, 2016, pp. 1-7.
21. Morin, É., Maman, M., Guizzetti R., Duda, A.: Comparison of the Device Lifetime in Wireless Networks for the Internet of Things. IEEE Access, vol. 5, pp. 7097-7114, 2017.
22. Schoellhammer, T., Osterwein, E., Greenstein, B., et al.: Lightweight temporal compression of microclimate datasets. In: Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks IEEE Computer Society, 2004, pp. 516-524.
23. Tozlu, S.: Feasibility of Wi-Fi enabled sensors for Internet of Things. In: 2011 7th International Wireless Communications and Mobile Computing Conference, Istanbul, 2011, pp. 291-296.
24. de Carvalho Silva, J., Rodrigues, J.J.P.C., Alberti, A.M., Solic, P., Aquino, A.L.L.: LoRaWAN — A low power WAN protocol for Internet of Things: A review and opportunities. In: 2017 2nd International Multidisciplinary Conference on Computer and Energy Science (SpliTech), Split, 2017, pp. 1-6.
25. Ayele, E.D., Hakkenberg, C., Meijers, J.P., Zhang, K., Meratnia, N., Havinga, P.J.M.: Performance analysis of LoRa radio for an indoor IoT applications. In: 2017 International Conference on Internet of Things for the Global Community (IoTGC), Funchal, 2017, pp. 1-8.
26. Poursafar, N., Alahi, M.E.E., Mukhopadhyay, S.: Long-range wireless technologies for IoT applications: A review. In: 2017 Eleventh International Conference on Sensing Technology (ICST), Sydney, NSW, 2017, pp. 1-6.
27. Wang, H., Fapojuwo, A.O.: A Survey of Enabling Technologies of Low Power and Long Range Machine-to-Machine Communications. IEEE Communications Surveys & Tutorials, vol. 19, no. 4, pp. 2621-2639, Fourthquarter 2017.
28. Xu, J., Yao, J., Wang, L., Ming, Z., Wu, K., Chen, L.: Narrowband Internet of Things: Evolutions, Technologies and Open Issues. IEEE Internet of Things Journal, 2017.
29. Yassein, M.B., Shatnawi, M.Q., Aljwarneh, S., Al-Hatmi, R.: Internet of Things: Survey and open issues of MQTT protocol. In: 2017 International Conference on Engineering & MIS (ICEMIS), Monastir, 2017, pp. 1-6.
30. Bandyopadhyay, S., Bhattacharyya, A.: Lightweight Internet protocols for web enablement of sensors using constrained gateway devices. In: 2013 International Conference on Computing, Networking and Communications (ICNC), San Diego, CA, 2013, pp. 334-340.
31. Mun, D.H., Dinh, M.L., Kwon, Y.W.: An Assessment of Internet of Things Protocols for Resource-Constrained Applications. In: 2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC), Atlanta, GA, 2016, pp. 555-560.
32. Adnan, S.F.S., Isa, M.A.M., Hashim, H.: Analysis of asymmetric encryption scheme, AAβ Performance on Arm Microcontroller. In: 2017 IEEE Symposium on Computer Applications & Industrial Electronics (ISCAIE), Langkawi, 2017, pp. 146-151.
33. Diro, A.A., Chilamkurti, N., Nam, Y.: Analysis of Lightweight Encryption Scheme for Fog-to-Things Communication. IEEE Access.
34. Ariffin, M.R.K., Asbullah, M.A., Abu, N.A., Mahad, Z.: A New Efficient Asymmetric Cryptosystem Based on the Integer Factorization Problem of N=P^2 .q. In: Malaysian J. Math. Sci. 7(S) 19-37 Spec. Issue 3rd Int. Conf. Cryptol. Comput. Secur. 2012, vol. 7, pp. 1– 6, 2012.
35. Adnan, S.F.S., Isa, M.A.M., Hashim, H.: Energy analysis of the AAβ lightweight asymmetric encryption scheme on an embedded device. In: 2016 IEEE Industrial Electronics and Applications Conference (IEACon), Kota Kinabalu, 2016, pp. 116-122.

II

PREDICTIVE PUMPING BASED ON SENSOR DATA AND WEATHER FORECAST

by

Olli Väänänen, Jari Hautamäki & Timo Hämäläinen, 2019

Proceedings of the 2019 IEEE Sensors Applications Symposium (SAS), pp. 1–5.

https://doi.org/10.1109/SAS.2019.8706018

Predictive pumping based on sensor data and weather forecast

Olli Väänänen
School of Technology
JAMK University of Applied Sciences
Jyväskylä, Finland
olli.vaananen@jamk.fi

Jari Hautamäki
School of Technology
JAMK University of Applied Sciences
Jyväskylä, Finland
jari.hautamaki@jamk.fi

Timo Hämäläinen
Department of Mathematical
Information Technology
University of Jyväskylä
Jyväskylä, Finland
timo.t.hamalainen@jyu.fi

Abstract—**In energy production, peat extraction has a significant role in Finland. However, protection of nature has become more and more important globally. How do we solve this conflict of interests respecting both views? In peat production, one important phase is to drain peat bog so that peat production becomes available. This means that we have control over how we can lead water away from peat bog to nature without water contamination with solid and other harmful substances. In this paper we describe a novel method how fouling of water bodies from peat bog can be controlled more efficiently by using weather forecast to predict rainfall and thus, minimize the effluents to nature.**

Keywords—Internet of Things, open data, predictive control, rain prediction

I. INTRODUCTION

Today, nature protection has become a more and more important issue. All technology solutions to help avoid nature resource overconsumption are welcome.

In Finnish peatlands the peat production area is approximately 68,000 hectares. In 2016, there were 45,000 hectares of energy peat production and 5,000 hectares of peatland in the production of environmental peat. During the 2000s, the production of energy peat has averaged about 400 megawatt hours per hectare. The total output of energy peat has varied from 8 terawatt hours to 35 terawatt hours in the 2000s [1].

Peat production is seasonal. The peat season is in normal years from mid-May to early September. The average summer lasts 40-50 days, when production is possible. The production is also weather dependent, and the yield per hectare varies both at the annual and regional levels [1].

Views on the harmfulness of peat production on watercourses are based on obsolete data and beliefs about peat production. These beliefs are bolstered by the fact that peat production has been accused of water contamination [1].

The central part in avoiding fouling of water bodies is predictable control of runoff water. An important role in this process is the ability to filter in significant quantities solid and other harmful substances before water accesses water bodies. The present method in Finland is based on pumping of the water from drainage reservoirs (pump pool) to filtering field by measuring the water height in reservoirs.

In this research paper, we introduce a novel method to powerfully filter solid and other substances from water. Our solution is based on using IoT technology with weather forecast and rainfall measurement locally in peat bog. It stabilizes the flow of water from the drainage reservoirs significantly and thus notably reduces the load on the peat production area.

II. COMBINING SENSOR DATA AND WEATHER FORECASTS IN IOT SYSTEM

There are some experiments of using weather forecasts for prediction in the IoT systems, for example in different agricultural systems and applications. Reference [2] proposes a smart water dripping system for the farmers to irrigate the farms efficiently. It is mainly based on local sensors like soil temperature, moisture and pH. The systems also collect the weather forecast information from websites and use its information to decide if the watering is needed. Due to possible forecast inaccuracy, the system gives for the operator possibility to manually override the system suggestion [2].

A farming automation system for plants watering which uses the weather prediction based on fuzzy logic algorithm has been introduced in [3]. The system is based on sensor data and forecast from the weather service provider. The system uses fuzzy logic algorithm to calculate if the plant should be watered. Sensor data includes the soil moisture data and rain sensor data. Weather prediction data is collected from two different weather service providers (WSP Open Weather and Weather Underground) [3].

Rainfall forecasts have been used in cyber physical systems for predicting and preventing flood hazards. Yang et al. [4] have used an ensemble numerical prediction system to get more reliable rainfall forecast. The ensemble numerical system used is based on 20 ensemble units. These ensemble units are various numerical weather prediction models with different configurations. The system is based on worldwide observation data of weather parameters. This data is obtained from various sources like satellites, atmospheric sounding devices, buoys, aviation routine weather reports, ships, and other sources. The ensemble system provides a 72-hour rainfall forecast every six hours with 5 km spatial resolution. Then the statistical artificial neural network method has been used to combine the 20 ensemble rainfall forecasts to improved 24-hour forecast. To further improve the short-term rainfall forecasts the real-time radar data has been included for the model [4].

Weather forecasts together with local weather data have been used to forecast crop frost. If the frost occurs in the growth season, the economic losses can be very significant for the farmers. The weather forecast accuracy to predict actual temperatures in the field has been researched by using several regression techniques. According to the different regression techniques used, it is not possible to predict the actual temperature in the field from the weather forecast with the accuracy needed for predicting the frost. More advanced and complex techniques have been proposed to be tested like genetic algorithm and neural network [5].

Weather forecasts have been also used for predicting the heat load for family houses [6][7]. Heating systems based on hot water circulation have the disadvantage and challenge of the long response time. It cannot react quickly enough for the outside temperature changes. By using the weather forecast as one input, it is possible to react proactively to weather changes and it can improve the comfort and reduce energy consumption.

Weather and especially rainfall forecasting is a challenging task locally. There is a lot of research done in that field. There are several methods to weather forecast. One of the most used methodologies is complex time series [8][9]. One of best-known methods for time series analysis are Exponential Smoothing [10] and Autoregressive Integrated Moving Average (ARIMA) [11].

Prediction of rainfall is a rather complex physical phenomenon. For this reason, methods such as machine learning are used today. Examples of such methods are among others Artificial Neural Network (ANN) [12], k-closest neighbor (knn) regression [13], Radial Basis Support Vector Regression ($RBSVR$) both separately and in combination as a hybrid model [14].

III. Smart pumping system

Water treatment in the peat production area is a very important and challenging task. For production, the peat needs to be dry. Water is flowing via ditches to the separated reservoirs. From the reservoirs, the water is pumped to the filtering field where the vegetation and soil is filtering the water. Water is filtered through the field that restrains the solids and nutrients before the water is flowing to the watercourses [15].

The filtration through the filtering field is more effective if the pumping and water flow through the field is as constant as possible. In traditional pumping systems, the pumping starts when the water level is high in pump pool and stops when the water level is rather low. The pumping takes place in constant speed. Therefore, the pumping and water flow is not smooth but occurs more in bursts.

With the frequency converter, it is possible to even out the pumping by slowing down the pumping speed as the water level in the pool is lowering. This is a very simple and effective method to even out the water flow through the field. It is also possible to program to the frequency converter a ramp-up time for soft starting and to lower the starting burst in water flow.

The next step to even out the pumping even more is to use the predictive pumping. When the weather forecast is predicting rain or if the rain has already started, but the water is still flowing to the pool, it could be possible to start pumping while the level in the pool has not yet reached the pumping level. This way the pool could buffer more water and the pumping of the raining water would take more time and the filtering field should filter the water more effectively.

There is a delay in water flow from the peat production area to the reservoirs. The field from which the raining water is flowed via ditches to the pool can be several tens of hectares. The flowing speed depends on, for example how dry the soil is. Dry soil can absorb more water thus lowering and slowing the water flowing to the pool.

This paper presents the implementation of the smart pumping control and the algorithm for the predictive pumping. The predictive pumping is based on the weather forecast for the area and the local weather station data together with the actual water level in the pump pool.

The construction of the IoT pump control can be seen in Fig 1. The decision is made in the ThingsBoard.io Open Source IoT platform, which is located in the cloud server. The data gathered to the IoT platform are the weather forecast (rainfall prediction from open data) data, local weather station data and the actual water level information.

The water level in the pump pool is measured with the hydrostatic level transmitter (water level transmitter). Its output signal is 2-wire 4-20 mA traditional current loop, which is widely used in automation technology due to its immunity for electromagnetic disturbance. The current loop level is converted to the digital I2C-bus with separate converter and connected to the Raspberry PI Linux computer. The Raspberry PI sends the water level data to the IoT platform (cloud) via 3G/4G cellular network.

The other data inputs for the IoT platform are local weather forecast (from Finnish Meteorological Institute) and data from local weather station located in close proximity to the pump station and peat production area.

The Finnish Meteorological Institute (FMI) offers several different data sets as Open Data. FMI offers weather forecast for 17,000 places in Finland. Weather forecast in open data is updated four times a day, thus every 6 hours. The rainfall forecast for the closest offered place from the peat production area was the main focus in this case [16].

The local weather station is located in close proximity of the pump station. Weather station sends the data to the cloud server (weatherlink) via cellular network and the data is obtained from there via interface to the IoT platform. The main factor gained from the local weather data in the predictive algorithm is the rain during the last hour. Water rained during last hour is still partly flowing to the reservoir, thus it does not immediately raise the water level in the reservoir.

The analytics from the obtained data is carried out on the IoT platform. According to the analytics, it creates the information for the current signal needed for guiding and controlling the frequency converter. This information is sent to the Raspberry PI via wireless 3G/4G connection and the Raspberry PI creates a new control signal with I2C/current (4-20 mA) converter. This current loop is connected to the frequency converter.

The flow meters are located after the filtering field. The flow meter data can be used to evaluate the smart pumping effectiveness to even out the water flow after the filtering field. It cannot be used for real time feedback to the pumping because there is a delay when the effect of pumping can be seen in water flow.

In a pilot case, the water level in the measuring well can vary from 0 meters to 3 meters. The frequency converter is programmed to keep the water level in 1.8 meters and the

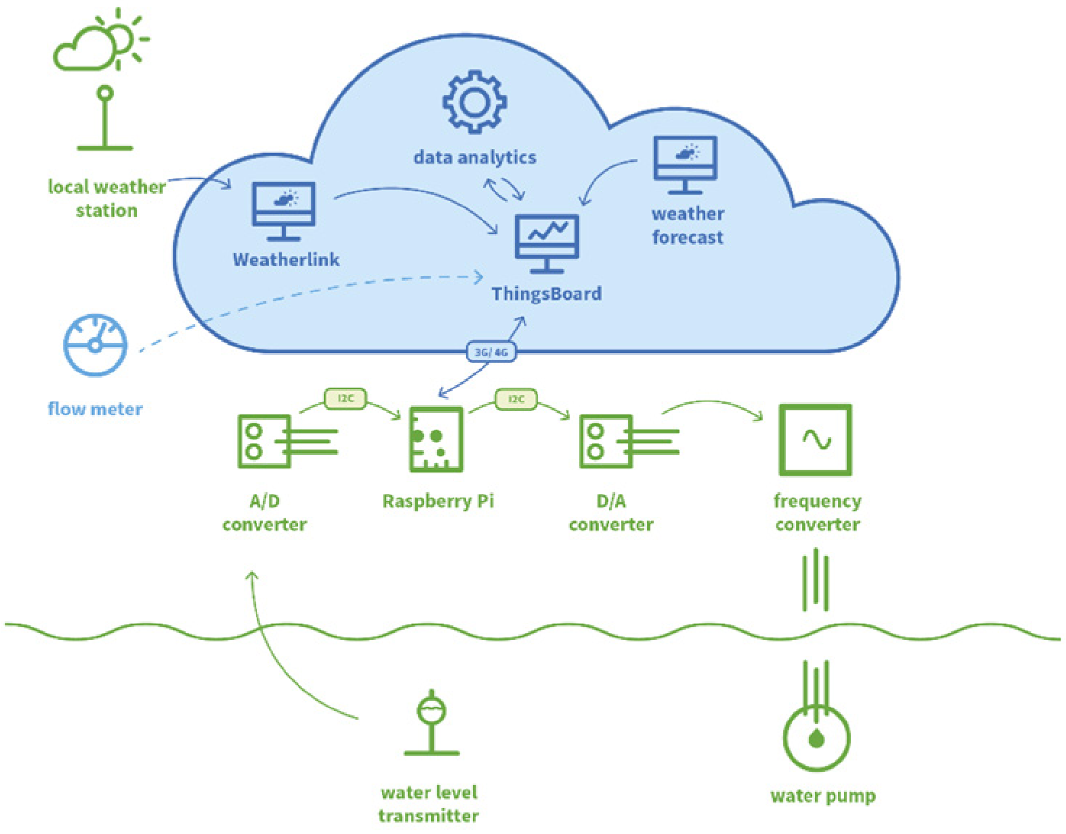


Fig. 1. The construction of the IoT pump control.

range is approximately between 1.8 m to 2.2 m. Thus, the pumping starts when the water level has reached the 2.2 meters. The pumping starts at nominal speed and it slows down stepless until the water level reaches 1.8 meters level.

The predictive pumping algorithm can start the pumping before the water level reaches the 2.2 meters level. Moreover, with predictive pumping the water level can be pumped down to 1.5 meters level to yield to raining water.

IV. Predictive Pumping Algorithm Based on Sensor Data and Weather Forecast

The algorithm used in the IoT platform is creating the information for control signal needed. The first version of the pumping algorithm is rather simple but it will be developed further in the future.

The control algorithm inputs are:

- next hour rainfall forecast from FMI open data R_1 (mm).
- last hour accumulated rainfall from the local weather station R_2 (mm).
- actual water level in pump station L_1 (mA)

The control signal L_2 (mA) for the frequency converter is:

$$L_2(L_1, X_1, X_2) = \begin{cases} L_1, \text{when } L_1 < y \\ L_1 + X_1 + X_2, \text{ when } L_1 \geq y \end{cases} \quad (1)$$

$$L_1, L_2, y \in [4,20]$$

$$X_1, X_2 \in [0,16]$$

The X_1 (mA) is the effect of the rainfall forecast on the control signal:

$$X_1(R_1) = \begin{cases} 0, \text{when } R_1 < b \\ aR_1, \text{when } b \leq R_1 \leq c \\ d, \text{when } R_1 > c \end{cases} \quad (2)$$

$$R_1, a, b, c \in [0, \infty[$$

$$d \in [aR_1, 16]$$

Where a is just a parameter to scale the control (slope) to the suitable level. The b is the minimum predicted amount of rain to take account (for example 0.5 mm/h), c is the limit for the predicted rain to be used for growing the X_1 (thus the rain prediction higher than c is not raising the X_1 anymore), d is the maximum for X_1 (for example 14 mA). Maximum value for d is 16 mA but in real situation, it needs to be much lower.

The X_2 (mA) is the effect of the last hour accumulated rain measured in local weather station:

$$X_2(R_2) = \begin{cases} 0, \text{ when } R_2 < f \\ eR_2, \text{ when } f \le R_2 \le g \\ h, \text{ when } R_2 > g \end{cases} \quad (3)$$

$$R_2, e, f, g \in [0, \infty[$$

$$h \in [eR_2, 16]$$

Where e is the parameter to scale the control level. The f is the minimum accumulated last hour rain to take into account (for example 0.5 mm/h), g is the amount of accumulated rain during last hour to be used for increasing the X_2. The h is the maximum for X_2.

The parameters used in this pilot case were y = 12 mA, a = 2, b = 0.5 mm/h, c = 7 mm/h, d = 14 mA, e = 2, f = 0.5 mm/h, g = 7 mm/h, h = 14 mA.

Thus the algorithm for weather (rainfall) prediction effect X_1 was:

$$X_1(R_1) = \begin{cases} 0, \text{ when } R_1 < 0.5 \\ 2R_1, \text{ when } 0.5 \le R_1 \le 7 \\ 14, \text{ when } R_1 > 7 \end{cases} \quad (4)$$

The effect of the local weather data:

$$X_2(R_2) = \begin{cases} 0, \text{ when } R_2 < 0.5 \\ 2R_2, \text{ when } 0.5 \le R_2 \le 7 \\ 14, \text{ when } R_2 > 7 \end{cases} \quad (5)$$

Lastly, the final control signal for the frequency converter:

$$L_2(L_1, X_1, X_2) = \begin{cases} L_1, \text{ when } L_1 < 12 \\ L_1 + X_1 + X_2, \text{ when } L_1 \ge 12 \end{cases} \quad (6)$$

The control chain in the IoT platform is presented in Fig 2. The Raspberry PI sends the water level transmitter data to the IoT platform. In the IoT platform, the effects of rainfall forecast and the last hour accumulated rain are calculated and added to the water level data if the water level is over a certain threshold level. This data is sent to the Raspberry PI in the field. If the water level is not over the threshold, the water level data is returned back to the Raspberry PI unchanged.

V. Results

The predictive pumping system has been in use during autumn 2018 for few months. Due to exceptional rain conditions during that period, reliably results and conclusions cannot be drawn yet. In addition, the predictive algorithm parameters were changed few times during the period. Originally, the parameters were set too conservatively and the predictive control did not activate easily.

Data from local weather station and weather forecast for that location have been collected during autumn 2018. The rainfall forecast and actual rainfall from local weather station is presented in Fig 3. from 9 October 2018 to 13 November 2018 in hour on an hour basis. Measuring period is rather short but at least during that observation period the correlation

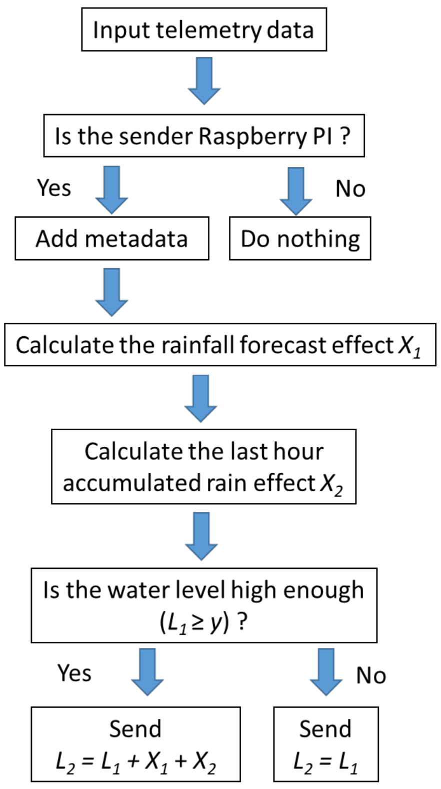

Input telemetry data
Is the sender Raspberry PI ?
Yes
No
Add metadata
Do nothing
Calculate the rainfall forecast effect X_1
Calculate the last hour accumulated rain effect X_2
Is the water level high enough ($L_1 \ge y$) ?
Yes
No
Send $L_2 = L_1 + X_1 + X_2$
Send $L_2 = L_1$

Fig. 2. Control chain in IoT platform.

between forecast and actual level is rather small. If the weather forecast is predicting rain and it actually does not start to rain, the pumping can start without the real need. Anyway, this is not a big problem; however, it gives rise to water flow after the filtering field. As a disadvantage, this system starts pumping with nominal speed. In a predictive mode, it could be better to start pumping with modest speed. This way the water flow would be more even and the unnecessary pumping (due to false rainfall prediction) would not cause as large burst in water flow after the filtering field. This is one of possible tasks in further development.

VI. Conclusions and Future Work

By using short-term (one hour) rainfall forecast we get quite good accuracy to the pump control. That way the rainfall prediction and pump control are simple. Better result to the control will be achieved by predicting rainfall using local weather information. In addition, one possibility is to use the rainfall forecast for the next few hours, not just the next hour. The forecast for following hours could have a smaller effect on the algorithm due to bigger inaccuracy.

The next step in the research will be to add the weather forecast, local weather station information and the peat bog's current capability of water absorption into the pump control algorithm. Water absorption depends on long-term rainfall and season. With these changes to algorithm, it is possible to get more even water discharge from the peat bog elsewhere to the nature. By adding water flow measurement information from the filtering field with water pumping information we get feedback from the algorithm by regression analysis.

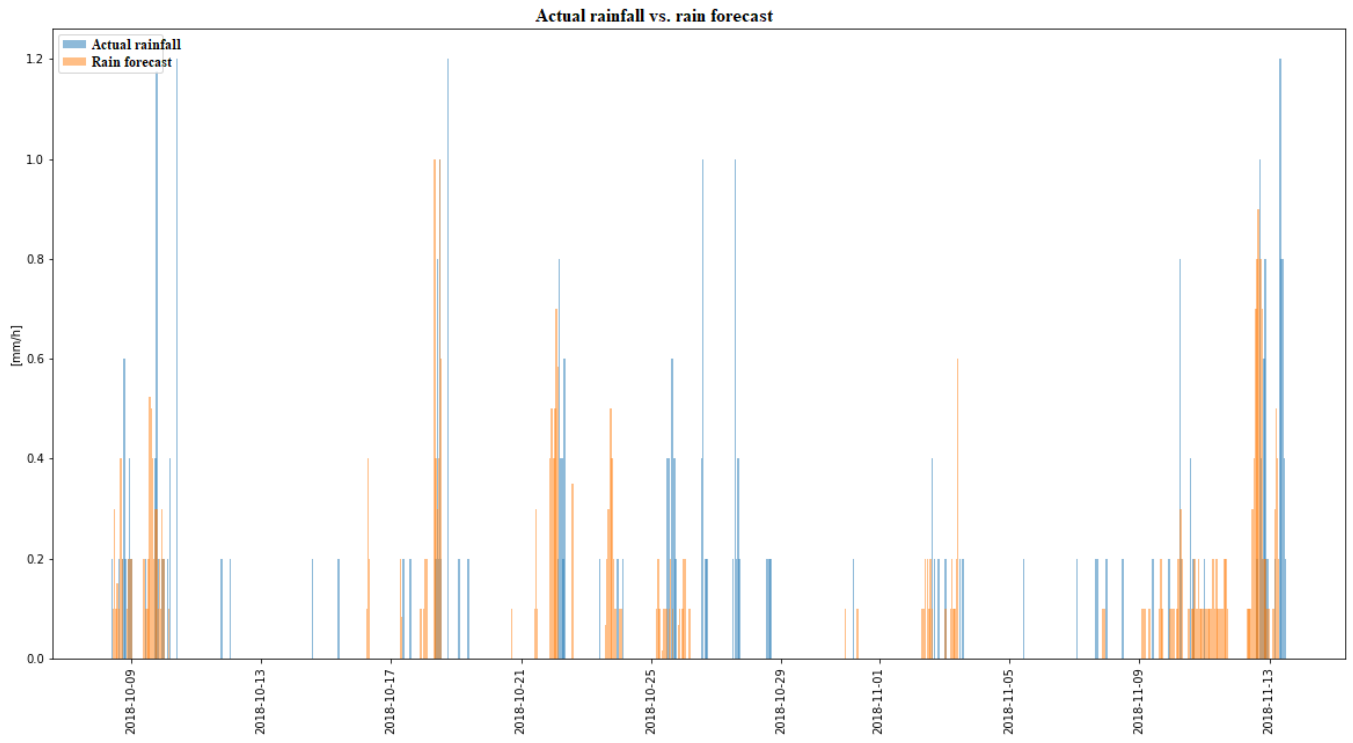


Fig. 3. Actual rainfall versus rain forecast from hour to hour.

In this system, all the intelligence and calculations are done in the cloud server (IoT platform). This kind of system requires a reliable wireless internet connection from the field to the cloud and back. The water level data is constantly sent to the IoT platform and the IoT platform sends the new control data at the same rate back to the Raspberry PI. In rural areas, the internet connection is not always reliable enough. The algorithm used is not very complex; thus, the calculations could be done locally in the Raspberry PI. This kind of approach is called edge computing. The weather forecast from open data used is updated every six hours so in edge computing approach there is no need for a constant internet connection. The amount of data in weather forecast is also rather small. It could be possible to download this amount of data to the edge device with a rather slow internet connection. Edge computing could also help significantly in possible connection shortages. Local weather station data could be connected directly to the control device without available internet connection.

REFERENCES

[1] http://turveinfo.fi/turve/turvetuotanto/turpeen-tuotanto/, accessed 15th November 2018

[2] P. Padalalu, S. Mahajan, K. Dabir, S. Mitkar and D. Javale, "Smart water dripping system for agriculture/farming," 2017 2nd International Conference for Convergence in Technology (I2CT), Mumbai, 2017, pp. 659-662. doi: 10.1109/I2CT.2017.8226212

[3] A. P. Kurniawan, A. N. Jati and F. Azmi, "Weather prediction based on fuzzy logic algorithm for supporting general farming automation system," 2017 5th International Conference on Instrumentation, Control, and Automation (ICA), Yogyakarta, 2017, pp. 152-157. doi: 10.1109/ICA.2017.8068431

[4] TH. Yang, SC. Yang, HM. Kao, MC. Wu, HM. Hsu, “Cyber-physical-system-based smart water system to prevent flood hazards,” in Smart Water 3: 1, 2018. https://doi.org/10.1186/s40713-018-0008-3

[5] M. Á. Guillén-Navarro, F. Pereñíguez-García and R. Martínez-España, "IoT-based System to Forecast Crop Frost," 2017 International Conference on Intelligent Environments (IE), Seoul, 2017, pp. 28-35. doi: 10.1109/IE.2017.38

[6] P. Bacher, H. Madsen and H. A. Nielsen, "Online short-term heat load forecasting for single family houses," IECON 2013 - 39th Annual Conference of the IEEE Industrial Electronics Society, Vienna, 2013, pp. 5741-5746. doi: 10.1109/IECON.2013.6700075

[7] M. K. Agesen et al., "Toolchain for user-centered intelligent floor heating control," IECON 2016 - 42nd Annual Conference of the IEEE Industrial Electronics Society, Florence, 2016, pp. 5296-5301. doi: 10.1109/IECON.2016.7794040

[8] N. Diodato, “Using Historical Precipitation Patterns to Forecast Daily Extremes of Rainfall for the Coming Decades in Naples (Italy)” Geosciences, 8(8), 2018, p. 293.

[9] C.L. Wu, K.W. Chau, and C. Fan, “Prediction of rainfall time series using modular artificial neural networks coupled with data preprocessing techniques,” J Hydrol, vol. 389, no. 1-2, pp. 146-167, 2010.

[10] V. Prema, "Time Series Decomposition Model for Accurate Wind Speed Forecast." Renewables: Wind, Water, and Solar 2, no. 1 (2015): 1-11.

[11] S. Zakaria, "ARIMA Models for Weekly Rainfall in the Semi-arid Sinjar District At Iraq." Journal of Earth Sciences and Geotechnical Engineering 2, no. 3 (2012).

[12] K. Abhishek, A. Kumar, R. Ranjan, and S. Kumar, “Rainfall Prediction Model using Artificial Neural Networks,” IEEEControl and Graduate Research Colloquim, 2012.

[13] M. Cristian, “Average monthly rainfall forecast in Romania by using K-nearest neighbors regression,” Analele Universității Constantin Brâncuși din Târgu Jiu : Seria Economie, 1(4), pp. 5-12.

[14] S. M. Sumi, M. F. Zaman, H. Hirose, “Rainfall forecasting method using machine learning models and its application to Fukuoka city case,” International Journal of Applied Mathematics and Computer Science, vol. . 4, pp. 841-854, 2012

[15] E. Alakangas, P. Hölttä, M. Juntunen, T. Vesisenaho, Fuel Peat Production Technology : Training material,., Publications of JAMK University of Applied Sciences 140, 2012. http://www.theseus.fi/handle/10024/126627

[16] Finnish Meteorological Institute’s open data –service https://en.ilmatieteenlaitos.fi/open-data

III

COMPRESSION METHODS FOR MICROCLIMATE DATA BASED ON LINEAR APPROXIMATION OF SENSOR DATA

by

Olli Väänänen & Timo Hämäläinen, 2019

Proceedings of the 19th International Conference on Next Generation Wired/Wireless Advanced Networks and Systems NEW2AN 2019

https://doi.org/10.1007/978-3-030-30859-9_3

Compression Methods for Microclimate Data Based on Linear Approximation of Sensor Data

Olli Väänänen[1][0000-0002-7211-7668] and Timo Hämäläinen[2][0000-0002-4168-9102]

[1] Industrial Engineering, School of Technology, JAMK University of Applied Sciences, Jyväskylä, Finland
olli.vaananen@jamk.fi
[2] Faculty of Information Technology, University of Jyväskylä, Jyväskylä, Finland
timo.t.hamalainen@jyu.fi

Abstract. Edge computing is currently one of the main research topics in the field of Internet of Things. Edge computing requires lightweight and computationally simple algorithms for sensor data analytics. Sensing edge devices are often battery powered and have a wireless connection. In designing edge devices the energy efficiency needs to be taken into account. Pre-processing the data locally in the edge device reduces the amount of data and thus decreases the energy consumption of wireless data transmission. Sensor data compression algorithms presented in this paper are mainly based on data linearity. Microclimate data is near linear in short time window and thus simple linear approximation based compression algorithms can achieve rather good compression ratios with low computational complexity. Using these kind of simple compression algorithms can significantly improve the battery and thus the edge device lifetime. In this paper linear approximation based compression algorithms are tested to compress microclimate data.

Keywords: Edge Computing, Internet of Things, Compression algorithm.

1 Introduction

Edge computing has been one of the most significant research topics in the field of Internet of Things during these years. The edge computing means that part of the data analysis is carried out in so-called edge devices. The edge devices are devices located on the edge of the network. Wireless sensor nodes are one example of typical edge devices. The edge devices are often computationally constrained and light devices [1].

Edge computing is not going to substitute the cloud computing but it is more like a supplement concept in the IoT field. Most of the data analysis has been carried out in the cloud and this will be probably the case in the future as well. As the amount of the data from the sensing devices is increasing all the time and these sensing devices are often battery or energy harvesting powered, the energy efficiency of those so called edge devices has become very important. It is known that transmitting data wirelessly from the edge device is the most energy-consuming task in these devices. It is more energy efficient to conduct some light data analysis or pre-processing locally and thus reduce the amount of data needed to send to the cloud. One possible pre-processing task for the sensor data is to filter clearly erroneous data and to compress the sensor data. The edge computing approach can also help to solve privacy and security issues concerning IoT data and offer minimized latency and improve the quality of service (QoS) [2].

There are different sensor data compression methods available. The suitability and efficiency of different methods depend on the data characteristics. Different methods differ in computational complexity, which is an important aspect in edge computing. This paper presents basic and light compression algorithms based on data linearity. Many environmental values are near linear in small time window. These compression algorithms' compression efficiency is tested for microclimate datasets. Datasets are temperature, air pressure and wind speed measurements from the Finnish Meteorological Institute's open data service.

The microclimate data is often nearly linear in short time window. For example, temperature normally changes slowly and if the measurement sampling rate is fast enough, the consecutive measurements cannot deviate much from each other [3]. Air pressure normally also changes slowly. Only approaching low pressure such as a thunderstorm, the air pressure can drop quickly [4]. Wind speed is slightly different because it can stay zero rather long periods. The wind speed also varies quite quickly and it is also quite abrupt in nature [3]. In this paper, the wind speed dataset is averaged data and thus represents more linear type data.

The microclimate data is very important for example in different agricultural applications. Agricultural applications for example for crop protection and to maximize crop production [5, 6] have been presented; however, microclimate measurements are important also in urban environment [7].

2 Lightweight Compression Methods for Sensor Data

In constrained edge devices, it is crucial to optimize resource usage. This means to optimize computational capacity, energy consumption and bandwidth usage [8]. These devices are often connected to the internet via wireless connection. Wireless transmitting is known to be the most energy-consuming task in these devices, thus it is in many cases more energy efficient to carry out data pre-processing and lightweight data analytics locally and thus reduce the amount of data needed to send via wireless link. A very simple method for reducing the amount of data is to compress the data. The other method is simply to reduce the sampling frequency of the sensor [3]. The drawback here is that information is lost between sampling points. Sampling a sensor is quite low energy operation compared to the energy consumption in radio transmission [3]. By using an effective and low computational complexity compression algorithm it is possible to keep radio transmitting rate low and thus keep the energy consumption on a low level, yet at the same time keep the accuracy of the higher sampling rate.

Typically, a simple edge device is a sensor node measuring some environmental magnitudes. Typical environmental magnitudes are for example temperature, humidity, air pressure and lightness. The measured values are then sent to the cloud and in the cloud, the data is combined with other data (for example open data) and together used for decision processing.

2.1 Lossy Methods and Lossless Methods

Sensor data compression methods are divided in lossy and lossless methods. Many different algorithms are presented for sensor data compression [9, 10]. The suitability of the compression algorithm is dependent on the sensor data characteristics. For example, many environmental magnitudes are nearly linear in short time scale, and thus some compression algorithms are more suitable for this kind of data. Some other type of data may require different types of compression algorithms.

If the reconstruction error accepted is more than zero, it is possible to use lossy compression algorithm. Compression ratio is dependent on accepted reconstruction error. Thus, the lossy compression algorithm will lead to loss of the information [11]. The advantages of lossy compression algorithms are the effective reduction of the data and in many cases, the computational simplicity. The compression and reduction of the data is done by eliminating some of the original information [11]. The accepted level of reconstruction error is very application dependent. In general, the lossy compression algorithms have higher a compression ratio together with lower computational complexity than lossless algorithms [12].

Many lossy algorithms have some latency and thus are not suitable for real-time applications. There are also lossy zero-latency compression algorithms. These compression methods are based on predictive filters (e.g. Kalman filter), which predict the data values from previous samples. In this method, the same filter is used in both sides of the network (sensor node and the user node where the data is analyzed further), thus the same estimation is used in both sides, and the new data is sent only if the value differs from the predicted value more than the tolerance level [8].

Lossless algorithms are able to reconstruct the original data without an error. The lossless methods perform two steps: the statistical model is first generated and then the second step uses this statistical model to map the input data to the bit sequences. In these bit sequences, the frequently occurred data generates a shorter output than infrequently occurred data. The two main encoding algorithms used are Huffman coding and arithmetic coding. The Huffman coding is computationally simpler and faster; however, it gives poor results in compression. Arithmetic coding is more efficient in compression but more complex. In many cases, the lossless algorithms are not suitable because the compression ratio is poor and computational complexity is higher than in lossy algorithms [13].

2.2 Lossy Compression Algorithms Based on Linear Approximation

Lossy data compression algorithms analyzed in this paper are based mainly on piecewise linear approximation. Piecewise linear approximation based compression algorithms are based on the fact that many environmental phenomena are near linear in short time window [3]. These kinds of phenomena are for example temperature, humidity, air pressure and wind speed.

A simple linear compression model is based on a regression line, which is calculated on the minimum of the first three measured values [13]. Least-squares regression line is used to approximation of discrete data [14]. In the least-squares regression line, the linear model is set to fit a set of data points. The least-squares method minimizes the sum of squares of the deviation between the data points and the fitting line thus gives a best fit to the data points. This is called a linear regression. A linear function $y = ax + b$ has two free parameters, a and b [14]. The general sum of squares of the deviation is [14]:

$$S = \sum_{k=1}^{N}[y_k - (ax_k + b)]^2 \tag{1}$$

Minimizing this equation and solving for a and b give [14]:

$$a = \frac{\sum_{k=1}^{N} x_k \sum_{k=1}^{N} y_k - N \sum_{k=1}^{N} x_k y_k}{\left(\sum_{k=1}^{N} x_k\right)^2 - N \sum_{k=1}^{N} x_k^2} \tag{2}$$

$$b = \frac{\sum_{k=1}^{N} x_k \sum_{k=1}^{N} x_k y_k - \sum_{k=1}^{N} x_k^2 \sum_{k=1}^{N} y_k}{\left(\sum_{k=1}^{N} x_k\right)^2 - N \sum_{k=1}^{N} x_k^2} \tag{3}$$

The parameters a and b give the best line fit to the N data points. To use these formulas it is needed to sum x_k, x_k^2, y_k, $x_k y_k$, and square the sum of x_k [14]. If the regression line is calculated from the first three measurements, then N is 3.

If the data is nearly linear, this regression line gives the prediction for the following measured data points with a certain error bound e. When the measured data point falls out of the error bound $\pm e$, then the new regression line is calculated. Hence, the data will be presented in piecewise linear segments.

There are several different versions of this kind of algorithm presented in literature. The algorithm is named here as Linear Regression based Temporal Compression (LRbTC). The algorithm is as follows:

1. Get the next three measured values and calculate the regression line to fit those three values.
2. Store (send) regression line point at time moment of the first measured value used to calculate regression line.
3. Get next measured value and compare it to the regression line.
4. If the difference is under the error bound e, then go to 3. Else, continue onto the next step.
5. Store (send) the regression line point when the measured value was last time under the error bound and go to 1.

Fig. 1 shows an example of this linear regression based compression for sensor data. Original temperature data is marked in blue circle. The regression line is calculated from the first three measured values (20, 20.3 and 20.1). Then following measured values are compared to the regression line value on that time moment. Regression line continues until the difference between measured value and regression line exceed the error bound *e*, which is in this example set to 0.5. Regression line is the green line in Fig. 1. At time moment 11 the difference between regression line and measured value exceeds 0.5, thus the first regression line is set to end at time moment 10. From time 1 to 10, the compressed data includes only the starting point of the regression line and the end point of that line. The next regression line is calculated from the measured values in time moments 11 to 13. At time moment 15, the difference exceeds the error bound and thus the new line is calculated from the values at time moments 15-17. In 20, the difference exceeds again the error bound. The first 19 measured values (time moments 1 to 19) are compressed to 6 values (three regression lines).

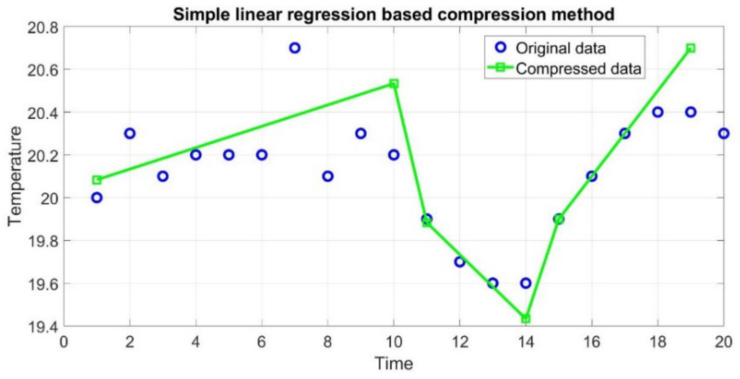


Fig. 1. Linear Regression based Temporal Compression (LRbTC) algorithm example.

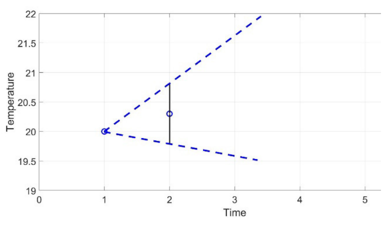
In the example in Fig. 1, the error bound was set to 0.5. Thus, the measured value and regression line value should not exceed 0.5. This can anyway happen in time moments that are used to calculate the regression line.

The modified version of the LRbTC (M-LRbTC) algorithm corrects the problem if the difference between regression line and the data values used to calculate this regression line exceed the error bound *e*. The modified version of the algorithm is as follows:

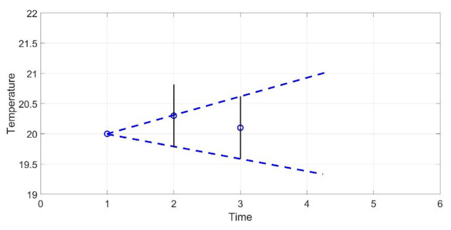
1. Get the next three measured values and calculate the regression line to fit those three values.
2. Compare the regression line and three values used to calculate the line.
3. If the difference is greater than error bound *e,* then store (send) the first two data points and get the next two measurement values and calculate new regression line and go to 2, else continue onto the next step.
4. Get the next measured value and compare it to the regression line.
5. If the difference is under the error bound *e*, then go to 4. Else, continue onto the next step.
6. Store (send) the last regression line point when the measured value was under the error bound and go to 1.

Lightweight temporal compression (LTC) was introduced in [3]. It is simple and very efficient compression algorithm for microclimate type data in a small enough time window. LTC's effectivity to compress data depends on the data characteristics. For linear type environmental data, it can obtain up to 20-to-1 compression ratio [3]. Compression ratio is also dependent on error bound used. It is recommended to use the sensor manufacturer's specified accuracy value as the error bound in LTC algorithm [3]. For example if the sensor used is a temperature sensor with 0.5 degrees accuracy, it is reasonable to use 0.5 as the error bound.

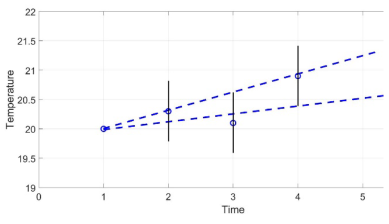
The LTC algorithm is explained in detail in [3, 11, 15, 16] and a modified version in [17]. The linear model starts with the first data value as a starting point. The lower line and upper line (limit lines) are drawn from the starting point to the next measured value ±*e* as seen in Fig. 2 a. The limit lines are tightened from the following values when error bound extreme or extremes are inside the previous limit lines as in Fig. 2 b. and c. The measured data is discarded from the linear model if the measurement cannot fit inside upper line and lower line determined by the previous data with the error bound ±*e*. Then the new linear model starts using as a starting point the middle point of the upper line and lower line in last time moment included in the linear segment. This procedure of the algorithm can be seen in Fig. 2 d.



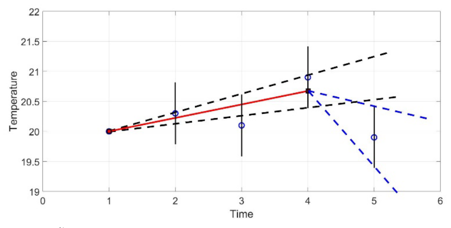
a)



b)



c)



d)

Fig. 2. Lightweight temporal compression (LTC) algorithm.

The reconstruction error never exceeds the error bound *e* in the LTC algorithm. The LTC algorithm has low computational complexity and thus it is suitable for constrained edge devices such as sensor nodes [17]. In Fig. 3, the LTC is compared to previously presented linear regression based algorithm.

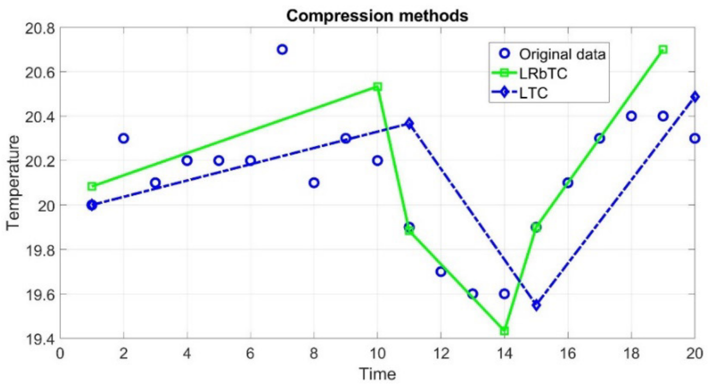


Fig. 3. LTC compared to the basic linear regression based algorithms.

The disadvantage of the LTC is that it is not well suited for real-time applications [8] and its suitability in general is very application dependent. LTC uses linear interpolation to represent the original signal, and the linear interpolation model is known only when the both extremes of the linear part is known. This introduces significant latency for the model [8]. The linear regression based algorithms presented previously suffer from the same problem.

2.3 Transform Based Compression Methods

Discrete Fourier Transform (DFT) is a well-known transform based algorithm. It is simple to use for compression by using the Fast Fourier Transform (FFT) algorithm [18]. The FFT algorithm expresses the time-series signal in frequency representation. By removing the coefficients with less energy, it is possible to reduce the amount of data and still keep the information to rebuild the time series data with reasonable reconstruction error. When the FFT is taken over a window of *N* samples and the first sample and last sample differ a lot, the information of discontinuity is spread across the frequency spectrum. To prevent this discontinuity it is possible to overlap the windows [18].

Another well-known transform based algorithm is Discrete Cosine Transform (DCT) and Modified Discrete Cosine Transform (MDCT) [12, 18, 19]. It has several advantages compared to FFT algorithm [18]. The DCT coefficients are real numbers; thus there is no need to deal with complex numbers. This saves memory and is less complex. The DCT also has the information concentrated to the few low-frequency components and the DCT does not suffer the edge discontinuity problem like FFT. The DCT is a well known and widely used compression algorithm for example in image compression and for time series type sensor data.

3 Testing the Algorithms with Real Microclimate Data

The linear approximation based compression algorithms are tested for microclimate type data and compared to the DCT algorithm. The datasets tested here are gathered from the Finnish Meteorological Institute's open data service [20]. Finnish Meteorological Institute has about 400 observation stations in Finland. Not all the stations have the same measured variables. For this research, Salla Naruska station's data from year 2018 in 10 minutes time sampling rate was chosen. The variables chosen were temperature, air pressure and wind speed. Salla Naruska measurement station is located in eastern Lapland and known as one of the coldest places in Europe. The exact situation of the station is: latitude 67.16226, longitude 29.17766 in decimal degrees. The temperature is in Celsius degrees (ºC), air pressure in hectopascals (hPa) and wind speed in meters per second (m/s). Th wind speed is measured in 10 minutes average. All variables are measured with one decimal resolution.

One year measurements in 10 minutes time interval mean 51,961 measurements for each variable. Some data was missing; however, the missing points were linearly interpolated. In air pressure data in total 102 points were missing, in temperature data 101 points were missing and in wind speed data 1,077 points were missing. For comparison, also the same data in one-hour measurement interval was used. This one-hour interval data for the whole year 2018 includes 8,761 measurements points for each magnitude. The missing values were also linearly interpolated.

The compression algorithms chosen were simple linear regression based approximation algorithm (LRbTC), modified linear regression based algorithms (M-LRbTC) and lightweight temporal compression (LTC). Basic discrete cosine transform (DCT) was used for comparison.

The algorithms were tested with MATLAB simulation. LRbTC, M-LRbTC and LTC algorithms were programmed in MATLAB by using mainly functions *polyfit* and *polyval*. *Polyfit* function was used for linear regression in LRbTC, and M-LRbTC and to create upper and lower lines in LTC instead of equations 2 and 3 [21].

Discrete cosine transform (DCT) was tested by using the MATLAB built-in function *dct*. In this example, the DCT was used with window of five measured values to calculate DCT. It was then tested with different threshold values to cancel the smallest coefficient values. After rebuilding the signal, the maximum difference (variation) from the original values was calculated.

Algorithms were compared to each other by compression ratio versus reconstruction error. The compression ratio (*CR*) was calculated by:

$$CR = \frac{original\ data}{compressed\ data} \tag{5}$$

Thus, the bigger the *CR* value is, the more efficient the compression algorithm is. The *CR* varies significantly according the error bound *e* used.

Temperature data was tested first. In the total 51,961 measured values the highest temperature was +30.4 ºC and the lowest temperature -33.8 ºC. With 10-minute measurement interval, the temperature data is mostly near linear; however, in some extremes the temperature changes quite a lot between consecutive measurements.

LRbTC algorithm showed very big reconstruction errors for tested temperature data, which is because when measuring the regression line, the values used to calculate may differ from the regression line more than the error bound *e* used. The data used is with 10-minute interval and in extremes, the values may differ significantly from measurement moment to the next moment. Higher measurement sampling rate would help the situation.

The modified version of the basic linear regression based algorithm (M-LRbTC) works as it is intended. It was tested with different quantity of measurement values to calculate the regression line. In Fig. 4, the red line is used with three values to calculate the regression line; cyan line is with four values and magenta with five values.

Fig. 4 illustrates the results for M-LRbTC, LTC and DCT algorithms for temperature data. With typical error bound *e* = 0.5 ºC, the M-LRbTC algorithm can achieve 3.9-4.8 compression ratio. LTC is significantly more effective with *CR* = 9.5. DCT was tested with five values time window to calculate DCT. DCT compression ratio is significantly lower compared to the other tested algorithms. DCT suffers from the small window used and it can achieve higher compression ratios with bigger window used. Small time window was chosen to be more realistic for sensor data stream.

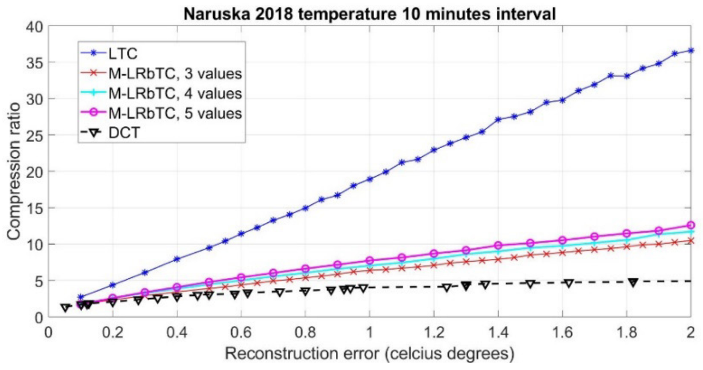


Fig. 4. Linear approximation based compression algorithms and DCT for temperature data.

Fig. 5 shows the results for the air pressure data. Air pressure values varied between 976.4 hPa – 1056.4 hPa. The variation is nearly linear in short time window, however, the data includes few clear errors. Three times the air pressure value changes from measurement to the next more than it is normally possible. The biggest difference in consecutive measurements is 12.7 hPa (in 10 minutes), which is clearly an error. Normally the air pressure can change up to 5 hPa/hour and only in some very quickly progressing low pressure it can be more than 5 hPa/hour [4]. The quick changes in air pressure data can be due to clear measurement error or for example due to sensor calibration. The results for the air pressure data are similar to the temperature data, except the compression ratios are much higher. This indicates that the air pressure data is behaving very linearly with the 10-minute measurement interval. The LTC algorithm can achieve high compression ratios.

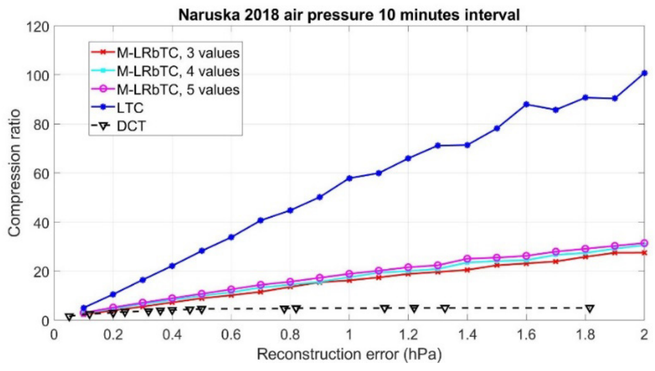


Fig. 5. Linear approximation based compression algorithms and DCT for air pressure data.

In Fig. 6 are the results of the wind speed measurements. Wind speed is measured in 10-minute average values. Wind speed is a different characteristic compared to the temperature and air pressure data. Wind speed can remain quite a long period in 0 m/s. Wind speed can also change quickly and quite significantly; however, here the 10-minute average measurement averages the results significantly. The compression ratios for wind speed data are on the same level as for the temperature data.

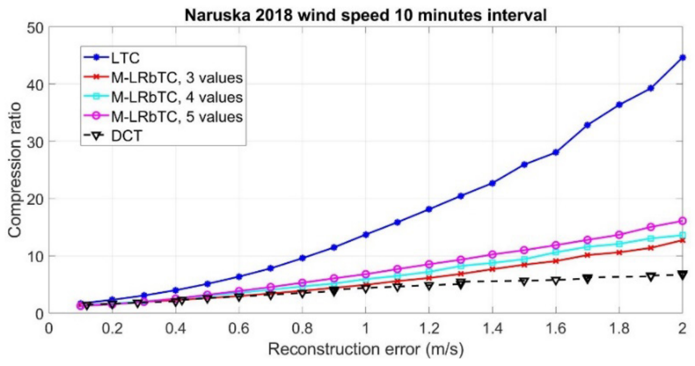


Fig. 6. Linear approximation based compression algorithms and DCT for wind speed data.

In every comparison, it can be seen that LTC is the most effective compression method. M-LRbTC also works well and it is a very simple algorithm and easy to apply. Table 1 illustrates a comparison of the compression algorithms between two different datasets with error bound *e* set to 0.5 for each quantity. The datasets are the same 10-minute interval sets as used previously and also with 1 hour measurement interval. It can be seen in table 1 that all compression algorithms are significantly more effective for 10-minute sampling rate data. This is because with 10-minute sampling rate, the data behaves more linearly.

Table 1. Comparison of the compression ratios for 10 min and 1 hour interval datasets.

Compression algorithm	Temperature (e = 0.5 °C)		Air pressure (e = 0.5 hPa)		Wind speed (e = 0.5 m/s)	
	10 min	1 hour	10 min	1 hour	10 min	1 hour
M-LRbTC, 3 values	3.9	1.85	8.94	3.05	2.62	1.88
M-LRbTC, 4 values	4.46	1.95	9.94	3.45	3.01	2.04
M-LRbTC, 5 values	4.78	1.86	10.75	3.79	3.18	1.97
LTC	9.49	3.19	28.22	8.28	5.09	3.03
DCT	3.07	1.75	4.63	2.72	2.6	1.8

The disadvantage in these linear approximation based algorithms is the latency. These methods are not directly suitable for real-time applications. LRbTC based methods are possible to modify to work better for almost real-time operations: After calculating the new regression line, the first point of the line and line coefficients can be sent. The receiver can use that information until the new point and line are received. Thus, the latency is in maximum when the new regression line is calculated, and it depends on how many point data is used for calculating the regression line.

4 Conclusions and Future Work

Compression algorithms were tested with some real measurement data. In this case, the environmental microclimate data such as temperature, air pressure and wind speed were used. Many environmental quantities are near linear in nature at least if the observation window is short. Linear approximation based compression algorithms benefit from this environmental data behavior. In this research, it was shown that these simple compression algorithms are rather efficient for this kind of data. The performance of compression algorithms for compression compared to reconstruction error was the main property to compare. The next step will be to test these algorithms in edge devices and to take into account the computational complexity of the algorithms.

References

1. Väänänen, O., Hämäläinen, T.: Requirements for Energy Efficient Edge Computing: A Survey. In: The 18th International Conference on Next Generation Wired/Wireless Advanced Networks and Systems NEW2AN 2018, August 29 - 31, 2018, St.Petersburg, Russia. doi: 10.1007/978-3-030-01168-0_1
2. Alrowaily, M., Lu, Z.: Secure Edge Computing in IoT Systems: Review and Case Studies. In: 2018 IEEE/ACM Symposium on Edge Computing (SEC), Seattle, WA, pp. 440-444. (2018). doi: 10.1109/SEC.2018.00060
3. Schoellhammer, T., Osterwein, E., Greenstein, B., et al.: Lightweight temporal compression of microclimate datasets. In: Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks IEEE Computer Society, pp. 516-524. (2004)
4. Finnish Meteorological Institute, https://ilmatieteenlaitos.fi/ilmanpaine
5. Norbu, J., Pobkrut, T., Siyang, S., Khunarak, C., Namgyel, T. and Kerdcharoen, T.: Wireless Sensor Networks for Microclimate Monitoring in Edamame Farm. In: 2018 10th In-

ternational Conference on Knowledge and Smart Technology (KST), Chiang Mai, pp. 200-205. (2018) doi: 10.1109/KST.2018.8426200

6. Muhammad, A. R., Setyawati, O., Setyawan R. A. and Basuki, A.: WSN Based Microclimate Monitoring System on Porang Plantation. In: 2018 Electrical Power, Electronics, Communications, Controls and Informatics Seminar (EECCIS), Batu, East Java, Indonesia, pp. 142-145. (2018) doi: 10.1109/EECCIS.2018.8692849
7. Rathore, P., Rao, A. S., Rajasegarar, S., Vanz, E., Gubbi J. and Palaniswami, M.: Real-Time Urban Microclimate Analysis Using Internet of Things. In: IEEE Internet of Things Journal, vol. 5, no. 2, pp. 500-511, April 2018. doi: 10.1109/JIOT.2017.2731875
8. Giorgi, G.: A Combined Approach for Real-Time Data Compression in Wireless Body Sensor Networks. In: IEEE Sensors Journal, *vol.* 17, no. 18, pp. 6129-6135, 15 Sept.15, 2017.
9. Bose, T., Bandyopadhyay, S., Kumar, S., Bhattacharyya, A., Pal, A.: Signal Characteristics on Sensor Data Compression in IoT - An Investigation. In: 2016 IEEE International Conference on Sensing, Communication and Networking (SECON Workshops), pp. 1-6. London (2016)
10. Ying, Y. B.: An energy-efficient compression algorithm for spatial data in wireless sensor networks. In: 2016 18th International Conference on Advanced Communication Technology (ICACT), *P*yeongchang, pp. 161-164. (2016) doi: 10.1109/ICACT.2016.7423312
11. Fallah, S. A., Arioua, M., El Oualkadi, A. and El Asri, J.: On the performance of piecewise linear approximation techniques in WSNs. In: 2018 International Conference on Advanced Communication Technologies and Networking (CommNet), Marrakech, pp. 1-6. (2018)
12. Jaafar K. Alsalaet, Abduladhem A. Ali, Data compression in wireless sensors network using MDCT and embedded harmonic coding, ISA Transactions, Volume 56, Pages 261-267, (2015) ISSN 0019-0578, https://doi.org/10.1016/j.isatra.2014.11.023.
13. Aggarwal, Charu C.: Managing and Mining Sensor Data. Springer. 2013. DOI: 10.1007/978-1-4614-6309-2
14. Lopez, Robert J. Advanced Engineering Mathematics. Addison-Wesley. United States of America (2001). ISBN: 0-201-38073-0
15. Sharma, R.: A data compression application for wireless sensor networks using LTC algorithm. In: 2015 IEEE International Conference on Electro/Information Technology (EIT), *Dekalb,* IL, pp. 598-604. (2015) doi: 10.1109/EIT.2015.7293435
16. Azar, J., Makhoul, A., Darazi, R., Demerjian, J. and Couturier, R.: On the performance of resource-aware compression techniques for vital signs data in wireless body sensor networks. In: 2018 IEEE Middle East and North Africa Communications Conference (MENACOMM), Jounieh, pp. 1-6. (2018) doi: 10.1109/MENACOMM.2018.8371032
17. Parker, D., Stojanovic, M. and Yu, C.: Exploiting temporal and spatial correlation in wireless sensor networks. In: 2013 Asilomar Conference on Signals, Systems and Computers, *Pacific G*rove, CA, pp. 442-446. (2013) doi: 10.1109/ACSSC.2013.6810315
18. Zordan, D., Martinez, B., Vilajosana, I. and Rossi, M.: On the Performance of Lossy Compression Schemes for Energy Constrained Sensor Networking. In: ACM Trans. Sen. Netw. 11, 1, Article 15, 34 pages. (2014). doi: http://dx.doi.org.ezproxy.jyu.fi/10.1145/2629660
19. Tan, L.: Digital Signal Processing: Fundamentals and Applications. Academic Press, Elsevier, United States of America (2008), ISBN: 978-0-12-374090-8
20. Finnish Meteorological Institute's open data –service. https://en.ilmatieteenlaitos.fi/open-data
21. Matlab polyfit function documentation. https://se.mathworks.com/help/matlab/ref/polyfit.html

IV

LINEAR APPROXIMATION BASED COMPRESSION ALGORITHMS EFFICIENCY TO COMPRESS ENVIRONMENTAL DATA SETS

by

Olli Väänänen, Mikhail Zolotukhin & Timo Hämäläinen, 2020

Web, Artificial Intelligence and Network Applications. WAINA 2020. Advances in Intelligent Systems and Computing, vol. 1150, pp. 110-121, Springer, Cham.

https://doi.org/10.1007/978-3-030-44038-1_11

Linear Approximation Based Compression Algorithms Efficiency to Compress Environmental Data Sets

Olli Väänänen[1], Mikhail Zolotukhin[2] and Timo Hämäläinen[2]
[1] Industrial Engineering, School of Technology, JAMK University of Applied Sciences, Jyväskylä, Finland
olli.vaananen@jamk.fi
[2] Faculty of Information Technology, University of Jyväskylä, Jyväskylä, Finland
mikhail.m.zolotukhin@jyu.fi, timo.t.hamalainen@jyu.fi

Abstract. Measuring some environmental magnitudes is a very typical application in the field of Internet of Things. Wireless sensor nodes measuring these environmental magnitudes are often battery powered devices. Thus, the energy efficiency is an important topic in these measuring devices. The most efficient method to reduce energy consumption in wireless devices is to reduce the amount of data needed to transmit via wireless connection. A simple method to reduce the amount of the data is to compress sensor data. Environmental data behaves quasi linearly in short time window and many compression algorithms utilize this data behavior. In this paper the different environmental data sets characteristics and their effect on compression algorithms' compression ratio are evaluated. The results can be used to evaluate and choose the suitable compression algorithm for the application and to predict the lifetime of the battery powered device.

1 Introduction

In the field of Internet of Things (IoT), sensors measuring some environmental magnitudes are very typical applications. The IoT applications measuring and utilizing some environmental data can be found and used in many sectors in the society. The need to measure some environmental magnitudes is especially typical in agricultural applications [1]. In agricultural applications, the devices are often spread across the field and thus the resources available are often limited, e.g. reliable power supply and good quality wireless connections, which often also means limited computational power.

In agriculture, the Internet of Things applications can be used for e.g. crop management, crop protection, soil monitoring and water management. [2, 3] Many IoT applications and solutions in the field of agriculture are still in their infancy; however, the field is changing very fast [4].

Energy efficiency and energy saving are very important aspects in battery powered wireless sensor nodes [5, 6]. One very efficient way to reduce energy consumption in wireless sensor nodes is to compress the sensor data. By compressing the sensor data, it is possible to reduce the amount of data needed to transmit via a wireless connection. Wireless connection is known to be the most energy consuming operation in the wireless sensor node. [7]

Compressing the amount of actual sensor data needed to transmit wirelessly is only one way to reduce energy consumption. [6] However, due to the simplicity of many compression methods presented, it is a very easy and powerful method for maximizing the lifetime of a battery powered device.

In this paper, several data linearity-based compression methods have been evaluated and compared to each other. The efficiency of the compression methods to compress certain environmental data sets are evaluated and the effect of the data sets' characteristics in the compression ratio achieved have been evaluated. The correlation of certain data sets' characteristics to the compression ratio has been evaluated. With the correlation found, it is possible to choose the suitable compression method for a certain application.

2 Compression Methods Based on Time Series Data Linearity

Various sensor data compression methods have already been introduced several decades ago. After the proliferation of the wireless sensor networks (WSN) and the Internet of Things (IoT), the topic of sensor data compression has received a great deal of new attention in the field of research. [8] Typical sensor data sets are for example some environmental variable data sets such as temperature, humidity, air pressure and wind speed. Additionally, different Wireless Body Sensors measuring some parameters or behavior of the human body have gained much attention [9]. They are used in different wearable and wellness devices and applications.

There are various types of compression methods presented in research papers. There are time domain and transform domain methods. Well-known transform domain methods are for example Wavelets, Chebyshev Transform, Discrete Fourier Transform (DFT) and Discrete Cosine Transform (DCT). Many times the domain methods are based on data linearity. Linearity based methods are for example Piecewise Linear Approximation (PLA), Lightweight Temporal Compression (LTC), Piecewise Aggregate Approximation (PAA) and Piecewise Constant Approximation (PCA). These methods are lossy compression methods. [6, 10-12]

Transform domain methods are not well suited for constrained wireless sensor nodes due to their computational complexity and limited memory. [10] Many compression methods such as LTC also suffer from latency and are not well suited for real-time or near real-time applications [13]. DFT and DCT also suffer latency dependent on the window size *N* used.

Even though linear approximation and data linearity-based methods are well known and simple methods, there is recent and ongoing research on the topic. Several different variations of these methods have been introduced during recent years. [7, 11, 14, 15]

The LTC is very efficient compression method for environmental data with a linear nature at least in short time window. Its compression ratio can be quite high. For temperature data with 10 minutes measuring rate and error bound $\varepsilon = 0.5$ °C, the LTC algorithm can achieve a compression ratio 10 to 1. The compression ratio is very dependent on the data set characteristics and the error bound used. [11] At the same time, it is rather a simple compression algorithm and thus it can be used in constrained IoT devices with limited memory and processing power.

3 Effect of Environmental Data Set on Compression Ratio

Many environmental magnitudes behave near linearly if the observation window is short enough. [16] For example, if the environment microclimate temperature is rising, it can be predicted that it will continue rising at least in the near future. This linear behavior can be used to compress the amount of data needed to transmit from the sensor node. Simple sensor data compression methods utilizing this behavior are based on data linearity. Perhaps the simplest compression algorithm for this kind of data is to use linear regression of n ($n \geq 3$) measured values and with allowing certain error bound $\pm\varepsilon$ to the calculated regression line. The calculated regression line with error bound can be used to predict the following values. There are many different versions of linearity-based compression methods presented in the literature. [11, 16, 17] This kind of methods are lossy methods.

Environmental data has a linear behavior if the observation window is short. The more constant the measured magnitude remains, the more efficiently these data linearity-based compression algorithms compress the data. [12] However, the environmental magnitudes do not remain constant; instead, the values are mostly changing. There is natural variation in the values of environmental magnitudes in function of time but with allowing some random variation in values, the main trend is often rather stable for some time period. Many compression methods utilize this behavior.

3.1 Data Set Characteristics Evaluated

In this paper, the efficiency of different linearity-based time series compression algorithms to compress environmental data is tested for different environmental data sets. Data sets' characteristics are evaluated, and the parameters affecting the compression ratio have been evaluated.

The tested and evaluated data set parameters are:

- *AC,* the average absolute change between consecutive measurements in the whole data set
- *SD*, the standard deviation of the change between consecutive measurements in the whole data set

For the measured values x_i: $i \in [1, n]$, the average change (*AC*) between consecutive measurements is calculated with the equation (1):

$$Average\ change\ =\ AC\ = \frac{\sum_{i=1}^{n-1} |x_{i+1} - x_i|}{n-1}. \quad (1)$$

Standard deviation (*SD*) is calculated from the consecutive measurement change values with the equation (2):

$$Standard\ deviation \ = \ SD \ = \ \sqrt{\frac{\sum_{i=1}^{n-1}\left((x_{i+1}-x_i)-(\overline{x_{i+1}-x_i})\right)^2}{n-2}}. \tag{2}$$

where,

$$(\overline{x_{i+1}-x_i}) \ = \ \frac{1}{n-1}\sum_{i=1}^{n-1}(x_{i+1}-x_i)\,. \tag{3}$$

3.2 Data Sets

The used data sets were gathered from the Finnish Meteorological Institute's (FMI) open data service [18]. The data sets gathered from FMI service were Naruska measurement station data from whole year 2018. The Naruska measurement station is located in Eastern Lapland in Finland. It is one of the official measurement stations in Finland. Temperature, air pressure and wind speed data with a 10-minute measurement interval were used. The data sets were divided into monthly data sets, and the whole year data set was also used. 20-minute, 30-minute, 40-minute, 50-minute and 1-hour measurement interval data sets were derived from the original 10-minute measurement interval data set by cancelling the values from the original data set.

Thus, there were in total 78 data sets for each environmental variable (temperature, air pressure and wind speed).

The whole year 2018 data set with a 10-minute measurement interval was the largest data set with 51 961 measured values for each variable. The smallest data set used was February 2018 with 1-hour measurement interval with 672 measured values for each variable.

The average change *AC* values and standard deviation *SD* values were compared to the compression ratios achieved with different time series compression algorithms. The compression ratios were calculated with the equation (4):

$$CR \ = \ \frac{original\ data}{compressed\ data}. \tag{4}$$

where the *original data* is the amount of values in original data set and the *compressed data* is the amount of values in a compressed data set.

4 Compression Algorithms' Compression Ratio Compared to the Characteristics of Selected Data Set

Compression algorithms tested and evaluated were Lightweight Temporal Compression (LTC) [16] and Linear Regression based Temporal Compression (LRbTC) [11]. The LTC algorithm is originally presented in reference [16]. LTC uses the piecewise linear function to estimate data points. LTC calculates the upper and lower bound from every new data point by using the selected error bound. The LTC algorithm is explained in detail in references [7, 11, 16]. LRbTC algorithm uses the *n* measured values to calculate the regression line which can be used to predict following values with allowing a certain error bound ±ε from the line. When the measured value falls out from the allowed area, the new regression line is calculated which predicts the future values. [11] LRbTC algorithms were tested with 3, 4 and 5 values used to calculate the linear regression line. The error bound used was 0.5 ºC for temperature data sets, 0.5 hPa for air pressure data sets and 0.5 m/s for wind speed data sets. The compression ratios achieved were compared to the data sets' characteristics *SD* and *AC,* which have been previously explained in this paper. The compression algorithms were programmed on Matlab as in reference [11]. The LRbTC algorithm used was the slightly modified version M-LRbTC [11], and the LTC algorithm used was the original version originally presented in the reference [16].

4.1 Temperature Data Sets

For temperature data sets (78 data sets in total) the results can be seen in Fig. 1. Fig. 1 presents the compression ratio for each temperature data set with LTC and LRbTC algorithms. Discrete Cosine Transform (DCT) algorithm with window size of 5 values was used just for comparison. The results are presented in the function of the standard deviation (*SD*) of the data set's consecutive measurements change (on the left) and in the function of average change (*AC*), as previously explained in this paper. The trend line (solid line) is visually the best fit polynomial regression line of the data points presented.

Fig. 1 clearly indicates that the LTC is the most effective compression algorithm compared to the others. The similar results have been achieved in reference [11]. The highest compression ratio (19.24) has been achieved with LTC algorithm from December 2018 data set with a 10-minute measurement interval. The highest compression ratio with LRbTC is 8.21 from the same data set. LRbTC with 3 values used to calculate regression line is slightly worse than the versions with 4 and 5 values used to calculate the regression line. The difference between 4 and 5 values used to calculate regression line is almost negligible.

The correlation between the compression ratio and the standard deviation of the consecutive measurements change is clear; however, the correlation is not linear. A small standard deviation means that the value changes are small from measurement to measurement, which means more constant and linearly behavior data. When the SD value decreases from the value 1 to 0.2, the compression ratio raises strongly.

In Fig. 1 on the right side, the same data sets' compression ratios were compared to the average change (*AC*) in the absolute value of the consecutive measurements' change as explained previously in this paper. Similar results can be seen with the *AC* as with the *SD*. The correlation is similar as in *SD* comparison except the dispersion is smaller in *AC* comparison. Thus, it seems that the *AC* predicts the compression ratio achieved better than the *SD*.

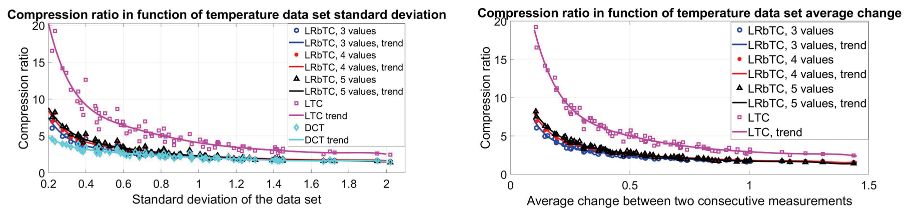


Fig. 1. Compression ratio in function of *SD* (on the left) and *AC* (on the right) from the temperature data sets.

The trend lines (fitting lines) in Fig. 1 are 8th degree polynomials (y = p1*x^8 + p2*x^7 + p3*x^6 + p4*x^5 + p5*x^4 + p6*x^3 + p7*x^2 + p8*x + p9). The 8th degree polynomials were chosen here because they give visually the best fit for the data. Additionally, the norm of residuals value, which is the measure of the goodness of the fit, was best or almost the best of the basic fitting functions. The smaller the norm of residuals value is, the better the fit. The polynomial coefficients and norm of residuals values for each compression algorithm in function of *SD* and *AC* can be seen in Table 1 and Table 2.

Table 1. Correlation between the compression ratio and standard deviation for temperature data sets, polynomial coefficients and the norm of residuals values.

Coefficients	LRbTC, 3 values	LRbTC, 4 values	LRbTC, 5 values	LTC
p1	6.7423	6.2854	6.8386	6.548
p2	-72.267	-67.776	-70.398	-80.376
p3	325.72	307.75	306.91	406.94
p4	-805.25	-767.62	-739.53	-1118.2
p5	1192.3	1148.7	1076.8	1831.5
p6	-1081.4	-1055.3	-971.24	-1837.2
p7	587.81	583.09	533.42	1109.9
p8	-178.12	-180.98	-167.79	-378.84
p9	26.555	28.115	27.281	63.733
Norm of residuals	2.4946	2.5065	2.5082	7.0922

Table 2. Correlation between the compression ratio and average change for temperature data sets, polynomial coefficients and the norm of residuals values.

Coefficients	LRbTC, 3 values	LRbTC, 4 values	LRbTC, 5 values	LTC
p1	177.09	181.2	141.07	35.377
p2	-1166.5	-1174.1	-896.79	-331.22
p3	3232.7	3207.3	2407.7	1253.5
p4	-4906.3	-4813.2	-3566.6	-2552
p5	4447.9	4335.5	3197.7	3089
p6	-2465.2	-2405	-1791.4	-2295
p7	821.88	811.44	625.16	1036
p8	-156.24	-158.91	-131.74	-270.25
p9	16.452	17.68	16.636	37.549
Norm of residuals	1.4334	1.3387	1.4053	3.9424

The norm of residuals values demonstrate that the correlation between compression ratio and *AC* is better than with *SD*.

If the data values are varied a great deal from measurement to measurement, it means faster changes in data in function of time. This indicates that this kind of data have higher frequencies. The frequency spectrum of the data set can be calculated with Discrete Fourier Transform (DFT). The data set DFT was calculated in Matlab with FFT (Fast Fourier Transform) function. The comparison of the frequency spectrum of two very different temperature data sets can be seen in Fig. 2. The red line is FFT from data set: Naruska July 2018 with a 1-hour interval. It has the standard deviation SD = 2.021 and average change *AC* = 1.437. The compression ratio with LTC is 2.439. The blue line is the FFT from data set: Naruska December 2018 with a 10-minute interval. It has the *SD* = 0.235 and *AC* = 0.106. The compression ratio with LTC algorithm is 19.241. Data sets have been normalized as both have been sampled with the same sampling rate. The frequency spectrums indicate that December 2018 with the 10-minutes measurement interval behaves more linearly because it has lower energy in high frequencies. The higher levels in high frequencies in July 2018 data indicate quick changes from measurement to measurement. Both data sets can be seen in function of time in Fig. 3.

In summertime, the temperature is changing on a daily basis approximately 15-20 degrees according to July 2018 measurement as can be seen in Fig. 3 on the left. With 1-hour measurement interval that means a significant change in value between two consecutive measurements. In wintertime, the change is not that big daily and there are long periods when the temperature remains quite constant as can be seen from December 2018 data (Fig. 3 on the right side) and specially when the measurement interval is short like 10 minutes in this example, then the data behaves quite linearly and remains during many consecutive measurements quite constant. Standard deviation and average change values indicate this.

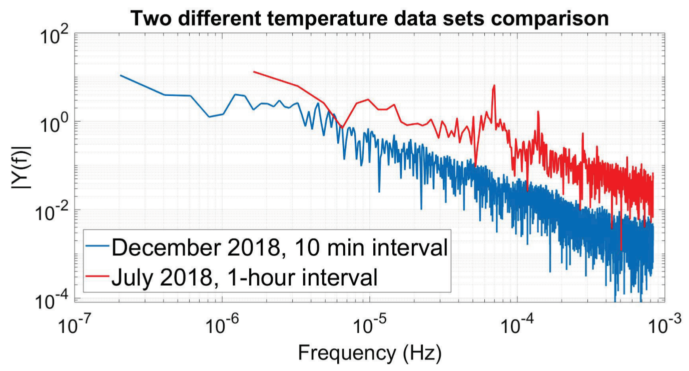


Fig. 2. The frequency spectrum of two temperature data sets.

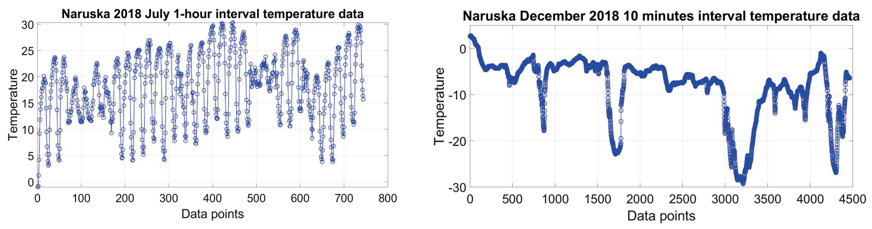


Fig. 3. The data sets with the highest *AC* and lowest *AC*.

The results presented here can be used to evaluate the suitability of these compression algorithms to compress certain data sets if the data characteristics are known. The choice between three different versions of the LRbTC can be made by evaluating the compression efficiency and computational complexity. In real measurement applications the future data set's characteristics are not known but the history data can be used to predict the probable data behavior and thus to choose the suitable algorithm.

4.2 Air Pressure Data Sets

The similar observations as for temperature data sets were done for air pressure data. The results can be seen in Fig. 4 where the left side illustrates the compression ratios of different compression algorithms in function of the *SD*, as described in this paper.

It can be seen in Fig. 4 on the left side that some data sets are dispersed slightly far from the other points. Those data sets are October 2018 data sets with all measurement intervals used. There is a clear error in the data because during October 2018 in air pressure data there is two times over 10 hPa air pressure change in 10 minutes. The air pressure change of more than 5 hPa/hour is rare and occurs only if there is an incoming thunderstorm [18]. Those two big sudden and atypical changes in air pressure rise the *SD* value relatively much; yet, it is not seen in *AC* value. For example, in October 2018 the air pressure data with 10-minute measurement interval have 4 064 measured values. Thus, those two big changes in measured values do not affect the *AC* value much. Thus, this behavior cannot be seen in Fig. 4 on the right side which is the compression ratio in function of *AC*.

The trend lines in Fig. 4 are fitting lines which are in this case 4th degree polynomials (y = p1*x^4 + p2*x^3 + p3*x^2 + p4*x + p5). The coefficients of fitting lines and the norm of residuals can be seen in Table 3 and Table 4.

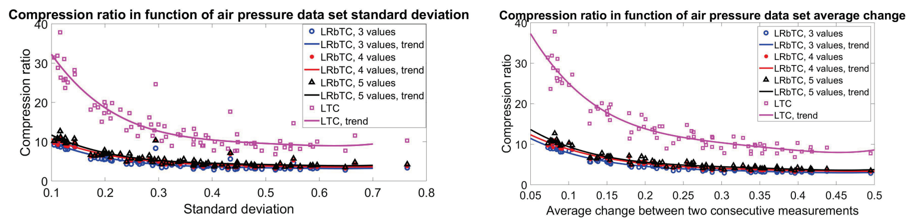


Fig. 4. Compression ratio in function of the *SD* (on the left side) and the *AC* (on the right side) for the air pressure data.

Table 3. Correlation between the compression ratio and standard deviation for air pressure data sets, polynomial coefficients and norm of residuals.

Coefficients	LRbTC, 3 values	LRbTC, 4 values	LRbTC, 5 values	LTC
p1	168.35	148.3	215.54	620.34
p2	-351.23	-316.4	-439.59	-1272.5
p3	277.08	257.58	338.11	990.82
p4	-99.784	-96.847	-119	-353.54
p5	17.371	18.119	20.632	58.903
Norm of residuals	6.0527	7.6021	7.9608	23.117

Table 4. Correlation between the compression ratio and average change for air pressure data sets, polynomial coefficients and norm of residuals.

Coefficients	LRbTC, 3 values	LRbTC, 4 values	LRbTC, 5 values	LTC
p1	743.39	555.94	961.93	2788
p2	-1051.1	-837.53	-1332.6	-3821.2
p3	561.64	482.03	694.26	1982.9
p4	-139.09	-129.9	-167.6	-481.17
p5	17.099	17.713	20.423	56.847
Norm of residuals	3.4408	4.2411	5.0525	18.369

The correlation in general is similar as with temperature data sets. The correlation is not linear but the general behavior can be seen in Fig. 4. The compression ratios are much higher for air pressure data than for temperature data, which indicates that air pressure data is rather linear in behavior and it is changing slowly. The compression algorithms based on data linearity are rather effective for this kind of data. The error bound used was 0.5 hPa which can be rather high in some applications. If the error bound is smaller, then the compression ratio is smaller.

4.3 Wind Speed Data Sets

Wind speed data from the Finnish Meteorological Institute's open data service is measured in a 10-minute average value [18]. In general, the wind speed has a slightly different behavior compared to the other environmental data. Wind speed can remain in 0 m/s for a while, and wind speed can also change quickly and there can be gusts. A 10-minute average measurement evens out the quick variation; however, the wind speed value remains in 0 m/s sometimes for long periods.

The results can be seen in Fig. 5. The correlation is again quite clear, and it is not as non-linear as with temperature and air pressure data sets.

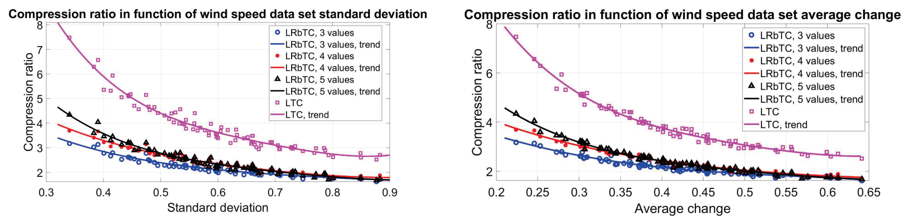


Fig. 5. Compression ratio in function of the *SD* (on the left side) and the *AC* (on the right side) for wind speed data.

The trend lines in Fig. 5 are 4th degree polynomials (y = p1*x^4 + p2*x^3 + p3*x^2 + p4*x + p5). The polynomial coefficients and the trend line norm of residuals can be seen in Table 5 and Table 6.

Table 5. Correlation between the compression ratio and standard deviation for wind speed data sets, polynomial coefficients and norm of residuals.

Coefficients	LRbTC, 3 values	LRbTC, 4 values	LRbTC, 5 values	LTC
p1	11.793	22.697	43.109	127.38
p2	-38.01	-61.805	-126.05	-352.64
p3	47.695	67.138	141.3	372.1
p4	-28.642	-36.706	-74.163	-181.75
p5	8.804	10.612	17.597	38.372
Norm of residuals	0.82744	0.83827	0.92181	1.7419

Table 6. Correlation between the compression ratio and average change for wind speed data sets, polynomial coefficients and norm of residuals.

Coefficients	LRbTC, 3 values	LRbTC, 4 values	LRbTC, 5 values	LTC
p1	-13.996	23.544	113.51	274.47
p2	2.7715	-54.641	-240.49	-561.96
p3	21.602	54.168	196.51	443.05
p4	-18.535	-27.962	-76.098	-164.2
p5	6.2976	7.8401	13.9	27.494
Norm of residuals	0.55859	0.5111	0.63325	1.1643

5 Conclusions

According to the research presented in this paper, it is possible to predict the compression ratio for selected compression methods according to average change (*AC*) and standard deviation (*SD*) values of the data. The correlation is better between the compression ratio and *AC* than compression ratio and *SD* in every type of environmental data tested. This can be seen by comparing the norm of residuals value between *AC* and *SD* results. The *AC* value is also very easy to calculate from the history data. The history data can be used to predict the compression ratio with the selected compression method. The results can be used to choose a suitable compression method and with the estimated compression ratio it is possible to predict the battery powered wireless sensor node lifetime.

The best correlation is with wind speed data sets and the worst with air pressure data set. At the same time, the compression methods selected and tested are most efficient for the air pressure data, and the least efficient for the wind speed data, whereas the temperature data is between these.

References

1. Poornima., Ayyanagowadar, M.S.: Internet of things in agriculture: A review. In: Agricultural Reviews, Volume 39, issue 4. (2018) 338-340. doi: 10.18805/ag.R-1836
2. Salam, A., Shah, S.: Internet of Things in Smart Agriculture: Enabling Technologies. In: 2019 IEEE 5th World Forum on Internet of Things (WF-IoT), Limerick, Ireland, (2019) 692-695. doi: 10.1109/WF-IoT.2019.8767306
3. Reddy, S.S., Azharuddin, M.R., Khan, K.: Importance of Internet of Things in Agriculture. In: International Journal of Recent Trends in Engineering Research. Vol.4(4), (2018) 372-373
4. Tzounis, A., Katsoulas, N., Bartzanas, T., Kittas, C.: Internet of Things in agriculture, recent advances and future challenges. In: Biosystems engineering 164, Elsevier. (2017) 31-48. doi: 10.1016/j.biosystemseng.2017.09.007

5. Abbasi, M., Yaghmaee, M.H., Rahnama, F.: Internet of Things in agriculture: A survey. In: 2019 3rd International Conference on Internet of Things and Applications (IoT), Isfahan, Iran. (2019) 1-12. doi: 10.1109/IICITA.2019.8808839
6. Väänänen, O., Hämäläinen, T.: Requirements for Energy Efficient Edge Computing: A Survey. In: The 18th International Conference on Next Generation Wired/Wireless Advanced Networks and Systems NEW2AN 2018, St.Petersburg, Russia. (2018) doi: 10.1007/978-3-030-01168-0_1
7. Sarbishei, O.: Refined Lightweight Temporal Compression for Energy-Efficient Sensor Data Streaming. In: 2019 IEEE 5th World Forum on Internet of Things (WF-IoT), Limerick, Ireland. (2019) 550-553. doi: 10.1109/WF-IoT.2019.8767351
8. Luo, G., *et al.*: Piecewise linear approximation of streaming time series data with max-error guarantees, 2015 IEEE 31st International Conference on Data Engineering, Seoul. (2015) 173-184. doi: 10.1109/ICDE.2015.7113282
9. Grützmacher, F., Beichler, B., Hein, A., Kirste, T. and Haubelt, C.: Time and Memory Efficient Online Piecewise Linear Approximation of Sensor Signals. Sensors, 18(6), (2018) 1672. doi: 10.3390/s18061672
10. Li, J., Li, G. and Gao, H.: Novel ε-Approximation to Data Streams in Sensor Networks. In IEEE Transactions on Parallel and Distributed Systems, vol. 26, no. 6, pp. 1654-1667, 1 June 2015. doi: 10.1109/TPDS.2014.2323056
11. Väänänen, O., Hämäläinen, T.: Compression methods for microclimate data based on linear approximation of sensor data. In: NEW2AN 2019: Internet of Things, Smart Spaces, and Next Generation Networks and Systems: Proceedings of the 19th International Conference on Next Generation Wired/Wireless Networking, and 12th Conference on Internet of Things and Smart Spaces, Lecture Notes in Computer Science, 11660. Cham: Springer, (2019) 28-40. doi: 10.1007/978-3-030-30859-9_3
12. Hung, N. Q. V., Jeung, H. and Aberer, K.: An Evaluation of Model-Based Approaches to Sensor Data Compression. In: IEEE Transactions on Knowledge and Data Engineering, vol. 25, no. 11, pp. 2434-2447, Nov. 2013. doi: 10.1109/TKDE.2012.237
13. Giorgi, G.: A Combined Approach for Real-Time Data Compression in Wireless Body Sensor Networks. In: IEEE Sensors Journal, vol. 17, no. 18, pp. 6129-6135, 15 Sept.15, 2017.
14. Wee, C. K., and Nayak, R.: Alternate approach to Time Series reduction. In: 2018 International Conference on Soft-computing and Network Security (ICSNS), Coimbatore. (2018) 1-4. doi: 10.1109/ICSNS.2018.8573685
15. Belov, A. A. and Proskuryakov, A. Y. Time Series Compression in Telecommunication Systems for Environmental Monitoring of Polluting Emissions. In: 2018 XIV International Scientific-Technical Conference on Actual Problems of Electronics Instrument Engineering (APEIE), Novosibirsk. (2018) 391-395. doi: 10.1109/APEIE.2018.8545336
16. Schoellhammer, T., Osterwein, E., Greenstein, B., et al.: Lightweight temporal compression of microclimate datasets. In: Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks IEEE Computer Society, pp. 516-524. (2004)
17. Aggarwal, Charu C.: Managing and Mining Sensor Data. Springer. 2013. doi: 10.1007/978-1-4614-6309-2
18. Finnish Meteorological Institute's open data–service. https://en.ilmatieteenlaitos.fi/opendata

V

SENSOR DATA STREAM ON-LINE COMPRESSION WITH LINEARITY-BASED METHODS

by

Olli Väänänen & Timo Hämäläinen, 2020

Proceedings of the 2020 IEEE International Conference on Smart Computing (SMARTCOMP)

https://doi.org/10.1109/SMARTCOMP50058.2020.00049

Sensor Data Stream On-line Compression with Linearity-based Methods

Olli Väänänen
School of Technology
JAMK University of Applied Sciences
Jyväskylä, Finland
0000-0002-7211-7668

Timo Hämäläinen
Faculty of Information Technology
University of Jyväskylä
Jyväskylä, Finland
0000-0002-4168-9102

***Abstract*—The escalation of the Internet of Things applications has put on display the different sensor data processing methods. The sensor data compression is one of the fundamental methods to reduce the amount of data needed to transmit from the sensor node which is often battery powered and operates wirelessly. Reducing the amount of data in wireless transmission is an effective way to reduce overall energy consumption in wireless sensor nodes. The methods presented and tested are suitable for constrained sensor nodes with limited computational power and limited energy resources. The methods presented are compared with each other using compression ratio and inherent latency. Latency is an important parameter in on-line applications. The improved variation of the linear regression-based method called RT-LRbTC is tested and it has proved to be a potential method to be used in a wireless sensor node with a fixed and predictable latency. The compression efficiency of the compression algorithms is tested with real measurement data sets.**

Keywords—edge computing, internet of things, sensor data, compression algorithm

I. Introduction

Simple linearity-based compression methods are not a new research topic; however, they have gained a great amount of attention recently due to the growing interest in the Internet of Things and wireless sensor networks. That kind of compression methods have been available already for decades. Most of these methods are based on analyzing the data stream retrospectively when all or at least a significant amount of the data need to be already available [1]. Thus, these methods are not well suitable for compressing the data stream in real-time or even near real-time.

Applications using some sensor data in control could benefit from effective real-time compression methods based on data linearity; in particular if the measured magnitude were some environmental magnitude which behaves quite linearly in short time window. These kinds of rather slowly changing and thus linearly behaving magnitudes are for example temperature, air pressure, humidity and wind speed. These kinds of measurements are typical in agricultural applications and in many other different IoT applications [2].

Using some simple and computationally light compression method can be a very effective way to save in energy consumption and thus lengthen the lifetime of battery powered sensor nodes which often operate wirelessly [3].

II. Linearity Based Compression Algorithms and Their Suitability for Real-Time Operations

As mentioned in the Introduction, many linearity-based compression methods analyzes the data retrospectively when the data is already available. It is easy to test and find the best possible compression algorithm if the data set is already available. This kind of approach is useful and suitable in Periodic Sensor Networks (PSN) [4]. In PSNs the node sends the data periodically to the sink [5]. This kind of measurement network does not work in real-time; however, the latency is known and can be adjusted by adjusting the sending period (frequency). There are many methods and protocols for PSNs to achieve longer battery lifetime by reducing the energy consumption with data aggregation and the amount of data needed to transmit wirelessly. Some methods are very simple based on constant approximation and some methods are slightly more complex [4]. Very simple methods suitable for constrained sensor nodes in PSNs are for example Piecewise Constant Approximation (PCA), Adaptive Piecewise Constant Approximation (APCA), Poor man's compression and Piecewise Linear Histogram (PWLH) [6][7][8][9][10][11]. These methods are so called model-based methods [1].

PCA is a very simple on-line algorithm which divides the data stream to constant linear segments. It guarantees that the compressed data satisfies the error bound (maximum allowed deviation between original data and linear model) requirements compared to the original raw data [11]. PCA divides the data set in to fixed lengths linear segments called as windows. PCA method first takes the number of window size of sensor signals and calculates the difference between maximum value and minimum value. If the difference is smaller than the error bound accepted, then all the data points in that segment (window) are represented with a constant value which is the middle point of the maximum value and minimum value. This is not the most effective method for compression, however, it is a very simple and computationally light method. It also has a fixed latency which is set by the window length [1][11].

APCA's functionality is very close to PCA. It varies from PCA thus that constant value segments vary in length. The length of the constant value segment is as long as it still meets the demands of the error bound. As a result of APCA's compression, there are constant segments of varying length. Each segment is represented by two values, the median value of the data points and the end time stamp of the segment [1][7]. If this model is applied to the sensor node, then the sensor node transmits two values after each segment to the sink (receiver). Because the segments vary in length, the latency is not known in advance, and the latency also varies depending on the length of the segments. The more stable the data values remain, the longer the segments are (higher compression ratio) which results in higher latency.

PWLH has similarities to APCA but the linear segments need not have constant values. Thus, the linear segments can be represented with lines the slope of which can be other than zero [1]. This method suffers also from the unknown length of the linear segments, and thus the latency cannot be anticipated.

These model-based methods are not well suited for the real-time operations with tight requirements for the latency. The benefits in these model-based methods are that they are simple and computationally very light. Thus, these methods are well suited for the battery powered computationally constrained devices.

There are also compression methods suitable to be used directly for the data stream. One very effective linearity-based compression method is called Lightweight Temporal Compression (LTC) [12]. It is a lossy method like all the other methods presented in this paper, and it is suitable to be implemented in constrained sensor node due to its computational simplicity. It is very effective and has a high compression ratio for the linearly behaving environmental data [2]. The significant drawback in this method is the latency; hence, it is not well suited for real-time applications [13]. The sensor node utilizing LTC algorithm sends the starting point of the linear segment to the receiver; however, the receiver does not know anything until the sensor node sends the end point of the linear section to the receiver. Between that there is no information available on the receiver side. The receiver does not know if the value is in average rising, staying at the same level or lowering, and after receiving the end point of the linear segment (which is at the same time the starting point of the next linear section), there is no information in which direction the values are changing after that.

There are also various linear regression-based algorithms available and presented in the field of the research. One method is called Piecewise Linear Approximation (PLA). It uses the linear regression to model data stream with a certain error bound allowed from the linear segment. Each linear segment is represented by the start and end time stamps and the line parameters (base and slope) or by the linear segment starting point (time stamp and value) and end point (time stamp and value) [10]. If the data set or a part of it is already available, it is possible to find the best amount of values to be used to calculate a regression line which determines the longest linear segment which still meets the error bound requirement to the data. This kind of approach is not well suited for real-time operations.

The linear regression can be calculated from the minimum of three data values; however, also more values can be used. This kind of approach is presented in [2] by the authors and the algorithm is called Linear Regression based Temporal Compression (LRbTC). In [2] 3, 4 and 5 values are used to calculate the regression line and have been tested to compress some environmental data (temperature, air pressure and wind speed). For near linearly behaving sensor data like temperature, there is a slight improvement in compression ratio if 4 or 5 values have been used to calculate the regression line. The disadvantage in LRbTC method is that the latency is not known in advance. The latency depends on how well the data is suited to the linear model. When the data behaves very linearly, it leads to a higher compression ratio but also higher latency at the same time. In this paper the authors present a modification for LRbTC method towards more real-time operation with known and fixed latency.

III. Linear Regression Based Compression Algorithms Towards Real-Time Operation

LRbTC as presented in [2] is a very simple compression algorithm. In basic form it is presented as a flow chart in Fig. 1. This method is based on linear regression of the N measured samples and the line calculated predicts future values allowing a certain error bound $\pm\varepsilon$ from the line. If the data is behaving linearly, the regression line gives quite a good prediction for the future values.

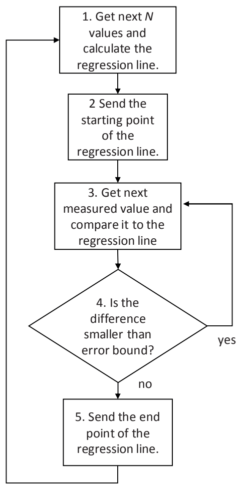

1. Get next N values and calculate the regression line.
2 Send the starting point of the regression line.
3. Get next measured value and compare it to the regression line
4. Is the difference smaller than error bound?
yes
no
5. Send the end point of the regression line.

Fig. 1. LRbTC algorithm.

As mentioned, this kind of algorithm in this form does not present a constant latency. Step 5 in Fig. 1. happens when the value is out of the regression line more than an error bound. When that happens depends on the measured values and cannot be predicted.

Model parameters: The raw data of sensor signal can be presented as $S = \langle(v_1,t_1), (v_2,t_2),\ldots (v_n,t_n)\rangle$, where v_i ($i \in \mathbb{N}$) is the measured value and t_i is the time stamp (moment). From the compression algorithm the compressed data stream consists of the starting points and the end points of the linear segments (c_i, τ_i), where c_i is the compressed value (start or end point of the linear regression line) and τ_i is the time stamp for that value. Thus, the output is $LRbTC(S) = ‹(c_1,\tau_1), (c_2,\tau_2),\ldots (c_n,\tau_n)›$.

One weakness in this basic version of LRbTC is in the first step where *N* values are used to calculate the regression line. The values used to calculate the new regression line can be more than an error bound away from the calculated line. In [2] the modified version M-LRbTC has been introduced, and its functionality can be seen in Fig. 2.

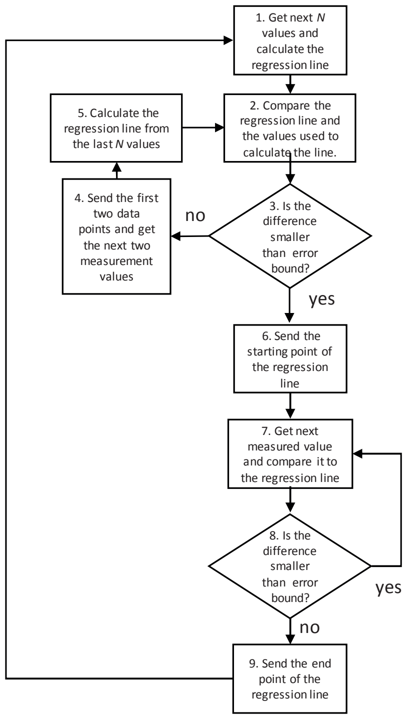

1. Get next N values and calculate the regression line
2. Compare the regression line and the values used to calculate the line.
5. Calculate the regression line from the last N values
4. Send the first two data points and get the next two measurement values
no
3. Is the difference smaller than error bound?
yes
6. Send the starting point of the regression line
7. Get next measured value and compare it to the regression line
8. Is the difference smaller than error bound?
yes
no
9. Send the end point of the regression line

Fig. 2. Modified LRbTC (M-LRbTC).

In M-LRbTC version, after calculating the regression line, the distance of the values used to calculate the regression line are compared to the line. If the difference is bigger than the error bound, then the first two data points $‹(v_1,t_1),(v_2,t_2)›$ are stored and/or sent to the sink, and the next *N* values are used to calculate the new regression line. This version has the same limitations as the original version for the real-time operations due to unknown latency.

One improvement for this kind of linear regression-based compression algorithm would be to send the regression line parameters (slope *a* and base *b*, as the line formula is $c = at + b$, where *c* is the value achieved from the linear model at the given time stamp *t*) with the starting point time stamp of the line to the sink (receiver). Thus, the latency would be the time of achieving *N* samples (*N*-1 sampling steps Δt), and the receiving part would know that the values follow the known line as long as the end point of the line is received. Thus, if the *N* is 3, then the latency is two times the measurement interval (2 x Δt) when the new regression line is calculated. The latency in the linear section (values following the regression line) is one measurement interval Δt. When the measurement value goes off the segment (more than error bound), then the last point of the line (which was one interval Δt before) is sent to the sink. This is a significant improvement compared to the most model-based piecewise approximation methods presented before and also compared to the LTC method. As a result from the compression the data is: $‹(a_1,b_1,\tau_1),(c_2,\tau_2),(a_3,b_3,\tau_3),(c_4,\tau_4),\ldots (a_{n-1},b_{n-1},\tau_{n-1}), (c_n,\tau_n)›$.

A. Towards real-time operations with predicted and constant latency

This LRbTC (M-LRbTC) method can be developed further to achieve an even shorter latency. When the measured value falls off from the allowed area (line with error bound), then the already measured values can be used to calculate the new regression line. Then the latency is only one measurement interval long (Δt). Only at the beginning of the measurement, when the first regression line is calculated, the latency is *N* - 1 intervals long. Simplified flow-chart of this kind of version is presented in Fig. 3. It is named here as Real-Time LRbTC (RT-LRbTC).

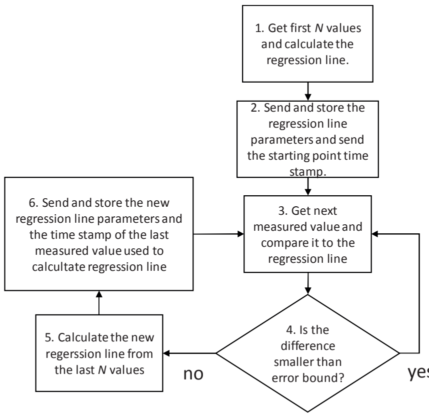

1. Get first N values and calculate the regression line.
2. Send and store the regression line parameters and send the starting point time stamp.
6. Send and store the new regression line parameters and the time stamp of the last measured value used to calcultate regression line
3. Get next measured value and compare it to the regression line
5. Calculate the new regerssion line from the last N values
no
4. Is the difference smaller than error bound?
yes

Fig. 3. Real-time LRbTC (RT-LRbTC).

The raw data of sensor signal is $S = ‹(v_1,t_1),(v_2,t_2),\ldots (v_n,t_n)›$. At the beginning of the algorithm the first *N* (*N* = 3, for example) value pairs are used to calculate the regression line. Thus, the values $‹(v_1,t_1),(v_2,t_2),(v_3,t_3)›$ are used to calculate the regression line ($c_1 = a_1t + b_1$) parameters a_1 and b_1. Three values are sent to the sink (a_1,b_1,τ_1) at time moment t_3 (plus the latency from the computational time), where $\tau_1 = t_1$. Thus, the algorithm latency at the beginning is $t_3 - t_1 = 2 \text{ x } \Delta t$. After that, the algorithm compares the following measured values to the regression line at the time of the value (step 3 in Fig. 3). When the measured value falls out more than the error bound from the regression line, then the new regression line (a_2,b_2,τ_2) is calculated from the last *N* values and sent to the sink. τ_2 is the time stamp of the value which fell out from the linear section. The receiving side knows that the previous regression line ended one measurement interval before ($\tau_2 - \Delta t$), thus the latency from the algorithm itself is one measurement interval Δt.

This basic version of RT-LRbTC has the same drawback as the original LRbTC when values used to calculate regression line can be more than an error bound away from the regression line. The version which corrects this problem is presented in Fig. 4.

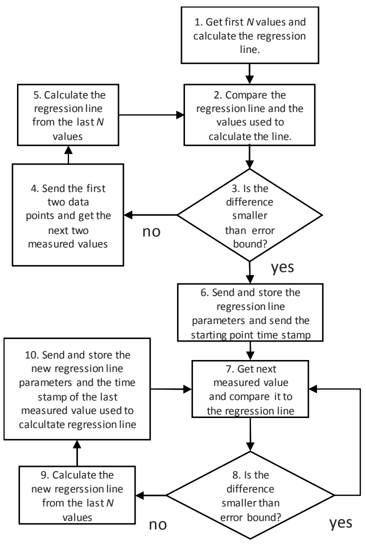

1. Get first N values and calculate the regression line.
2. Compare the regression line and the values used to calculate the line.
3. Is the difference smaller than error bound?
no
yes
4. Send the first two data points and get the next two measured values
5. Calculate the regression line from the last N values
6. Send and store the regression line parameters and send the starting point time stamp
7. Get next measured value and compare it to the regression line
8. Is the difference smaller than error bound?
no
yes
9. Calculate the new regerssion line from the last N values
10. Send and store the new regression line parameters and the time stamp of the last measured value used to calcultate regression line

Fig. 4. Improved RT-LRbTC.

Fig. 4. shows in step 2 the comparison between the regression line and the values used to calculate that regression line. If the distance from the line is more than the error bound, then first two values (‹(v_1,t_1),(v_2,t_2)›) are stored/sent and the new line is calculated from the following values ‹(v_3,t_3),(v_4,t_4),(v_5,t_5)›. If and when the difference is at an accepted level, then in step 6 the regression line parameters and starting time stamp of the line (a_1,b_1,τ_1) are stored and sent to the receiver. In that point the latency is $N-1$ time steps (measurement interval) long as in the basic version of RT-LRbTC. In step 7 the next measured values are compared to the line and if the difference is less or equal to the error bound (step 8), the comparison continues with the next value. As long as this continues, there is no need to send anything to the sink. In the sink the receiver knows that the values measured one time step (measurement interval, Δt) before are within error bound from the regression line as long as no new line is received.

There is one measurement interval time latency because when the measured value falls out from the linear section, then the new line is calculated using the last N values and the new line starts. The values sent to the sink are (a_2,b_2,τ_2). The previous line is ended one time step before ($\tau_2 - \Delta t$); however, the information of that is achieved only when the next value falls out from the line more than error bound $\pm\varepsilon$. Thus, in this method only one sending period is needed for each linear section and thus the amount of the sending periods is only half compared to most other linear regression based methods and LTC method. In basic form of linear regression-based methods and also in LTC method, there is a needed to send starting point value with time stamp and end point value with time stamp for each linear section.

In Fig. 5. the comparison of M-LRbTC and RT-LRbTC shows the difference between these algorithms. Both algorithms use N = 3 values to calculate (time stamps 1,2 and 3) the regression line at time stamp 3, thus at the beginning both algorithms get the same line (a_1,b_1). M-LRbTC sends the regression line starting value (line value c_1 at time $\tau_1 = t_1$) to the sink at time stamp 3. RT-LRbTC sends the line parameters with the starting point of the line (a_1,b_1,τ_1), where the $\tau_1 = t_1$ to the sink at time stamp 3. At time stamp 11 the measured value falls out from the regression line. Thus, M-LRbTC sends the regression line end value (c_2, τ_2), where $\tau_2 = t_{10}$, and calculates the new regression line from the measured values at time stamps 11, 12 and 13. M-LRbTC sends the new regression line starting value c_3 and line starting time stamp $\tau_3 = t_{11}$ at time moment 13 when the new line is calculated. At time stamp 11, RT-LRbTC calculates the new regression line from the values at time stamps 9, 10 and 11. When the receiver gets the new line parameters (a_2,b_2,τ_2), where $\tau_2 = t_{11}$ it knows that the previous line ended at time $\tau_2 - \Delta t = t_{10}$.

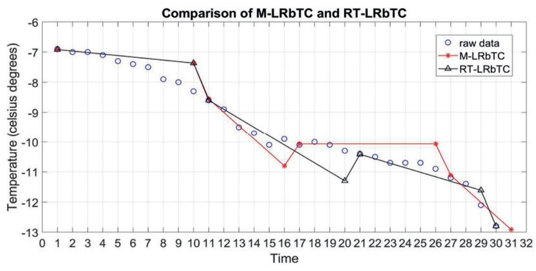


Fig. 5. Comparison of M-LRbTC and RT-LRbTC.

B. *Compression efficiency of RT-LRbTC to compress environmental data*

RT-LRbTC algorithm's compression efficiency was tested with the same data sets as the authors have used in [2] and with similar newer data sets. The Naruska measurement station data sets from 2018 and 2019 were achieved from the Finnish Meterological Institute's open data service [14]. The data sets used were temperature, air pressure and wind speed from the whole years 2018 and 2019 with a 10-minute measurement interval. For each magnitude there were 51,961 values in year 2018 data set and 52,463 values in year 2019 data set. The compression algorithm's ability to compress those data sets was tested with different error bounds from 0.1 to 2.0. RT-LRbTC method was compared to the original M-LRbTC method, which is presented and tested in [2], and LTC method which has been the best algorithm in [2] when comparing the compression ratios. The algorithms have been programmed in MATLAB. M-LRbTC, and LTC algorithms are exactly the same algorithms as in [2]. RT-LRbTC is a modification of M-LRbTC algorithm. M-LRbTC and RT-LRbTC used three values to calculate the regression line.

M-LRbTC method sends the starting point and ending point of each linear regression line segment. In RT-LRbTC only the line parameters and the time stamp for the line starting point are sent, thus the transmitting periods needed are reduced to half compared to the original method. In M-LRbTC method the two values (value and time) are sent twice for each linear segment compared to three values (line parameters a and b and time) needed to send once for each linear section in RT-LRbTC.

The compression ratio (CR) is calculated by dividing the amount of original data by the amount of compressed data.

The results for the temperature data can be seen in Fig. 6. The results are very similar for both data sets (2018 and 2019). The LTC is superior compared to the other two algorithms. RT-LRbTC benefits from the fact that there is only needed to send parameters once for each regression line. Actually, there are more regression lines needed in RT-LRbTC and in that way it is less efficient compared to M-LRbTC.

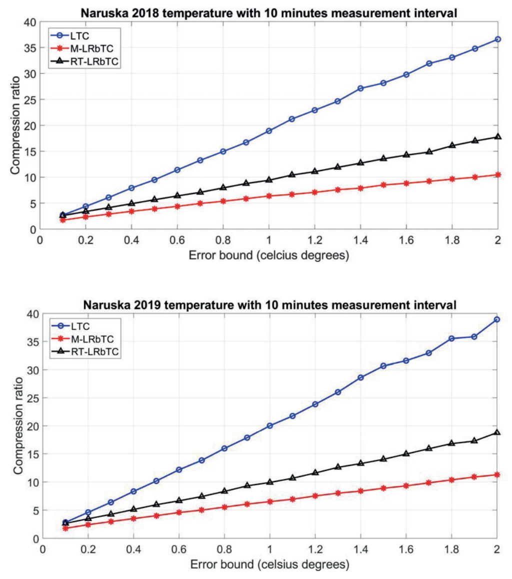


Fig. 6. Comparison on the algorithms with temperature data.

Similar comparison in compression ratios was done with air pressure data sets. The results for 2018 and 2019 data sets can be seen in Fig. 7.

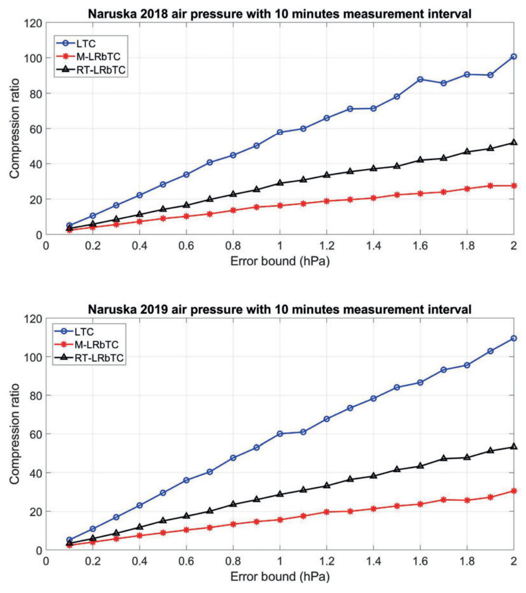


Fig. 7. Comparison of the algorithms with air pressure data.

LTC algorithm is again superior compared to the other two and the compression ratios are generally remarkably higher than with temperature data. This is an indication that air pressure data in general is changing quite slowly and behaves quasi linearly. The error bounds with temperature data are not fully comparable because temperature is in Celsius degrees and air pressure in hectopascals (hPa).

Fig. 8. Illustrates the comparison between algorithms to compress the wind speed data. The wind speed data is measured with 10-minute average value [14].

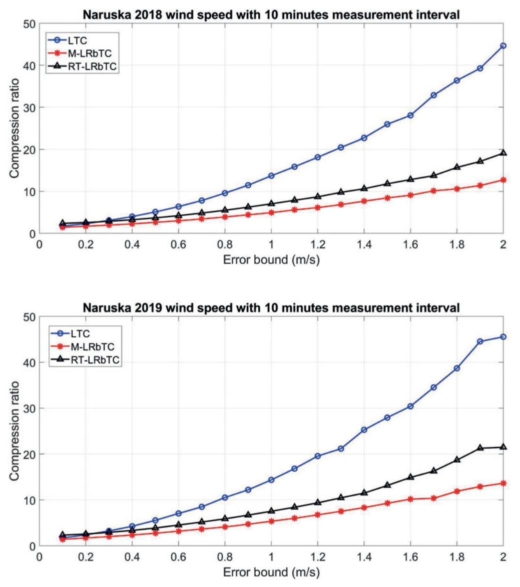


Fig. 8. Comparison of the algorithms with wind speed data.

Even with 10-minute average measurement the wind speed remains rather long periods in zero; however, on the other hand, it is also changing rapidly in other moments. Thus, it is not behaving that linearly and changing as slowly as the temperature and air pressure. The results in compression ratios are quite close compared to the temperature data. That is because of the rather long periods with consecutive measurements with zero value for wind speed.

IV. Results

The results of compression ratios comparison for LTC and M-LRbTC are similar as in [2] also for 2019 data. RT-LRbTC suffers from the amount of the regression line calculations but benefits more from the fact that only one transmitting period is needed for each regression line compared with the two transmitting periods with M-LRbTC.

TABLE I presents the results as compression ratios when the error bound is 0.5 °C for temperature, 0.5 hPa for air pressure data and 0.5 m/s for wind speed data. These are realistic error bounds which could be used in real application. It can be seen that the compression ratios are similar for both data sets (2018 and 2019) for each algorithm and each magnitude in comparison. Anyway, the compression ratios are slightly higher for 2019 data except M-LRbTC for air pressure data. The average change is the absolute average change between two consecutive values in the whole data set for given magnitude. For temperature data and wind speed data the average change is slightly smaller for 2019 data and it can indicate that the data is behaving slightly more linearly and thus resulting in a better compression ratio.

TABLE I. COMPARISON OF COMPRESSION RATIOS

Data set	Average change	Compression Algorithms' Compression Ratios		
		LTC	*M-LRbTC*	*RT-LRbTC*
Temperature 2018	0.223	9.49	3.90	5.65
Temperature 2019	0.208	10.17	4.02	5.96
Air pressure 2018	0,086	28.22	8.94	14.09
Air pressure 2019	0,086	29.56	8.84	14.99
Wind speed 2018	0,302	5.09	2.62	3.68
Wind speed 2019	0,284	5.54	2.74	3.87

TABLE II presents the comparison of the latencies in different phases of the algorithm operation. Only the algorithm's inherent latency is taken into account, not the latency from the computational delay or from the data transmission. Only M-LRbTC (2) and RT-LRbTC algorithms can present a predictable latency. RT-LRbTC presents the shortest latency in operation and it is dependent on the measurement interval.

TABLE II. COMPARISON OF LATENCIES

Latency	Compression Algorithm			
	LTC	*M-LRbTC (1)*	*M-LRbTC (2)*	*RT-LRbTC*
At the beginning	0	$(N\text{-}1) \times \Delta t$	$(N\text{-}1) \times \Delta t$	$(N\text{-}1) \times \Delta t$
In linear section	length of the linear section	length of the linear section	Δt	Δt
Calculating new line	not applicable	$N \times \Delta t$	$N \times \Delta t$	Δt

M-LRbTC (1): The linear regression line start point and end point values are sent.

M-LRbTC (2): The linear regression line parameters are sent with the starting time stamp.

V. CONCLUSIONS

Different versions of linearity-based sensor data compression algorithms were presented and tested in this paper. The main focus was on compression ratio and the inherent latency from the algorithm itself. Many linearity-based compression algorithms presented in the field of research are model based methods demanding a set of data already available to be implemented. Those methods are not well suited for analyzing the sensor data stream in on-line mode if there are requirements for the latency.

The presented and tested methods can be used in on-line mode for the sensor data stream; however, only the new variation RT-LRbTC can represent rather short and fixed latency. Its general compression efficiency is rather low with the tested data sets, but it benefits from the fact that only one transmitting period is needed for each linear segment. The wireless transmission is known to be the most energy consuming operation in wireless sensor nodes. The linearity-based methods presented benefits from the fact that environmental magnitudes behave rather linearly in a short time window.

The next step will be to implement these linearity-based methods in an embedded edge device such as a wireless sensor node and test the methods in on-line mode for the data stream. The actual effect on energy consumption will be tested and measured and the computational complexity of different methods will be taken into account and analyzed in detail.

REFERENCES

[1] N. Q. V. Hung, H. Jeung and K. Aberer, "An Evaluation of Model-Based Approaches to Sensor Data Compression," in IEEE Transactions on Knowledge and Data Engineering, vol. 25, no. 11, pp. 2434-2447, Nov. 2013. doi: 10.1109/TKDE.2012.237

[2] O. Väänänen and T. Hämäläinen, "Compression methods for microclimate data based on linear approximation of sensor data," in NEW2AN 2019: Internet of Things, Smart Spaces, and Next Generation Networks and Systems: Proceedings of the 19th International Conference on Next Generation Wired/Wireless Networking, and 12th Conference on Internet of Things and Smart Spaces, LNCS, 11660. Cham: Springer, 28-40. doi: 10.1007/978-3-030-30859-9_3

[3] O. Väänänen and T. Hämäläinen, "Requirements for energy efficient edge computing: a survey," in: Galinina, O., Andreev, S., Balandin, S., Koucheryavy, Y. (eds.) NEW2AN/ruSMART -2018. LNCS, vol. 11118, pp. 3–15. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-01168-0_1

[4] A. K. M. Al-Qurabat and A. K. Idrees, "Two level data aggregation protocol for prolonging lifetime of periodic sensor network," in Wireless Networks (2019) 25: 3623-3641. https://doi-org/10.1007/s11276-019-01957-0

[5] A. Makhoul, H. Harb and D. Laiymani, "Residual energy-based adaptive data collection approach for periodic sensor networks," in Ad Hoc Networks, Volume 35, 2015, Pages 149-160, ISSN 1570-8705, https://doi.org/10.1016/j.adhoc.2015.08.009.

[6] A. Mahbub, F. Haque, H. Bashar and M. R. Huq, "Improved Piecewise Constant Approximation Method for Compressing Data Streams," 2019 1st International Conference on Advances in Science, Engineering and Robotics Technology (ICASERT), Dhaka, Bangladesh, 2019, pp. 1-6. doi: 10.1109/ICASERT.2019.8934460

[7] E. Keogh, K. Chakrabarti, S. Mehrotra & M. Pazzani, "Locally adaptive dimensionality reduction for indexing large time series databases," in Sigmod Record, 30(2), 2001, pp. 151-162.

[8] C. Buragohain, N. Shrivastava and S. Suri, "Space Efficient Streaming Algorithms for the Maximum Error Histogram," 2007 IEEE 23rd International Conference on Data Engineering, Istanbul, 2007, pp. 1026-1035. doi: 10.1109/ICDE.2007.368961

[9] C. Wang, C. Yen, W. Yang and J. Wang, "Tree-Structured Linear Approximation for Data Compression over WSNs," 2016 International Conference on Distributed Computing in Sensor Systems (DCOSS), Washington, DC, 2016, pp. 43-51. doi: 10.1109/DCOSS.2016.37

[10] C. C. Aggarwal, Managing and Mining Sensor Data. Springer. 2013. Doi: 10.1007/978-1-4614-6309-2

[11] I. Lazaridis and S. Mehrotra: Capturing Sensor-Generated Time Series with Quality Guarantees. In: Proc. Int'l Conf. Data Eng. (ICDE), 2003, pp. 429-440.

[12] T. Schoellhammer, B. Greenstein, E. Osterweil, M. Wimbrow and D. Estrin, "Lightweight temporal compression of microclimate datasets [wireless sensor networks]," 29th Annual IEEE International Conference on Local Computer Networks, Tampa, FL, USA, 2004, pp. 516-524. doi: 10.1109/LCN.2004.72

[13] G. Giorgi, "A Combined Approach for Real-Time Data Compression in Wireless Body Sensor Networks," in IEEE Sensors Journal, vol. 17, no. 18, pp. 6129-6135, 15 Sept.15, 2017.

[14] Finnish Meteorological Institute's open data–service. https://en.ilmatieteenlaitos.fi/opendata

VI

LORA-BASED SENSOR NODE ENERGY CONSUMPTION WITH DATA COMPRESSION

by

Olli Väänänen & Timo Hämäläinen, 2021

Proceedings of the 2021 IEEE International Workshop on Metrology for Industry 4.0 & IoT (MetroInd4.0&IoT)

https://doi.org/10.1109/MetroInd4.0IoT51437.2021.9488434

LoRa-Based Sensor Node Energy Consumption with Data Compression

Olli Väänänen
School of Technology
JAMK University of Applied Sciences
Jyväskylä, Finland
0000-0002-7211-7668

Timo Hämäläinen
Faculty of Information Technology
University of Jyväskylä
Jyväskylä, Finland
0000-0002-4168-9102

***Abstract*—In this paper simple temporal compression algorithms' efficiency to reduce LoRa-based sensor node energy consumption has been evaluated and measured. It is known that radio transmission is the most energy consuming operation in a wireless sensor node. In this paper three lightweight compression algorithms are implemented in an embedded LoRa platform to compress sensor data in on-line mode and the overall energy consumption is measured. Energy consumption is compared to the situation without implementing any compression algorithm. The results show that a simple compression algorithm is an effective method to improve the battery powered sensor node lifetime. Despite the radio transmission's high energy consumption, the sleep current consumption is a significant factor for the device overall lifetime if the measurement interval is long.**

Keywords—Internet of Things, edge computing, sensor data, compression, energy efficiency

I. INTRODUCTION

It is known that the most significant single energy consumer of wireless sensor node is the radio transmitter. A well-known method to reduce wireless sensor node energy consumption is to compress the data and thus reduce the radio transmitting periods. In this paper, three simple temporal compression algorithms are implemented in a LoRa-based sensor node, and the overall energy consumption is measured. The effect of the compression algorithm in battery lifetime has been evaluated. In this paper it has been demonstrated that if the measurement interval is rather long, then the most effective method to lengthen the battery lifetime is to minimize the sensor node sleep current consumption between the measurement and transmission periods. Minimizing the sleep current also improves the effectivity of the powerful compression algorithm to lengthen battery lifetime. Thus, the combination of very low sleep current consumption together with light temporal compression algorithm has been demonstrated to be an effective method to lengthen the battery lifetime of the LoRa sensor node. This kind of application could be useful for example in agricultural applications for measuring some environmental magnitudes. In different smart farming applications typical measured magnitudes are for example temperature and humidity. These magnitudes behave rather linearly, and thus the measurement interval can be rather long. In agricultural applications the sensor nodes are often located in the fields, thus requiring wireless transmission and a battery powered supply. Suitable applications could be also different smart city applications. Temporal compression algorithms used in this research are very effective for linearly behaving data.

II. RELATED WORK

There are some research papers about the energy consumption of wireless sensor nodes. In [1], many myths related to energy models of wireless sensor networks have been busted. For example, it is generally known that the radio transmission (sending data packets) is the most energy consuming operation, but actually keeping the radio in idle mode listening is in total a more consuming operation. This is because the transmitting and receiving last a very short time compared to the time that the radio is listening for incoming packets. It is also demonstrated that generic energy models do not fit for all devices even if the devices use a similar technology [1]. Thus, measurements need to be made to gain reliable results for certain devices used. The energy consumption models and battery lifetime experiments of different wireless sensor node technologies are published for example in [2][3].

LoRa technology has been developed to be used in the Internet of Things applications. LoRa is a low-energy wide area network especially suitable for transmitting sensor data. There are several published research papers regarding energy consumption modelling for LoRa based sensor nodes, such as [4][5][6][7]. In [8] the energy consumption of different wireless sensor network technologies is compared. Papers have demonstrated that LoRa and SIGFOX offer the best lifetime for low intensity traffic. It is also demonstrated that the sleep power consumption is a significant factor for the device lifetime if the transmission period frequency is low [8].

In [6] the effect of LoRa parameters on the LoRa node energy consumption has been researched and measured. Higher spreading factor (SF) increases the time on air and thus increases the power consumption of transmission. But to achieve long range, high SF value needs to be used. In fact, the overall LoRa transmission energy consumption is dependent on different LoRa/LoRaWAN parameters such as spreading factor, coding rate, payload size, and bandwidth [6].

One well known method to reduce wireless sensor node energy consumption is to use data compression. Temporal lossy compression algorithms are often light and simple. If the wireless transmission periods could be reduced using data compression, it would reduce the overall energy consumption as the radio could be a longer time period on sleep mode.

For example, with LoRaWAN Class A the device is not listening to downlink messages except after uplink transmission for two short downlink windows. Thus, it can remain in sleep mode until the data needs to be transmitted [9].

Thus, there are many research papers dealing with LoRa device energy consumption and other papers dealing with compression algorithms efficiency to reduce energy consumption. But not many experiments have been published on how the simple compression algorithm can reduce LoRa-sensor node energy consumption and thus lengthen the battery lifetime. In this paper the effectiveness of simple temporal compression algorithms to reduce overall energy consumption of the LoRa sensor node is evaluated and measured.

III. Temporal Compression Methods for Sensor Data

Compression algorithms implemented and tested in this paper are simple temporal compression methods based on data linearity. Many environmental magnitudes behave rather linearly, and simple linearity-based algorithms utilize that behavior. The methods are very simple, easy to understand and easy to implement in an embedded platform. IoT wireless sensor nodes are often computationally constrained, and thus simple and light compression algorithms suit these devices well. Wireless sensor nodes are also often battery powered or even harvesting the energy from the environment. Thus, minimizing the energy consumption is important for lengthening the device lifetime.

The implemented algorithms are Lightweight Temporal Compression (LTC) and Real-Time Linear Regression based Temporal Compression (RT-LRbTC). LTC is originally presented in [10]. RT-LRbTC is a modification of other linear regression-based compression algorithms, and it is originally presented by the authors of this paper in [11]. These algorithms are lossy methods meaning that reconstructed data after compression is not exactly the same as the original data. The maximum reconstruction error is determined by the error bound (ε) used.

A. Lightweight Temporal Compression

Lightweight Temporal Compression (LTC) is a well-known temporal compression method. It was first presented in 2004 in [10], and several modifications on that algorithm have been presented since then. It has proved to be a very effective compression algorithm, especially for environmental magnitudes [12]. As a disadvantage, the LTC has unpredictable latency, and thus it is not well suited for real-time or near real-time applications [13]. In this paper the LTC is used as it is in its original version and transformed and implemented for embedded Arduino board from MATLAB version used in [11] and [12]. The LTC algorithm itself is computationally very light. With every new measured value, comparisons need to be made between the new value with error bound extremes to the previously calculated upper and lower limit lines. As a result of the comparisons in maximum two new lines are calculated. Calculating line parameters (slope and y-intercept) requires one division, one multiplication, one summation and subtractions. Thus, the compression algorithm is computationally very light.

B. Real-Time Linear Regression based Temporal Compression

Real-Time Linear Regression based Temporal Compression (RT-LRbTC) is also a very simple temporal compression algorithm. It is a modification from other linear regression-based algorithms, and it is developed especially for compressing the on-line sensor data stream. The algorithm's suitability for compressing on-line data stream is based on minimizing the algorithm's inherent latency. In general, this algorithm's inherent latency is one measurement interval (Δt) long. The basic form of the algorithm uses 3 previous data values ($N = 3$) to calculate the regression line and uses it to predict future values with certain error bound. The algorithm is explained in detail in [11]. For every new measured value its value needs to be compared to the previously calculated regression line value in that time stamp. If the difference between the new value and regression line is more than error bound, then a new regression line is calculated. Calculating the regression line requires the sum of N times x_k, x_k^2, y_k, $x_k y_k$, and the square of the sum of x_k. x_k is the time stamp and y_k is the measured value. In this paper also $N = 4$ version is tested to see if calculating regression line with 4 values has any effect on the complexity and energy consumption of the compression algorithm calculation. The implemented version is derived to Arduino from MATLAB version used in [11].

IV. Energy Consumption of Implemented Compression Methods with LoRa Connection

Embedded LoRa-based sensor node used in this experiment was Arduino MKR WAN 1310 board. It has a Microchip SAM D21 32-bit Arm Cortex-M0+ based low power microcontroller and Murata CMWX1ZZABZ LoRa module. It also has a crypto chip ATECC508 by Microchip [14]. Arduino boards are widely used by hobbyists but also in research and in the industry due to their simplicity and ease of use.

DHT22 sensor was used to measure temperature. DHT22 is a low-cost temperature and humidity sensor with a digital output [15]. It is widely used by hobbyists for its low cost, and there are many project examples available on the internet where DHT22 sensor is used. There are also support libraries available for most used embedded development boards. The sensor has measurement resolution 0.1 Celsius degrees for temperature, and its accuracy is <±0.5 Celsius degrees for temperature measurement [15].

Arduino MKR WAN 1310 was powered by Lithium-Polymer (Li-Po) battery. The Li-Po battery used had 2 000 mAh capacity with 3.7 V nominal voltage. Thus, its energy capacity was 7.4 Wh which is 26 640 Ws. The Arduino board supports Li-Po battery with JST-PH connector, and it has a built-in battery charger. Arduino board can also be powered from the USB port, but the energy consumption is lower if powered from the battery.

A. Measurement Setup

The compression algorithms implemented were set to measure the temperature with DHT22 sensor at regular intervals. The embedded board was set to go to deep sleep mode between the measurements. After every measurement the algorithm compressed the data, and the compressed data was stored and sent via LoRa network if needed. The LoRa network used was a commercial network in Finland.

The board's current consumption was measured with oscilloscope current probe from the battery plus-wire, except that the deep sleep current consumption was measured with a digital multimeter (DMM), which was set in series with the battery. The other possibility would be to use a so called shunt resistor in series with the battery and measure the voltage over the resistor with oscilloscope voltage probe. The current probe was chosen to be used because of its simplicity and ease of use even though its accuracy is not very good when measuring very low-level current values.

The measurement devices used were Tektronix MSO 4104 Mixed Signal Oscilloscope with 1 GHz measurement bandwidth, and the probes used were Tektronix TCP0030 Current Probe and TAP1500 Active Voltage Probe. The current probe has 1 mA sensitivity and 1 % DC accuracy [16]. The DMM used was Tenma RS232C TrueRMS model.

B. Deep Sleep Current Consumption

Deep Sleep current consumption was measured with the algorithms implemented on the board. The board was set to deep sleep mode between measurements. The DMM was set in series with battery wire. The measured deep sleep current was not dependent on the algorithm used, as the measurement result was the same with every algorithm tested and also without compression algorithm implemented. The measured deep sleep current was 116 µA – 117 µA. After the measurement instant and possibly LoRa-transmission, when the Arduino board went to deep sleep mode, the current dropped down immediately to 150 µA - 160 µA level, and continued going down quickly to 120 – 130 µA level and then more slowly to stabilize to 116 µA – 117 µA level. It took about 20 seconds to gain the 117 µA level. The battery voltage was measured at the same time with the oscilloscope's voltage probe connected to the board's battery connection. The battery voltage was 3.99 V while measuring the deep sleep current. Thus, the power consumption in deep sleep mode was: $P_{ds} = U_{batt} \text{ x } I_{ds} = 3.99 \text{ V x } 117 \cdot 10^{-6} \text{ A} = 4.6683 \cdot 10^{-4} \text{ W} = 0.46683$ mW.

The 117 µA current consumption in deep sleep mode is not very low for a modern microcontroller development board, and with some other boards it could be possible to achieve lower levels.

C. Sensor Measurement and Algorithm Current Consumption

On a regular basis the embedded board wakes up from the deep sleep mode and makes the measurement. The real-time clock (RTC) is running even when the board is in deep sleep mode, and it wakes up the board. After the wake up, the board measures the temperature from DHT22 sensor and applies the compression algorithm implemented. As a result of algorithm calculations, the board either sends the compressed value via LoRa-connection or falls into deep sleep mode to wait for the next measurement period.

The current/energy consumption during sensor measurement and compression algorithm period was measured with oscilloscope current probe, and oscilloscope automatic measurement functions were used to show the mean values. The battery voltage was measured at the same time with voltage probe, and the oscilloscope MATH-function was used to multiply the measured values to get the power value. The oscilloscope measurement function "area" was used to calculate the power graph area to get the energy consumption. In Fig. 1. one measurement result can be seen. The deep sleep energy consumption with the same oscilloscope settings was also measured to be subtracted from the measurement period result. Thus, only sensor measurement and algorithm calculation period consumption were achieved. The sensor measurement, data compression and board shut down to deep sleep takes about 60 ms time (the high pulse in Fig. 1.). After the measurement and algorithm calculations, the current consumption did not go directly to the deep sleep level but remained slightly higher for 390 ms due to DHT22 sensor.

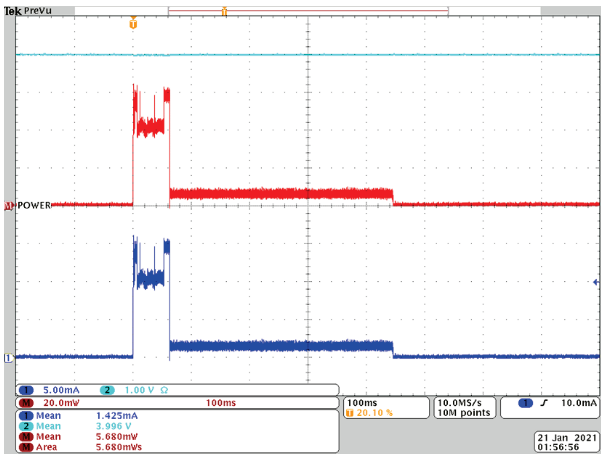


Fig. 1. RT-LRbTC: sensor measurement and algorithm period.

The oscilloscope measurement settings were: current probe 5 mA/div, voltage probe 1 V/div and horizontal scale 100 ms/div. Each measurement was repeated ten times, and the average values of the measurements were used for better measurement reliability. The measurement results can be seen in TABLE I. Each value in TABLE I is an average value of the ten separated measurements in one second measurement window (oscilloscope screen scale 100ms/div = total 1 second, as seen in Fig. 1.).

TABLE I. SENSOR MEASUREMENT AND ALGORITHM MEASUREMENT RESULTS

	No compression	**LTC**	**RT-LRbTC, *N*=3**	**RT-LRbTC, *N*=4**
Current (mA)	1.3977	1.4247	1.4016	1.4500
Battery voltage (V)	3.9949	3.9925	3.9906	3.9963
Power (mW)	5.5802	5.6836	5.6208	5.7814
Energy (mWs)	5.5775	5.6836	5.6208	5.7814
Deep sleep energy (mWs)	1.0003	0.9013	0.8428	0.9492
Measurement + algorithm (mWs)	4.5772	4.7823	4.7780	4.8322

The last line of TABLE I shows the results for the energy consumed for sensor measurement and algorithm calculations for each algorithm and also without any compression algorithm for comparison. The result is achieved by subtracting the deep sleep energy value (second last row) from energy value (third last row). It can be seen from the results that the values are on the same level for each algorithm and even without compression algorithm implemented, this means that the algorithm calculations do not affect at all the board's energy consumption, or the effect is so small that it is not possible to recognize it with this measurement setup. As can be seen from the results, there is no significant difference between the algorithms. The results between the algorithms and without an algorithm are within measurement uncertainty. The measured differences are negligible.

D. *LoRa Transmission Energy Consumption*

LoRa node used the spreading factor SF10 for transmitting, and the network sent the confirmation with SF9 or SF12. During the whole testing period the network confirmation was sent with SF9 for 49.2 % of all occasions and with SF12 for 50.8 % for all occasions. TABLE II shows the energy consumption measurement results when LoRa node is sending the data packets with SF10, and the network sends the confirmation with SF9. In TABLE III are the results when the data up is with SF10 and data down with SF12. Every measurement is repeated ten times, and the results are the average values of the measurements. In Fig. 2. one result for the SF10 uplink and SF9 downlink situation can be seen. It seems that the LoRa sensor node is not receiving the confirmation from the network even though the network has sent the confirmation. Two 10 mA pulses, the first approximately one second after the transmit, and the second one two seconds after transmit are the two receive windows that follow the uplink. Because the downlink is not received during the first window, then the second receive window opens two seconds after the transmit. In Fig. 3. there is the result of SF10 uplink, SF12 downlink situation. In this case it seems that the LoRa node has received the downlink confirmation because the receive window is rather long and not followed by the second receive window.

In TABLE II and TABLE III, the last row transmission energy is calculated by subtracting the deep sleep energy and previously measured sensor measurement and algorithm energy consumptions (TABLE I) from the measured transmission energy value (third last row in TABLE II and III). The deep sleep energy was measured with the same setup and measurement equipment settings while the LoRa-sensor node was in deep sleep mode. In Fig. 2. and Fig 3. the current measurement with the current probe is the blue line (probe 1), battery voltage is the light blue line (probe 2), and power is the red M-line which is voltage multiplied with current. The results in Fig. 3. include the sensor measurement and algorithm calculations, but Fig. 2. includes only the sensor measurement as the compression algorithm was not implemented in this case. The transmission period is approximately 400 ms long, and it is followed by the network confirmation time window one second after the transmission.

As can be seen in the transmission energy values (last row in TABLE II and TABLE III) there is not a big difference between the algorithms and taking into account the measurement uncertainty of low current values measured with current probe, the differences are negligible. The results with no compression algorithm implemented are even higher than with LTC algorithm. The results of LTC algorithm compression are the temperature value (float number) and time stamp (which is only a sequence number, integer), thus the transmitted data is 8 bytes. With RT-LRbTC algorithms the regression line parameters (slope and base, both float numbers) with time stamp (sequence number, integer) are sent in total 12 bytes. Thus, the amount of data sent is bigger with RT-LRbTC algorithms, and thus this could explain a slightly higher energy consumption in the transmission. In general, these results can be regarded to be approximately the same for each algorithm. The differences are negligible and can be explained by measurement uncertainty.

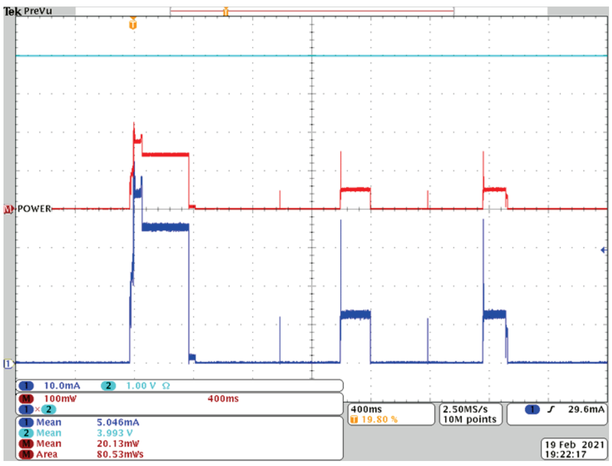
TABLE II. UPLINK SF10, DOWNLINK SF9

	No Compression	**LTC**	**RT-LRbTC, *N*=3**	**RT-LRbTC, *N*=4**
Current (mA)	4.9736	4.7976	5.1049	5.2293
Battery voltage (V)	3.9928	3.9924	3.9887	3.9897
Power (mW)	19.842	19.143	20.323	20.652
Energy (mWs)	79.372	76.565	81.289	82.605
Deep sleep energy (mWs)	3.5636	4.0993	4.971	4.632
Transmitting (mWs)	71.2312	67.6834	71.54	73.1408

TABLE III. UPLINK SF10, DOWNLINK SF12

	No Compression	**LTC**	**RT-LRbTC, *N*=3**	**RT-LRbTC, *N*=4**
Current (mA)	7.4714	7.3759	7.7266	7.598
Battery voltage (V)	3.9919	3.9918	3.9879	3.9851
Power (mW)	29.807	29.425	30.791	30.061
Energy (mWs)	119.21	117.68	123.15	120.26
Deep sleep energy (mWs)	3.5636	4.0993	4.971	4.632
Transmitting (mWs)	111.0692	108.7984	113.401	110.7958

The oscilloscope settings were 10 mA/div for current probe, 1 V/div for voltage probe, and the horizontal scale was 400 ms/div. The values were measured using oscilloscope mean value measurement function from the oscilloscope screen area which was 4 seconds in total. The settings can be seen in Fig. 2. and Fig. 3.



1 10.0mA 2 1.00 V Ω
M 100mW 400ms
1×2
1 Mean 5.046mA
2 Mean 3.993 V
M Mean 20.13mW
M Area 80.53mWs
400ms
19.80 %
2.50MS/s
10M points
1 29.6mA
19 Feb 2021
19:22:17

Fig. 2. Measurement and LoRa transmission period with SF10 uplink and SF9 downlink. No compression algorithm implemented

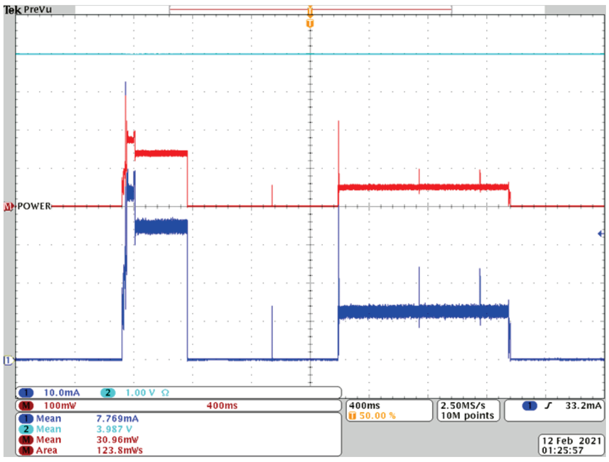


Fig. 3. RT-LRbTC (*N*=3). Measurement, algorithm and LoRa transmission period with SF10 uplink and SF12 downlink

V. Overall Energy Consumption and Battery Lifetime

The overall energy consumption of the LoRa sensor node with the compression algorithm implemented is combined from the measurement event and algorithm consumption (W_M), which happens on a regular basis (measurement interval Δt), the LoRa transmission event consumption (W_S) which frequency of occurrence depends on the measurement interval (Δt), and the compression ratio (*CR*). The overall time is t_x. Between the measurement events the device is in deep sleep mode and its power consumption is P_{ds}. The total energy consumption (W_{TOT}) can be calculated approximately by (1):

$$W_{tot} = P_{ds} t_x + \frac{t_x}{\Delta t} W_M + \frac{t_x}{CR \times \Delta t} W_s \tag{1}$$

Where $P_{ds}t_x$ is the energy that the device uses in deep sleep mode during the whole time t_x. This value includes the measurement and transmission periods as well as the deep sleep values were subtracted from the other energy values in TABLEs I-III. $t_x/\Delta t$ is the number of the measurement periods. $t_x/(CR \times \Delta t)$ is the number of transmission periods.

As the DHT22 temperature sensor has the accuracy of ±0.5 Celsius degrees, it is reasonable to use error bound value ε = 0.5 Celsius degrees for compression algorithms as an example. In [11] the LTC and RT-LRbTC (N = 3) have achieved the compression ratios CR = 9.5-10.2 (LTC) and CR = 5.5-6.0 (RT-LRbTC) for real temperature data with a 10-minute measurement interval (Δt), when the error bound used has been ±0.5 Celsius degrees. RT-LRbTC with N = 4 has not been tested with available temperature data set. The temperature data sets used in [11] were real temperature data sets achieved from Finnish Meteorological Institute's open data service. The data sets were Salla Naruska measurement station data from whole years 2018 and 2019. The temperature data used was measured and presented with 0.1 Celsius degrees resolution.

As an example, the 2 000 mAh battery lifetime can be calculated with 10-minute measurement intervals and with measurement results from TABLEs I, II and III. The equation (1) solved for overall time t_x is (2):

$$t_x = \frac{W_{TOT}}{P_{ds} + \frac{W_M}{\Delta t} + \frac{W_S}{CR \cdot \Delta t}} \tag{2}$$

With a 10-minute measurement interval, the Δt = 600 s. The 2 000 mAh Li-Po battery total energy is W_{TOT} = 7.4 Wh = 26 640 Ws. This amount of available energy was used even though it is a rather optimistic estimation. For example, if the battery powered sensor node is located outside, the temperature can be as low as -30 Celsius degrees in Finland. It is well known that the capacity of lithium-based batteries can drop significantly in cold conditions.

The battery lifetime without any compression algorithm can be estimated by using CR = 1 value. For other parameters the no compression measurement values from TABLEs I-III can be used. The values used (no compression): P_{ds} = 0.46683 mW, W_M = 4.5772 mWs, W_S = 91.4689 mWs. The W_S value is calculated from TABLE II and TABLE III values by taking into account that 49.2 % of transmitting periods were SF10, SF9 events and 50.8 % were SF10, SF12 events. The estimated battery lifetime without any compression algorithm used by equation (2) is: 42 494 353 s = 491.8 days

The battery lifetime with LTC algorithm implemented was calculated with compression ratio CR = 10. Thus, overall battery lifetime with LTC algorithm if the whole battery capacity can be used is: 54 415 973 s = 629.8 days. The battery lifetime is 28.1 % longer than without compression.

In TABLE IV are the results for all tested algorithms with their measured energy consumption values. The compression ratio used for RT-LRbTC algorithms was CR = 6.

TABLE IV. Battery Lifetime with Different Algorithms

	No Compression	**LTC**	**RT-LRbTC, N=3**	**RT-LRbTC, N=4**
CR	1	10	6	6
Battery lifetime (days)	491.8	629.8	615.9	616.0

With RT-LRbTC algorithms the battery lifetime is lengthened by 124.2 days, which is 25.2 % longer lifetime than without any compression.

The deep sleep energy consumption and measurement intervals are very significant parameters for the LoRa-node lifetime together with the compression ratio. In this example the measurement interval is rather long as is often the case in agricultural applications. Thus, the node's deep sleep energy consumption determines mostly the battery lifetime. The Arduino MKR WAN 1310 deep sleep current (measured 117 µA) is not an especially low level for a modern embedded sensor node. If the deep sleep power consumption could be reduced by 50 %, then the battery lifetime without any compression would be 783.6 days, with LTC algorithm 1203.7 days and with RT-LRbTC algorithms 1154 days. Thus, the LTC algorithm would lengthen the lifetime by 420 days, which is 53.6 % longer lifetime than without any compression. The RT-LRbTC algorithms would lengthen the lifetime by 370 days, which is 47.2 % longer than without any compression.

If a bigger reconstruction error is allowed, then the error bound can be higher than 0.5 degrees. With bigger error bound the compression ratio is better thus reducing the overall energy consumption. In [11] the authors have simulated the same compression algorithms with real temperature data with 10-minute measurement interval. For those data sets the compression ratio has been up to 20 for LTC with 1.0 degrees error bound and up to 10 for RT-LRbTC algorithm with 1.0 degrees error bound. Fig. 4. presents the 2000 mAh battery lifetime for LTC and RT-LRbTC (N=3) algorithms with different error bound values. The CR values for different error bounds are the same as in [11]. The battery lifetime lengthening is rather limited if the error bound is increased from 0.5 degrees to 1.0 degrees. The effect is only about 10 days. Thus, it should be considered if using higher error bound is worth of having a significantly bigger reconstruction error.

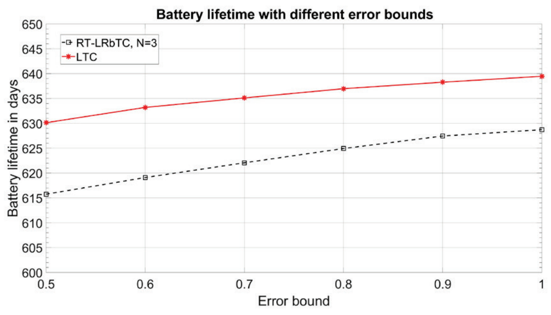


Fig. 4. The effect of error bound on battery lifetime

In general, it is not possible to affect the energy consumption that the measurement itself consumes and the LoRa-transmission consumes a lot. LoRa radio can be set to use certain spreading factor for lower energy consumption, but it can shorter the range. Thus, the most effective ways to reduce overall energy consumption in this kind of sensor node is to choose the low energy consumption embedded platform, which has very low sleep current consumption and to use an effective compression algorithm suitable for low power sensor node. If the measurement interval was shorter, then the data would behave more linearly, and the linearity-based compression algorithms would behave more effectively and result in higher compression ratio. The effect in energy saving achieved with data compression would be bigger because there would be more data transmission periods if no compression algorithm is used, and a higher compression ratio would result in higher reduction in transmission periods. It would underline the compression algorithm's effect on reducing the sensor node's overall energy consumption. In [11] the compression ratio values achieved with LTC and RT-LRbTC (N=3) for air pressure data with 10-minute measurement interval were almost CR = 30 (for LTC) and CR = 15 (for RT-LRbTC) when the error bound was 0.5 hPa. Thus, it can be estimated that the effect of these compression methods on overall energy consumption would be bigger if the air pressure was the measured magnitude.

VI. Conclusions

In this paper we implemented simple sensor data compression algorithms on LoRa-based sensor node. The overall energy consumption was measured with and without implemented compression algorithm. The measurement results demonstrated that the tested compression algorithms are computationally so light that due to calculations they do not have any effect on embedded device energy consumption. The overall saving in energy consumption is due to the reduced amount of radio transmission periods thanks to data compression. The measurement interval was ten minutes, which is rather long but can be typical in agricultural applications measuring some environmental magnitudes. Due to the long measurement interval, the device's sleep energy consumption was proved to be the most significant factor in the device's lifetime.

References

[1] D. Harrison, D. Burmester, W. Seah, and R. Rayudu, "Busting myths of energy models for wireless sensor networks," Electron. Lett., 52: 1412-1414, 2016. https://doi.org/10.1049/el.2016.1591

[2] M. Srbinovska, V. Dimcev and C. Gavrovski, "Energy consumption estimation of wireless sensor networks in greenhouse crop production," *IEEE EUROCON 2017 -17th International Conference on Smart Technologies*, Ohrid, 2017, pp. 870-875, doi: 10.1109/EUROCON.2017.8011235.

[3] J. Rahmé, N. Fourty, K. Al Agha and A. Van den Bossche, "A Recursive Battery Model for Nodes Lifetime Estimation in Wireless Sensor Networks," *2010 IEEE Wireless Communication and Networking Conference*, Sydney, NSW, Australia, 2010, pp. 1-6, doi: 10.1109/WCNC.2010.5506424.

[4] N. Madiyar, and K. Nurzhigit, "Prediction of energy consumption for LoRa based wireless sensors network," Wireless Networks, 26(5), pp. 3507-3520, 2020. doi:10.1007/s11276-020-02276-5

[5] T. Bouguera, J. Diouris, J. Chaillout and G. Andrieux, "Energy consumption modeling for communicating sensors using LoRa technology," *2018 IEEE Conference on Antenna Measurements & Applications (CAMA)*, Vasteras, 2018, pp. 1-4, doi: 10.1109/CAMA.2018.8530593.

[6] T. Bouguera, J. Diouris, J. Chaillout, R. Jaouadi, and G. Andrieux, "Energy Consumption Model for Sensor Nodes Based on LoRa and LoRaWAN," Sensors (Basel, Switzerland), 18(7), p. 2104, 2018, doi:10.3390/s18072104

[7] L. Casals, R. Vidal, and C. Gomez, "Modeling the Energy Performance of LoRaWAN," Sensors, 17(10), p. 2364. 2017, doi:10.3390/s17102364

[8] É. Morin, M. Maman, R. Guizzetti and A. Duda, "Comparison of the Device Lifetime in Wireless Networks for the Internet of Things," in *IEEE Access*, vol. 5, pp. 7097-7114, 2017, doi: 10.1109/ACCESS.2017.2688279.

[9] About LoRaWAN. https://lora-alliance.org/about-lorawan/

[10] T. Schoellhammer, B. Greenstein, E. Osterweil, M. Wimbrow and D. Estrin, "Lightweight temporal compression of microclimate datasets [wireless sensor networks]," *29th Annual IEEE International Conference on Local Computer Networks*, Tampa, FL, USA, 2004, pp. 516-524, doi: 10.1109/LCN.2004.72.

[11] O. Väänänen, and T. Hämäläinen, "Sensor Data Stream on-line Compression with Linearity-based Methods," *2020 IEEE International Conference on Smart Computing (SMARTCOMP)*, Bologna, Italy, 2020, pp. 220-225, doi: 10.1109/SMARTCOMP50058.2020.00049.

[12] O. Väänänen and T. Hämäläinen, "Compression Methods for Microclimate Data Based on Linear Approximation of Sensor Data" The 19th International Conference on Next Generation Wired/Wireless Advanced Networks and Systems NEW2AN 2019, August 26 - 28, 2019, St.Petersburg, Russia.

[13] G. Giorgi, "A Combined Approach for Real-Time Data Compression in Wireless Body Sensor Networks," in *IEEE Sensors Journal*, vol. 17, no. 18, pp. 6129-6135, 15 Sept.15, 2017, doi: 10.1109/JSEN.2017.2736249.

[14] Arduino MKR WAN 1310. https://store.arduino.cc/mkr-wan-1310

[15] DHT22 temperature and humidity sensor. https://www.sparkfun.com/datasheets/Sensors/Temperature/DHT22.pdf

[16] Tektronix TCP0030 Current Probe Instruction Manual. https://www.tek.com/current-probe-manual/tcp0030

VII

EFFICIENCY OF TEMPORAL SENSOR DATA COMPRESSION METHODS TO REDUCE LORA-BASED SENSOR NODE ENERGY CONSUMPTION

by

Olli Väänänen & Timo Hämäläinen, 2022

Sensor Review, vol. 42, no. 5, pp. 503–516, 2022

https://doi.org/10.1108/SR-10-2021-0360

Efficiency of temporal sensor data compression methods to reduce LoRa-based sensor node energy consumption

Olli Väänänen
School of Technology, JAMK University of Applied Sciences, Jyväskylä, Finland, and
Timo Hämäläinen
Faculty of Information Technology, University of Jyväskylä, Jyväskylä, Finland

Abstract
Purpose – Minimizing the energy consumption in a wireless sensor node is important for lengthening the lifetime of a battery. Radio transmission is the most energy-consuming task in a wireless sensor node, and by compressing the sensor data in the online mode, it is possible to reduce the number of transmission periods. This study aims to demonstrate that temporal compression methods present an effective method for lengthening the lifetime of a battery-powered wireless sensor node.

Design/methodology/approach – In this study, the energy consumption of LoRa-based sensor node was evaluated and measured. The experiments were conducted with different LoRaWAN data rate parameters, with and without compression algorithms implemented to compress sensor data in the online mode. The effect of temporal compression algorithms on the overall energy consumption was measured.

Findings – Energy consumption was measured with different LoRaWAN spreading factors. The LoRaWAN transmission energy consumption significantly depends on the spreading factor used. The other significant factors affecting the LoRa-based sensor node energy consumption are the measurement interval and sleep mode current consumption. The results show that temporal compression algorithms are an effective method for reducing the energy consumption of a LoRa sensor node by reducing the number of LoRa transmission periods.

Originality/value – This paper presents with a practical case that it is possible to reduce the overall energy consumption of a wireless sensor node by compressing sensor data in online mode with simple temporal compression algorithms.

Keywords Internet of things, Energy efficiency, Compression, Sensor data, Edge computing

Paper type Research paper

1. Introduction

Sensors are fundamental components of internet of things (IoT) design. According to a report published in 2021, 40% of IoT engineers use environmental sensing in their IoT design, and only 14% do not use sensor technology at all in their IoT design (Farnell, 2021). Typical applications using environmental sensing are, for example, different home control-related applications where the sensors are measuring, for instance, air quality, temperature, humidity and air pressure. Environmental sensors are also widely used in industrial applications and particularly in agricultural applications. In agriculture, precision agriculture (PA) requires up-to-date information from the environment and from the field for decision-making to improve quality and production (Jawad *et al.*, 2017).

Most IoT solutions use wireless connections between the edge device, gateway and cloud. According to the Farnell report, 77% of the engineers who responded to a survey used a wireless connection in their IoT design. Only 23% use wired connectivity (Farnell, 2021). Sensors and wireless sensor networks (WSNs) are fundamental technologies, for example, in smart environment monitoring systems. Smart environment monitoring systems can be used in agriculture for smart farming and for monitoring air quality, water pollution and radiation pollution (Ullo and Sinha, 2020).

Globally, there are already more IoT devices than people. It is estimated that the number of IoT devices will triple from 8.74 billion in 2020 to more than 25.4 billion in 2030 (Farnell, 2021). IoT devices are often battery-powered and will be used in every area of our lives. Therefore, the energy consumption of IoT devices is an important issue. By minimizing the energy consumption, it is possible to lengthen the device or battery lifetime, thus reducing the overall costs. At the same time, it is a well-known fact that the wireless connectivity is the single most energy-consuming task in a wireless IoT sensor node. The most

The current issue and full text archive of this journal is available on Emerald Insight at: **https://www.emerald.com/insight/0260-2288.htm**

Sensor Review
42/5 (2022) 503–516
Emerald Publishing Limited [ISSN 0260-2288]
[DOI 10.1108/SR-10-2021-0360]

Received 14 October 2021
Revised 19 March 2022
Accepted 7 June 2022

energy-consuming modes in wireless sensor nodes are when the radio is transmitting, receiving or idle mode (Harrison *et al.*, 2016; Lin *et al.*, 2021). According to Farnell report (Farnell, 2021), Wi-Fi is the most popular type of wireless connectivity followed by cellular (4 G/LTE/5G), Bluetooth low-energy and LoRa. LoRa is a low-power wide-area network technology that is well suited for energy-constrained sensor devices located far from the base station.

In IoT home applications, it is rather easy to power the devices from the mains power supply, but even these are often battery-powered for installation simplicity. In many other sectors, it is not even possible to use the mains power supply to power the devices. Typical applications where the devices need to be battery-powered or energy-harvesting powered are, for example, different agricultural and environmental monitoring applications where IoT devices can be spread to a geographically wide area in the field (Prauzek *et al.*, 2018).

There are many methods for reducing the overall energy consumption of a wireless sensor node. Wireless sensor nodes consume energy mainly in sensing, processing and data communication. At other times, it can be in the sleep mode (Lin *et al.*, 2021). The effective use of sleep modes between sensor measurements is a significant method for reducing energy consumption (Väänänen and Hämäläinen, 2021). Another method to reduce energy consumption in IoT sensor nodes is to lengthen the sampling interval, thus keeping the device in sleep mode for longer periods (Lin *et al.*, 2021). This also reduces the number of transmitting periods with a wireless connection; thus, it is a very effective way to reduce energy consumption. Lengthening the sampling interval results in lower precision in the measured data, as it is not possible to detect sudden and rapid changes when they occur between measurements. It is impossible to obtain any information regarding the measured magnitude changes and the direction of change between measurements.

Different sensor data compression methods are one solution to improve this situation. It is possible to compress the raw sensor data in the sensor node, thus reducing the amount of data required to transmit via wireless connection (Săcăleanu *et al.*, 2018). It is also possible to reduce the number of transmitting periods by using a temporal compression algorithm, and at the same time, obtain the information if the measured values change rapidly. Many temporal compression methods are computationally light and simple. These methods can be used for IoT devices that are computationally constrained and have limited energy resources.

In this study, the energy consumption of LoRa sensor node with different LoRa modulation parameters and temporal compression algorithms was evaluated with practical measurements. This study demonstrates that the sleep mode energy consumption, LoRa modulation parameters and compression algorithm used have a significant effect on the overall energy consumption of an IoT sensor node. The remainder of this paper is organized as follows. The LoRa and LoRaWAN wireless IoT protocols and their basic parameters are presented in Section 2. Section 3 presents the basics of temporal compression algorithms, and the implemented compression algorithms are presented in more detail. The test and measurement setup and measurement challenges are presented in Section 4. The measurement results with different LoRa parameters and implemented algorithms are discussed in Section 5. The combined results and overall energy consumption with different LoRa parameters and algorithm combinations are presented in detail in Section 6. Finally, Section 7 concludes the paper.

2. LoRa and LoRaWAN

LoRaWAN is a low-power wide-area (LPWA or LPWAN) networking protocol developed to be an energy-efficient wireless protocol to connect battery-powered IoT devices to the internet (LoRa Alliance, What is LoRaWAN Specification). LoRaWAN is optimized to extend the battery lifetime, capacity and range of IoT devices as well as to minimize costs.

Several other wireless technologies are available for use in IoT devices. Wi-Fi and Bluetooth low energy are widely used for communication in personal devices, especially for short distances. Cellular technology is suitable for applications in which a large amount of data must be transmitted over a long range. LoRa offers very low power consumption and a long range for transmitting sensor data a few times per hour (LoRa Alliance, What is LoRaWAN Specification). Another low-power and long-range wireless technology is the SIGFOX. Both LoRa and SIGFOX are asynchronous technologies; therefore, nodes can be in sleep mode and wake up only when there is a need to transmit data (Morin *et al.*, 2017). Each wireless technology has its own characteristics, advantages and disadvantages. Thus, there is not one single technology suitable for every application. LoRa is a very potential technology for sensor devices when the transmitted amount of data is rather limited and low-energy consumption is required. Commercial LoRa networks have good geographical coverage. For example, in Finland, the commercial network is operated by Digita, and its network covers almost the entire country if a terminal device is located outdoors (Digita, LoRaWAN network coverage in Finland). Even for indoor devices, the network covers most of the country.

LoRa is a physical layer that includes wireless modulation, enabling long-range connectivity. LoRa uses chirp spread spectrum (CSS) modulation, which enables low power consumption and long range in wireless connectivity at the same time (LoRa Alliance, What is LoRaWAN Specification).

LoRaWAN is a communication protocol and system architecture that uses the LoRa physical layer to achieve a low-power operation and a long communication range. LoRaWAN network uses a star topology in which the nodes are not associated with a specific gateway/base station. The transmitted data can be received by several base stations, and the network side removes redundant packets (Lora Alliance, What is LoRaWAN Specification).

Three different device classes are described in the LoRaWAN protocol. The most energy efficient of the three classes is Class A. The Class A device does not listen to the downlink messages from the network, except for two short time windows after every uplink transmission (LoRa Alliance, What is LoRaWAN Specification). Thus, between the transmitting periods, the device and LoRa radio can be in sleep mode. In addition to effective modulation, operation at the sub-GHz level enables a long communication range. LoRa communication uses an unlicensed industrial, scientific and medical band (Lavric and

Popa, 2018). Sub-GHz frequency range helps signal penetration through the obstacles between the device and base station. Sub-GHz frequency and LoRa modulation enable the long range as well as the good network coverage for devices located indoors.

The CSS modulation spreads the narrowband signal over a large frequency band, thus enabling the signal to be very resistant to noise and immune for interference (Lavric and Popa, 2018). In LoRa CSS modulation, the spread spectrum is achieved with a chirp signal that continuously varies its frequency (Semtech, What are LoRa and LoRaWAN). LoRa supports spreading factors (SFs) from 6 to 12. The higher SF allows a longer range, but it also results in higher time on air (ToA) values and lower data rates (DRs) (Lavric and Popa, 2018; Semtech, What are LoRa and LoRaWAN). For example, according to Semtech (Semtech, What are LoRa and LoRaWAN), the range with the upper link SF10 is 8 km, but with SF7, the range is only 2 km. These values are examples and vary greatly depending on the circumstances. The ToA is correspondingly 371 ms with SF10 and 61 ms with SF7. The range depends significantly on the terrain, but this provides an idea of the SF effect on the range.

The LoRa nodes have the possibility of setting the DR from 0 to 6. The DR represents a predefined set of LoRa settings such as the SF (Lavric and Popa, 2018). The LoRa specification describes the DR settings as presented in Table 1 (LoRa Alliance, RP002-1.0.0 LoRaWAN Regional parameters). The configuration column presents the SF and channel width.

The LoRaWAN protocol defines an adaptive DR (ADR) mechanism, which optimizes the DR used. LoRa devices located close to the base station do not require a high link budget that is with SF12. ADR optimizes the SF used and minimizes the ToA. The ADR has simple rules for changing the DR used. If the link budget is high, then the SF can be increased and vice versa (Semtech, What is an ADR). It is also possible to set the LoRa node to use a certain DR, but the ADR is designed to optimize the SF and ToA, and thus, it should be the recommended setting. The ADR scheme maximizes the battery lifetime. LoRaWAN also supports optional acknowledgments (ACK) and message retransmissions. The LoRaWAN node can indicate whether an ACK is requested in each transmission. If ACK is required, the node is expecting to receive ACK (confirmation) in one of the two receive windows after message transmission. If the ACK is not received, the LoRaWAN node retransmits the message with the same DR as originally, and then DR decreases every two attempts to lower DR until DR = 0 if the ACK is not received. (Casals *et al.*, 2021). If the ACK is not requested, the LoRaWAN node listens to the possible downlink message from the network in any case but does not retransmit the message if the confirmation is not received. The LoRa network server can send a downlink message even if it is not requested.

Table 1 LoRaWAN DR settings

DR	Configuration	Physical bit rate (bit/s)
0	SF12/125 kHz	250
1	SF11/125 kHz	440
2	SF10/125 kHz	980
3	SF9/125 kHz	1,760
4	SF8/125 kHz	3,125
5	SF7/125 kHz	5,470
6	SF7/250 kHz	11,000

3. Temporal compression methods

Many temporal compression methods are well suited for sensor-based 1D data. Data compression is a common method for reducing data size. Compression methods can be divided into lossless and lossy methods. The compression ratios achieved are not very high with lossless methods (Lin *et al.*, 2019). Lossy algorithms can achieve a compression ratio that is several times higher than that of lossless algorithms, but with the cost of reconstruction error (Lu *et al.*, 2021). There is often a temporal correlation in sensor data if the observation window is short. Temporal compression methods use this temporal correlation (Lin *et al.*, 2019).

The temporal compression methods used in this study are simple and computationally lightweight compression algorithms. The methods used in this study are based on data linearity. The environmental magnitudes behave rather linearly if the observation window is short. For example, air temperature in a shadow does not change significantly in seconds. It normally requires minutes to observe the temperature change, even when it is changing at its extreme speed. If the temperature is rising, it changes quite linearly as it behaves similarly also if it is going down.

The compression methods used in this study either find linear segments from the sensor data stream with certain error bound or use linear regression from previous values to predict future values with allowing certain error bound. Thus, the compressed data set loses some information. These methods are computationally light and thus suitable for constrained battery-powered IoT sensor nodes. These methods are also easy to understand and implement.

The compression algorithms used in this paper were lightweight temporal compression (LTC) and two versions of the real-time linear regression-based temporal compression (RT-LRbTC). The two versions of the RT-LRbTC vary from each other by the number of sensor values used to calculate the regression line. Three and four values (*N* values) versions were tested.

3.1 Lightweight temporal compression

The LTC is a well-known compression algorithm that is particularly suitable for environmental data. It was first presented by Schoellhammer *et al.* (2004). It has also been used to compress sensor data in wireless body sensor networks (WBSN) (Giorgi, 2017). Several modifications of the original LTC have been presented (Parker *et al.*, 2013; Azar *et al.*, 2018; Sarbishei, 2019; Li *et al.*, 2018; Klus *et al.*, 2021).

LTC has proven to be a very effective compression algorithm, particularly for linearly behaving environmental sensor data (Väänänen and Hämäläinen, 2019). One major disadvantage of LTC is its unpredictable latency. As the LTC compresses the data in the online mode by finding the best and longest linear segment from the incoming sensor data, it sends the linear segment endpoint to the sink only when the algorithm finds it as the new value falls off from the linear segment. If the method compresses the data very efficiently, then the transmitting intervals become long, and the receiving side does not even know in which direction the values are changing. Thus, LTC is not well suited for compressing sensor data in real-time or near-real-time applications (Giorgi, 2017).

In this study, the LTC was used as in its original version presented by Schoellhammer *et al.* (2004). The same version was used by Väänänen and Hämäläinen (2019, 2020) as MATLAB version. In this study, the LTC was programmed for the Arduino and implemented on the Arduino MKR WAN 1310 LoRa board.

The LTC itself is computationally light because with every new value, it is only necessary to make a comparison between the new value with error bound extremes and the previously calculated upper and lower limit lines. As a result of the comparison, a maximum of two new lines must be calculated to create new upper and lower limit lines that will be used with the next value. Calculating the new limit line parameters (slope and y-intercept) requires one division, one multiplication, one summation and subtractions, thus resulting in a computationally simple algorithm.

3.2 Real-time linear regression-based temporal compression

RT-LRbTC uses linear regression calculated from previous sensor values to predict future sensor values. This type of compression algorithm works well if the measured data behaves rather linearly. This is the case for many environmental magnitudes, such as temperature, humidity and air pressure.

RT-LRbTC is based on several other simple linear regression-based algorithms. Other simple linear regression-based compression algorithms have also been developed (Väänänen and Hämäläinen, 2019; Hung *et al.*, 2013; Duvignau *et al.*, 2019). RT-LRbTC was originally presented by Väänänen and Hämäläinen (2020). RT-LRbTC was developed especially for compressing sensor data in online mode. It has a shorter inherent latency than other linearity-based compression methods, which is its most significant benefit compared to other methods (Väänänen and Hämäläinen, 2020).

The inherent latency of the RT-LRbTC algorithm is one measurement interval Δt in the linear section. The algorithm uses N previously measured values to calculate the regression line, which predicts future values with a certain error bound (ε) allowed from the line. New measured sensor value is compared to the previously calculated regression line. If the difference from the line is smaller than ε, then the algorithm waits for the next measured value. When the new value falls off from the linear section (distance greater that the error bound ε from the line), then the new line is calculated from the values already available. From the calculated line, the line parameters and time stamp are sent/stored. On the network side, if new parameters are not received, then the values follow the previous regression line with the error bound allowed. N is a minimum of three, and in this study, N was three and four (two versions). Calculating the new regression line requires some calculation: the sum of N times x_k, x_k^2, y_k, $x_k y_k$ and the square of the sum of x_k. x_k is the time stamp (sample number) and y_k is the measured value. $N = 3$ is the basic form of RT-LRbTC, and $N = 4$ version was also tested in this study to see whether the required calculations had any effect on the overall energy consumption.

The RT-LRbTC algorithm was tested with real measured temperature data by Väänänen and Hämäläinen (2020). In the data sets used, the measurement interval was 10 min. The data sets were obtained from the Finnish Meteorological Institute's open data service. The data sets used were two full-year data sets with a 10-min measurement interval. As a result, with 0.5°C error bound, the RT-LRbTC algorithm has achieved compression ratio CR = 5.5–6.0 (Väänänen and Hämäläinen, 2020). The error bound ε = 0.5°C represents the maximum difference between the original measured values and reconstructed values from the compressed data. The average reconstruction error is smaller than the error bound used. The compression ratio achieved with a certain linearity-based compression algorithm depends on the measured magnitude characteristics and error bound used. A higher error bound results in a better compression ratio, but with the cost of a larger reconstruction error.

4. LoRa device energy consumption with compression algorithm implemented

The LoRa SF used determines the ToA and thus it also determines how much energy a transmission consumes. This is because if the ToA is longer, then the LoRa radio is transmitting for a longer time and consumes energy for a longer time as well.

Väänänen and Hämäläinen (2021) measured the LoRa device energy consumption with ADR set on, and thus, the LoRa device was transmitting with a DR of 2 (SF10). The device was in a stable place, and thus, the conditions did not change, and the DR remained constant. The only difference between the energy consumption of the transmitting periods is the difference between whether the downlink is received or not. If the downlink is received, the energy consumption increases as the device receives the data. If the downlink is not received, then the LoRa device only listens shortly during the two receive windows, and the overall energy consumption is lower. The downlink message is used by the network server for acknowledge messages (ACK) (Maudet *et al.*, 2021).

In this study, a setup similar to that of Väänänen and Hämäläinen (2021) was used for the practical experiments and energy consumption measurements. In this study, the ADR was not on, and the DR was set (fixed) for every transmitting period. The DRs tested ranged from DR0 to DR5. All of these DRs have a channel width of 125 kHz. The downlink SF was automatically set from the network and was not controlled in this study. Normally, a downlink message has the same SF as an uplink message if the first receive window is used. For the second receive window, the SF12 is used as the default (Casals *et al.*, 2021).

The LoRa device used was Arduino MKR WAN 1310 board. The Arduino MKR WAN 1310 has a Microchip SAM D21 32-bit Arm Cortex-M0+ based microcontroller and a Murata CMWX1ZZABZ LoRa module (Arduino MKR WAN 1310). The Arduino MKR WAN, 1310 was chosen because of its simplicity and ease of use. It is also very popular among hobbyists but is also widely used in industry for piloting and experiments.

The temperature was measured using a DHT22 sensor. The DHT22 also measures humidity, but for this experiment, only temperature was used. DHT22 is a low-cost sensor with a digital output. DHT22 sensor has measurement resolution for temperature 0.1°C and its accuracy is $< \pm 0.5$°C (DHT22 temperature and humidity sensor). Another popular and rather similar temperature and humidity sensor is DHT11, which is often used in agricultural applications (Rehman *et al.*, 2022). In this study, DHT22 was chosen because it is slightly more accurate and has wider measurement ranges; however, it is only slightly more expensive. It is well suited for this type of experiment, where the accuracy requirement is not high.

The Arduino MKR WAN 1310 was powered by a 2,000 mAh lithium-polymer (LiPo) battery. The LiPo battery has a 3.7 V nominal voltage, resulting theoretically 7.4 Wh total energy capacity, which equals 26,640 Ws. The battery has a JST-PH connector that can be directly used with the Arduino MKR WAN 1310 board. The energy consumption of the Arduino board is lower when powered by a battery than when powered by the USB port. If the board is powered by the USB port, then one LED is always ON, and also USB IC-chip consumes extra energy. The device setup is illustrated in Figure 1. The setup was built for this experiment only, but it could be used in real practical case when enclosed properly against environmental conditions.

4.1 Measurement setup

For the experimental energy consumption measurements, the Arduino board was set to measure the temperature with the DHT22 sensor at regular intervals. After each measurement, the compression algorithm was applied by microcontroller. If the result of the algorithm required data transmission, then the data were sent via the LoRa connection. After every measurement event and possible data transmission, the device was set to go to deep-sleep mode. The device woke up only for the measurement, compression algorithm and possible transmission periods. The LoRa network was a commercial network operating in Finland.

The current consumption in the active mode was measured using a shunt resistor. Two oscilloscope channels were used to measure the voltage across a 10 Ω shunt resistor, which was in series at the battery plus wire. Both oscilloscope channels used battery negative terminal as the reference level. The measurement setup is illustrated in Figure 2(a). Current

Figure 1 Arduino MKR WAN 1310 in test setup

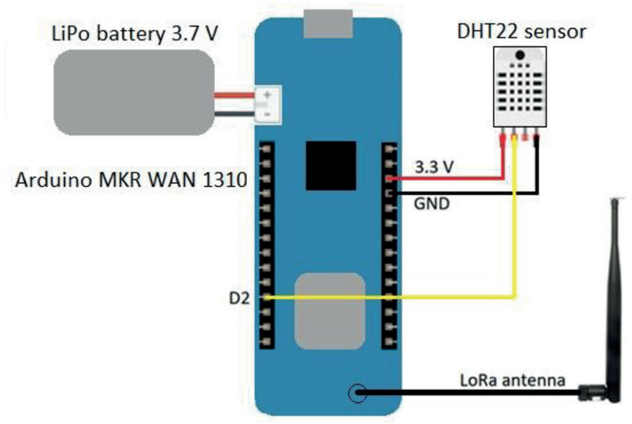
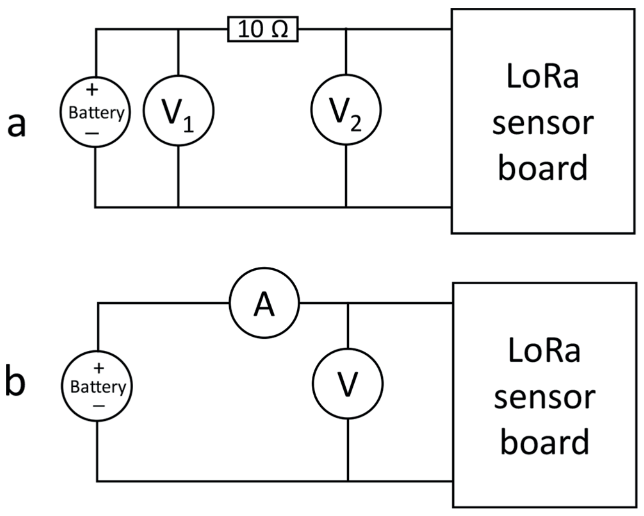


Figure 2 Current consumption measurement circuits



consumption was calculated from the measured voltages $I = (V_1 - V_2)/R$, where R is the shunt resistor (10 Ω). V_2 can also be used to measure the supply voltage in the battery connection of the board. Thus, the power consumption of the device is:

$$P = V \cdot I = V_2 \cdot \frac{(V_1 - V_2)}{R} \tag{1}$$

In a previous study by Väänänen and Hämäläinen (2021), current was measured using an oscilloscope current probe. That kind of setup is shown in Figure 2(b), where the ammeter is the current probe. The current probe is very easy to use, but its accuracy is poor when measuring low-level values. The shunt resistor measurement of the current is more accurate and repeatable. The results obtained by Väänänen and Hämäläinen (2021) were used in this paper for comparison when available.

In this study, the current consumption in the deep-sleep mode was measured using a digital multimeter (DMM) in series with a battery wire. The DMM is more accurate for measuring low-level deep-sleep current than the shunt resistor method used for active periods. A high-quality DMM can reliably measure μA level current value if it remains stable. The DMM cannot be used to measure current consumption for active periods because the current level is changing and does not remain static.

The device used for the measurements was a Tektronix MSO 4104 mixed signal oscilloscope with a 1 GHz measurement bandwidth, and the probes used were TAP1500 active voltage probes. The voltage in the battery connector of the board was also measured separately using a Tektronix P6139A passive voltage probe. The DMM used was a Tenma RS232C TrueRMS model.

5. Measurement results

The measurements were carried out without implementing a compression algorithm and with compression algorithms to see whether the algorithm calculations have any effect on energy consumption. The energy consumption for sensor measurement and algorithm calculations was measured with all algorithm combinations and without an algorithm. Transmission energy consumption was measured using two different payloads. If the measured raw or compressed value is sent with a time stamp, only 8 bytes are needed to transmit. This is the case without a compression algorithm or with the LTC algorithm. With the RT-LRbTC algorithm, a total of 12 bytes are transmitted because two line parameters (slope and base) and a time stamp are needed to transmit.

Figure 3 presents the overall LoRa sensor node energy consumption scenario over time. Number 1 in Figure 3 represents deep-sleep energy consumption (base consumption). Number 2 represents the extra energy consumption of the sensor measurement and data processing (with or without the compression algorithm implemented). It occurs at regular intervals that are determined by the measurement interval. Number 3 represents the energy consumption of the LoRa transmission. LoRa transmission is required after every measurement event if no compression algorithm is implemented. If a compression algorithm is implemented, LoRa transmission does not occur after every measurement event. The deep-sleep energy consumption is presented here as existing all the time, and the measurement events and LoRa transmissions were presented and measured as extra energy consumption on top of deep-sleep base energy consumption. When the device is measuring or transmitting, it is not in deep-sleep mode, but for measurement purposes, this type of presentation is easier. In any case, the deep-sleep power consumption is a fraction of the measurement event and/or transmitting event power consumption.

5.1 Deep-sleep current consumption

The deep-sleep current consumption was measured without implementing an algorithm, and the result was confirmed with algorithms implemented. As the device was in deep-sleep mode, there was no difference if the algorithm was implemented or not. The device was set to go into deep-sleep mode between the measurement periods. A similar setup was used by Väänänen and Hämäläinen (2021), and the results obtained in this study were in same level. The measured deep-sleep current was 106–107 μA. When the device goes into deep-sleep mode, the current drops immediately to 150–160 μA, and after 10–20 s, it reaches 107 μA level. The voltage in the battery connector of the board was measured with an oscilloscope at the same time. The battery voltage was 3.99 V in this case, thus resulting in deep-sleep power consumption $P_{ds} = V_{battery} \cdot I_{ds} = 3.99\,\text{V} \times 117 \cdot 10^{-6}\,\text{A} = 4.6683 \cdot 10^{-4}\,\text{W} = 0.46683\,\text{mW}$.

5.2 Sensor measurement and algorithm energy consumption

As the measurement interval is typically minutes, the sensor node remains most of the time in deep-sleep mode, but it wakes up with a regular basis to perform the measurement and algorithm calculations. If the result from the algorithm calculation is that there is no need to transmit any data, then the device returns to the deep-sleep mode. The real-time clock (RTC) runs even in deep-sleep mode and wakes the device up on a regular basis, which is determined by the measurement interval.

The energy consumed by the sensor measurement, and possible algorithm was measured with the oscilloscope using a shunt resistor, as explained in the measurement setup section. The oscilloscope measurement results are shown in Figure 4. Channels 2 and 3 (blue and purple lines on top of one another) were used to measure the voltage difference across the shunt resistor. Channel 4 (green line) measured the voltage in the battery connector. The power line (red MATH-line in mW) was calculated using the oscilloscope MATH-function from Channels 2, 3 and 4 data: $P = U \times I = \text{CH4} \times ((\text{CH2} - \text{CH3})/10)$. CH2-CH3 denotes the voltage across the shunt resistor. 10 is the resistor size in ohms. The oscilloscope measurement function was used to calculate the MATH line area (integral), which is the total energy consumed in the oscilloscope window timescale (5.022 mWs during 2 s in Figure 4). Then, the average value of the red MATH-line was measured before the measurement event (device wake up) when the device was in deep-sleep mode. The average deep-sleep value (average power) was multiplied by the timescale used in the oscilloscope screen (2 s in Figure 4) to obtain the base energy consumed, which was subtracted from the total energy measured (5.022 mWs in Figure 4). Thus, additional energy consumption from the sensor measurement and data processing was measured.

The same measurement was repeated a minimum of ten times for each algorithm implemented as well as without the algorithm implemented. The measurement results are listed in Table 2. The average value was calculated from all measurements using a certain algorithm (a minimum of ten measurements with each algorithm). Max and Min values show the maximum and minimum measured values, and Std Dev is the standard deviation calculated from all the measured values with the certain algorithm implemented. Last row presents measurement results with current probe (Väänänen and Hämäläinen, 2021). Figure 5 shows the average results with the maximum, minimum and standard deviation values for each algorithm.

It can be seen from the results presented in Table 2 and Figure 5 that the effect of the algorithm on the measurement and data acquisition event energy consumption is negligible. The algorithms implemented and evaluated were computationally so light that the possible effect on the energy consumption was smaller than the measurement inaccuracy.

5.3 LoRa transmission energy consumption

The scenario for the LoRa transmission energy consumption is shown in Figure 6 (as a function of time). Number 1 in the figure is the base energy consumption, which is the deep-sleep energy consumption. Number 2 represents the sensor measurement, data acquisition and algorithm energy consumption in addition to base energy consumption. Number 3 is the LoRa transmission uplink, and number 4 is the LoRa transmission downlink if received (in Figure 3, the uplink and downlink energy consumptions are combined and presented by number 3).

Figure 3 LoRa sensor node overall energy consumption scenario

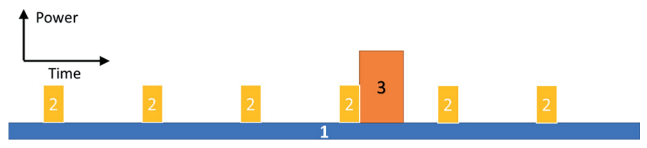
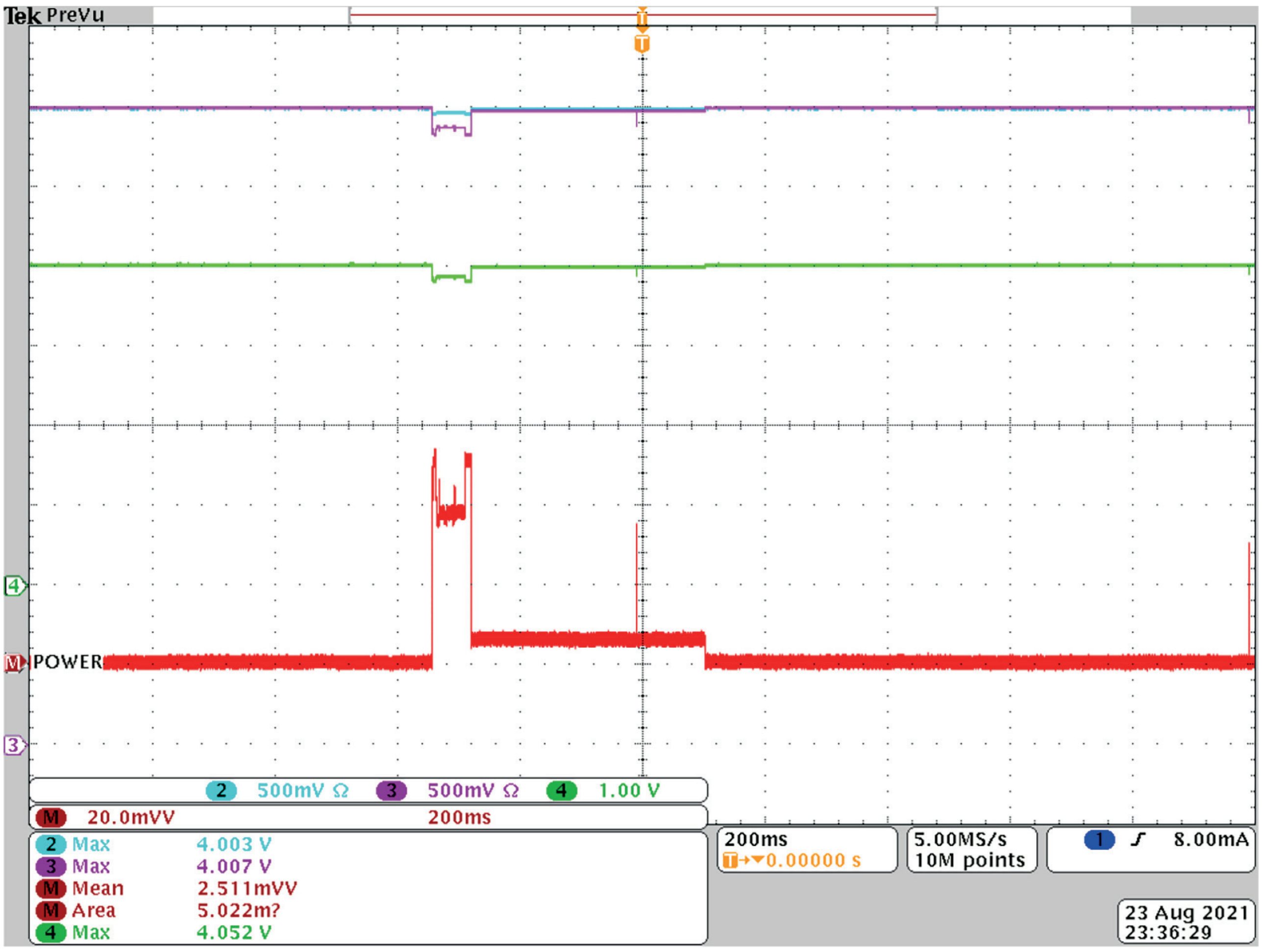


Figure 4 Sensor measurement and data processing energy consumption



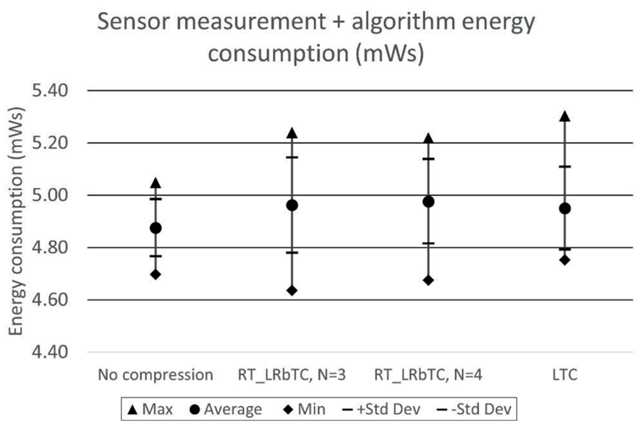
```
2  500mV Ω   3  500mV Ω   4  1.00 V
M  20.0mVV            200ms
2 Max     4.003 V
3 Max     4.007 V
M Mean    2.511mVV
M Area    5.022m?
4 Max     4.052 V
200ms            5.00MS/s     1  ƒ  8.00mA
T→▾0.00000 s     10M points
                              23 Aug 2021
                              23:36:29
```

Table 2 Sensor measurement and algorithm energy consumption with and without algorithms implemented. Results are in mWs

	No compression (mWs)	RT-LRbTC, *N* = 3 (mWs)	RT-LRbTC, *N* = 4 (mWs)	LTC (mWs)
Max	5.05	5.24	5.22	5.30
Average	**4.88**	**4.96**	**4.98**	**4.95**
Std dev	0.11	0.18	0.16	0.16
Min	4.70	4.64	4.68	4.75
From current probe measurement	4.57	4.78	4.83	4.78

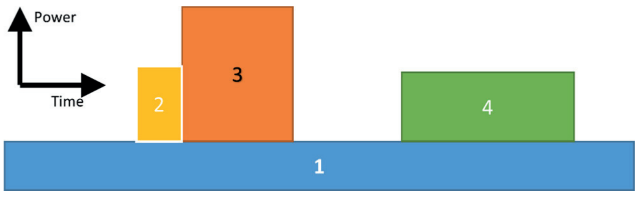
In this measurement setup, the overall energy consumption was measured using the oscilloscope from the timescale that was visible on the oscilloscope screen. The oscilloscope MATH-function was used to calculate the overall power (red MATH-line in Figures 7, 8 and 9) line in mW, and its area (integral) was calculated using the oscilloscope measurement function (in mWs). Then, the average base power level was measured using the oscilloscope ZOOM function from the time before the device wakes up (in deep-

Figure 5 Sensor measurement and possible algorithm overall energy consumption



sleep mode). The average power was used to calculate the average base energy consumption at that timescale (oscilloscope screen, 10 s in Figures 7, 8 and 9). This average base energy (number 1 in Figure 6) was subtracted from the overall measured energy consumption. This results

Figure 6 LoRa node energy consumption scenario



in extra energy that the sensor, algorithm and LoRa transmission add to the base energy consumption.

The sensor measurement and algorithm effect (Table 2, which is number 2 in Figure 6) was then subtracted from the measurement results, resulting in transmission-only energy consumption. LoRa transmission energy consumption was measured for 8 bytes (for LTC and no compression cases) and 12 bytes (for RT-LRbTC cases) payload situations with every SF (SF7-SF12). The DR can be adjusted in Arduino MKR WAN 1310 by enabling ADR and setting a certain DR value:

- modem.setADR(true); and
- modem.dataRate (0);//set data rate to be 0-5.

This needs to be done before every transmission period when the radio wakes up. The total transmission energy consumption for every SF case was measured a minimum of ten times, and the average values are listed in Table 3.

The significant differences in energy consumption depending on the SF used can be seen in Table 3. The downlink was sent from the network side every time, even though the ACK was not required, but quite often, it was not received. If the downlink is not received, then the transmission period energy consumption is lower, but that situation should not be the normal case. If these values are used to predict the device lifetime, the values with the downlink received should be used as the worst case for the energy consumption.

The difference between the received and unreceived downlinks can be seen in Figures 7 and 8. In Figure 8, the LoRa radio opens two short receive windows after transmission. The first receive window is approximately 1 s after the transmission, and the second window is 1 s after the first window. In Figure 7, the LoRa radio opens the first receive window 1 s after transmitting, and in this case, the LoRa radio receives the downlink message. The SF effect on the ToA is shown in Figures 7, 8 and 9. In Figures 7 and 8 with SF12, the transmission takes approximately 1,500 ms, while with SF8 (in

Figure 7 Lora transmission (8 bytes) with uplink SF12 and downlink SF12

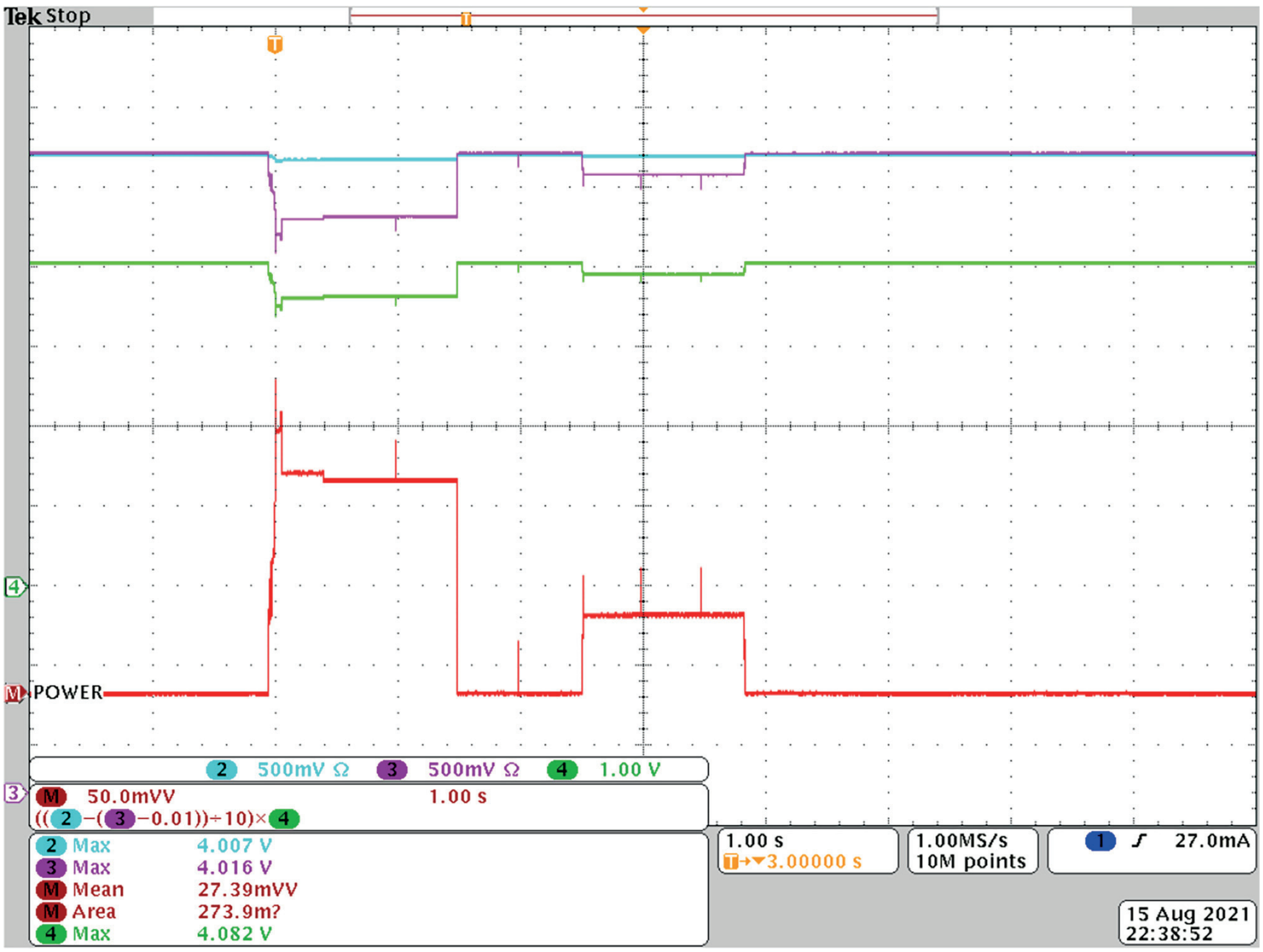


Figure 8 LoRa transmission (8 bytes) with uplink SF12 and downlink not received

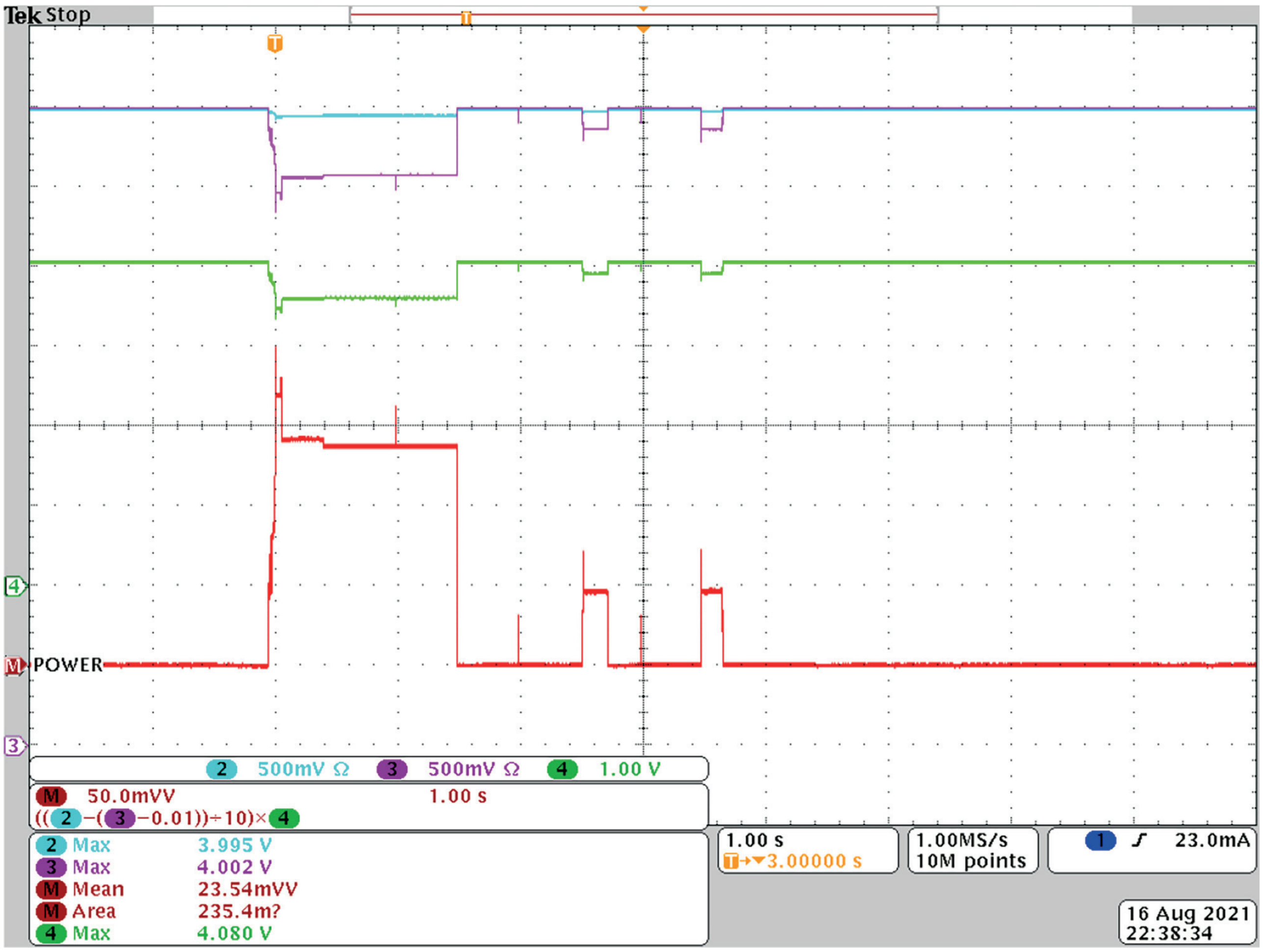


Figure 9), it takes approximately 100–200 ms. That explains the significant difference in energy consumption.

6. Overall energy consumption and average power consumption

The overall energy consumption of the LoRa sensor node is a combination of deep-sleep consumption, measurement event consumption, algorithm execution and possible LoRa transmission. In this study, the sensor measurement event and the algorithm execution were combined. The sensor measurement and algorithm calculations (W_M) take place on a regular basis and are determined by the measurement interval (Δt). The LoRa transmission event (energy consumed W_S) is determined by the measurement interval but also by the compression ratio (CR) that the algorithm achieves. The overall energy consumed during time t_x can be estimated using equation (2):

$$W_{tot} = P_{ds}t_x + \frac{t_x}{\Delta t}W_M + \frac{t_x}{CR \times \Delta t}W_s \tag{2}$$

where $P_{ds}t_x$ is the energy consumed by the device in the deep-sleep mode during time t_x. $t_x/\Delta t$ is the number of measurement periods. $t_x/(CR \times \Delta t)$ is the number of transmission periods. It can be seen from the equation that it is possible to minimize the overall energy consumption either by lengthening the measurement interval or using a compression algorithm, which results in a high compression ratio for the measured data stream. Other possibilities would require different hardware solutions.

If the total available energy is known (battery capacity for example), then the overall lifetime can be solved from equation (2):

$$t_x = \frac{W_{ToT}}{P_{ds} + \frac{W_M}{\Delta t} + \frac{W_S}{CR \cdot \Delta t}} \tag{3}$$

The average power consumption can be derived from equation (2) by dividing by time t_x as $P = W/t$. Resulting in equation (4):

$$P_{avg} = P_{ds} + \frac{W_M}{\Delta t} + \frac{W_S}{CR \cdot \Delta t} \tag{4}$$

The DHT22 temperature sensor has an accuracy of ±0.5°C. Thus, it was reasonable to use the error bound value ε = 0.5°C for the compression algorithms. Väänänen and Hämäläinen (2020)

Figure 9 LoRa transmission (8 bytes) with uplink SF8 and downlink SF10

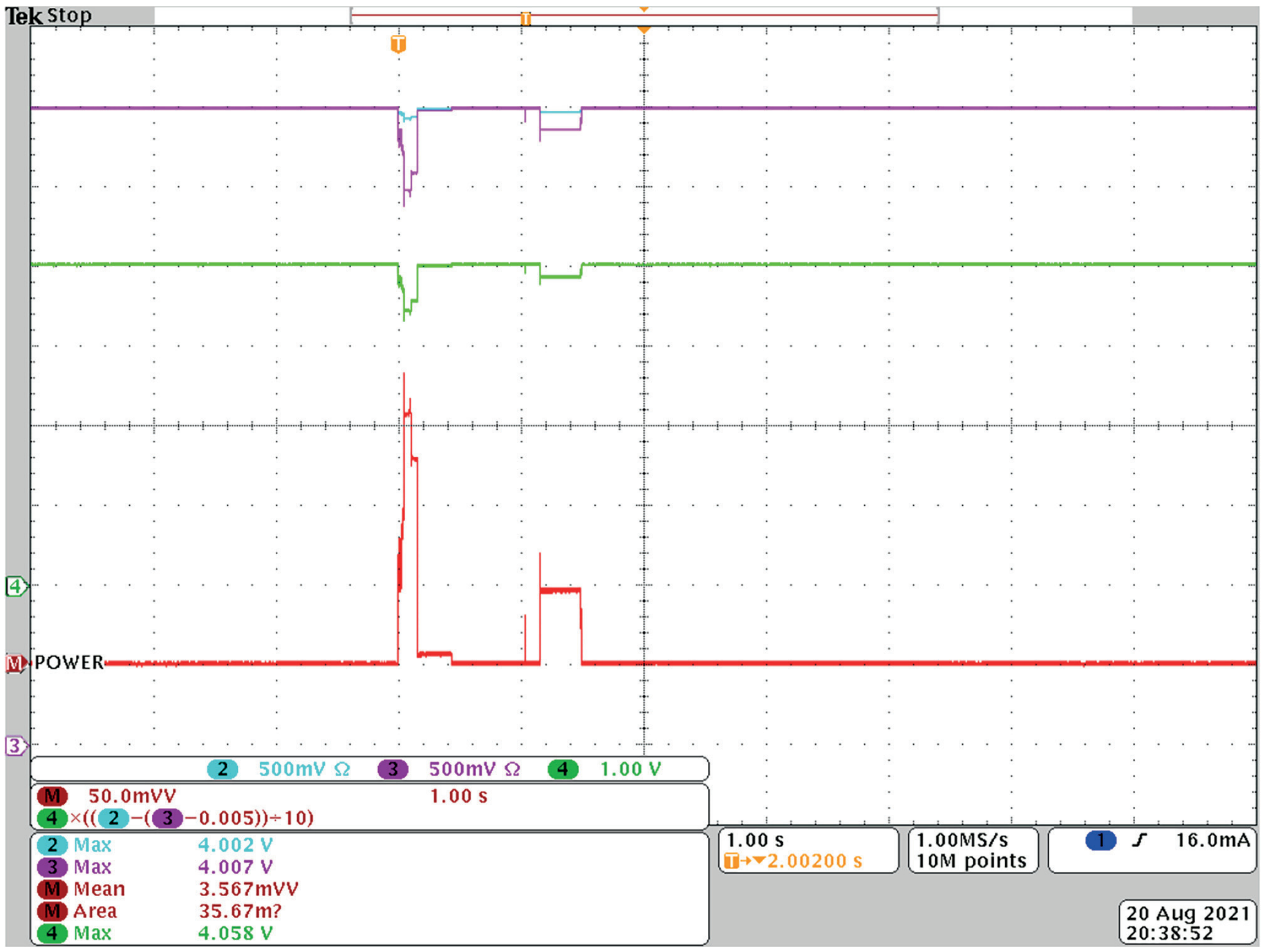


Table 3 LoRa Transmission energy consumption with different SFs and two different payloads. Results are in mWs

	8 bytes (mWs)	12 bytes (mWs)
Uplink SF12, downlink SF12	248.89	257.82
Uplink SF12, downlink SF9 (downlink not received)	201.02	212.32
Uplink SF11, downlink SF12	163.93	162.73
Uplink SF11, downlink SF9 (downlink not received)	119.63	120.12
Uplink SF10, downlink SF12	104.41	107.60
Uplink SF10, downlink SF9 (downlink not received)	65.28	68.03
Uplink SF9, downlink SF11	59.55	58.60
Uplink SF9, downlink SF9 (downlink not received)	41.01	42.50
Uplink SF8, downlink SF10	32.54	33.26
Uplink SF8, downlink SF9 (downlink not received)	27.89	28.78
Uplink SF7, downlink SF9	18.11	19.09
Uplink SF7, downlink SF9 (downlink not received)	13.99	No data
Uplink SF7, downlink SF9 (downlink received in second window)	20.32	No data

tested the LTC and RT-LRbTC algorithms for real temperature data sets with a 10-min measurement interval. Temperature data sets were obtained from the Finnish Meteorological Institute's open data service. The data sets used were 2018 and 2019 full-year data from the Salla Naruska measurement station. The temperature data were measured in degrees Celsius, with a resolution of 0.1°. With ε = 0.5°C, the compression algorithms achieved compression ratios of CR = 9.5–10.2 with LTC and CR = 5.5–6.0 with RT-LRbTC (N = 3).

Table 4 lists the average power consumption for different compression algorithms with different SF scenarios. The measurement interval was 10 min (Δt = 600 s). The values in

Table 4 Average power consumption with different algorithms implemented and with certain compression ratios. Results are in mW

	No compression (mW)	RT_LRbTC, *N* = 3 (mW)	RT_LRbTC, *N* = 4 (mW)	LTC (mW)
Uplink SF12, downlink SF12	0.898	0.548	0.548	0.517
Uplink SF12, downlink SF9 (downlink not received)	0.818	0.535	0.535	0.509
Uplink SF11, downlink SF12	0.756	0.522	0.522	0.503
Uplink SF11, downlink SF9 (downlink not received)	0.682	0.510	0.510	0.496
Uplink SF10, downlink SF12	0.657	0.506	0.506	0.493
Uplink SF10, downlink SF9 (downlink not received)	0.592	0.495	0.495	0.487
Uplink SF9, downlink SF11	0.582	0.493	0.493	0.486
Uplink SF9, downlink SF9 (downlink not received)	0.551	0.488	0.488	0.483
Uplink SF8, downlink SF10	0.537	0.486	0.486	0.481
Uplink SF8, downlink SF9 (downlink not received)	0.530	0.484	0.484	0.481
Uplink SF7, downlink SF9	0.513	0.482	0.482	0.479
Uplink SF7, downlink SF9 (downlink not received)	0.506	–	–	0.478
Uplink SF7, downlink SF9 (downlink received in second window)	0.517	–	–	0.479

Table 4 were calculated using equation (4) from the measured values from Tables 2 and 3 (8 bytes results for LTC and no compression, 12 bytes results for RT-LRbTC algorithms). The compression ratios were *CR* = 10 for LTC and *CR* = 6 for both RT-LRbTC algorithms, which are realistic values and were achieved by Väänänen and Hämäläinen (2020).

There were no significant differences in the power consumption values between the algorithms tested. The differences were larger for high SF values when the effective compression algorithm can achieve energy savings by reducing the number of LoRa transmission periods. LoRa transmission periods are significant energy consumers, particularly if the used SF is high. With high SF values, the compression algorithms used were very effective for reducing energy consumption. This is clearly shown in Figure 10 (from Table 4).

The battery used in this experiment was a 2,000 mAh LiPo battery with 3.7 V nominal voltage. Its overall capacity is 7.4 Wh, which is 26,640 Ws. This capacity is the nominal capacity in the optimal situation. For example, in cold weather, the capacity of lithium-based batteries significantly collapses (Li *et al.*, 2017). Aging also affects the battery capacity.

Figure 10 LoRa node power consumption with different SF values

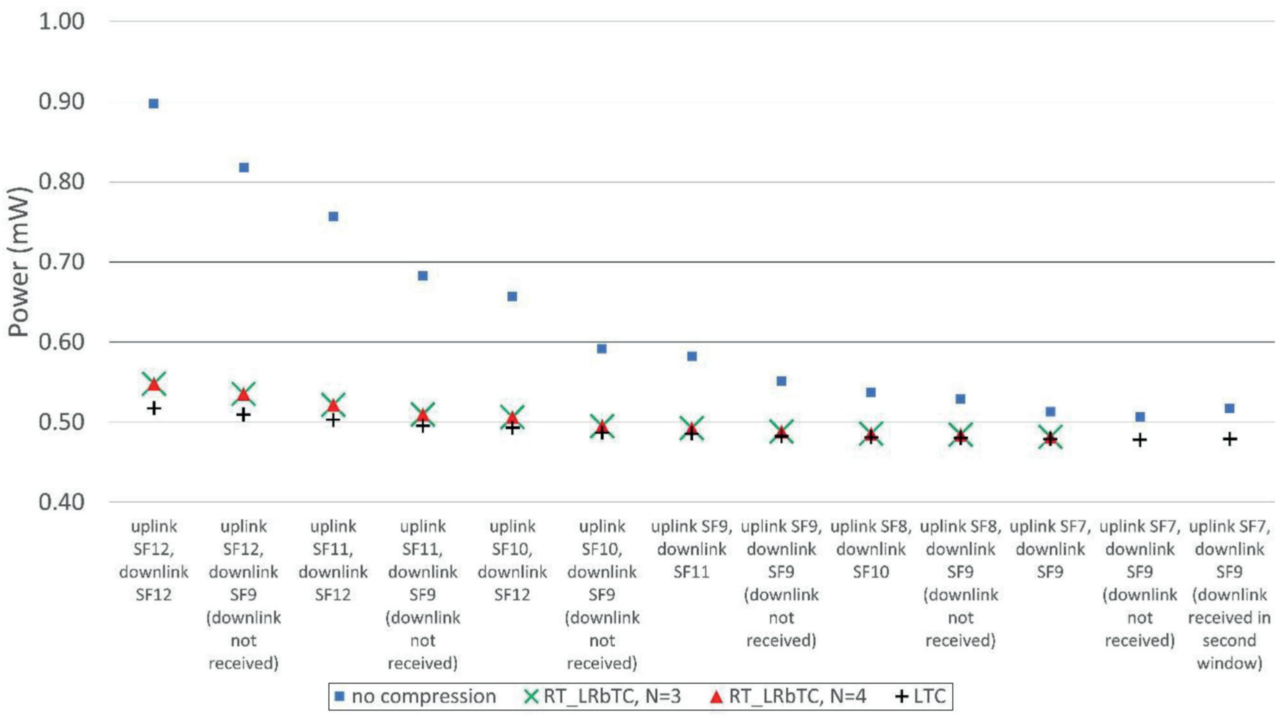


Table 5 Battery lifetime in days with different scenarios

	No compression (days)	RT-LRbTC, N = 3 (days)	LTC (days)
Uplink SF12, downlink SF12	343.4	562.6	596.0
Uplink SF12, downlink SF9 (downlink not received)	376.9	575.8	605.3
Uplink SF11, downlink SF12	407.7	591.0	612.7
Uplink SF11, downlink SF9 (downlink not received)	451.8	604.8	621.9
Uplink SF10, downlink SF12	469.2	608.9	625.0
Uplink SF10, downlink SF9 (downlink not received)	520.9	622.4	633.4
Uplink SF9, downlink SF11	529.5	625.7	634.7
Uplink SF9, downlink SF9 (downlink not received)	559.1	631.5	638.7
Uplink SF8, downlink SF10	574.2	634.8	640.6
Uplink SF8, downlink SF9 (downlink not received)	582.2	636.4	641.6
Uplink SF7, downlink SF9	600.7	640.0	643.8
Uplink SF7, downlink SF9 (downlink not received)	608.9	No results	644.7
Uplink SF7, downlink SF9 (downlink received in second window)	596.5	No results	643.3

The battery lifetime was calculated using equation (3) in the case where full capacity was available. The values used were W_{TOT} = 26,640 Ws and $P_{ds} = 4.6683 \cdot 10^{-4}$ W. W_M and W_S values were from the Tables 2 and 3. *CR* values of 10 for LTC and 6 for RT-LRbTC. The battery lifetimes in days for different algorithms and SF scenarios are listed in Table 5. RT-LRbTC with $N = 4$ is not presented here because its results are very close to RT-LRbTC with $N = 3$.

As can be seen from Table 5, it is easy to achieve over an 18-month lifetime if the RT-LRbTC algorithm is used (with a 0.5° error bound). Without implementing a compression algorithm, it is possible to have less than a 12-month lifetime if the device is located at a long distance from the base station, or if there are obstacles between the device and base station (thus using a high SF value). In a good network coverage situation, the difference is small between the compression algorithm used or without compression algorithm implemented. Generally, the deep-sleep current consumption of 107 μA is rather high for a modern microcontroller-based LoRa node, and it determines the overall lifetime.

In research by Väänänen and Hämäläinen (2021), the DR was not fixed, and instead, the ADR was used. Thus, in that case, the LoRa node was always transmitting with SF10, and in approximately 50% of the transmitting periods, the downlink was received (SF12). The average transmission energy consumption was measured and calculated to be 91.47 mWs. Using this value, the battery lifetime was calculated to be approximately 490 days if no compression algorithm was implemented, 630 days if the LTC algorithm was used and 616 days if the RT-LRbTC algorithm was used. This case is valid in that situation; however, in some other circumstances, the LoRa node may use other SF values, and its effect on the lifetime can be estimated by the results presented in Table 5.

7. Conclusions

From the results achieved in this study, the LoRa DR has a significant effect on the overall power consumption of the LoRa sensor node, especially if no compression algorithm is used. However, normally it is not possible to control the DR because the ADR adjusts the optimal SF value. If the base station is very far away, then a high SF must be used to achieve that long range, resulting in higher power consumption.

Simple temporal compression algorithms are very effective for reducing the overall energy consumption of the LoRa sensor node if the reconstruction error, determined by the error bound, is acceptable. From the results achieved in this study, the algorithm calculations did not have a significant effect on energy consumption. Nevertheless, these algorithms can significantly reduce the number of LoRa transmission periods and thus achieve significant energy consumption savings. The overall reduction in energy consumption was due to the reduced number of radio transmission periods. The LTC algorithm is very effective and simple algorithm, but its unpredictable latency is not well suited for online applications with latency requirements. RT-LRbTC is not as effective compression algorithm, and it is a bit more complicated, but with predictable latency, it is well suited for compressing environmental data in the online mode. In this research, the measurement interval was rather long, and thus, the LoRa node deep-sleep consumption became a significant factor determining the device lifetime.

References

Azar, J., Makhoul, A., Darazi, R., Demerjian, J. and Couturier, R. (2018), "On the performance of resource-aware compression techniques for vital signs data in wireless body sensor networks", *IEEE Middle East and North Africa Communications Conference (MENACOMM), Jounieh*, pp. 1-6, doi: 10.1109/MENACOMM.2018.8371032.

Casals, L., Gomez, C. and Vidal, R. (2021), "The SF12 well in LoRaWAN: problem and end-device-based solutions", *Sensors*, Vol. 21 No. 19, p. 6478, doi: 10.3390/s21196478.

Duvignau, R., Gulisano, V., Papatriantafilou, M. and Savic, V. (2019), "Streaming piecewise linear approximation for efficient data management in edge computing", *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*. Association for Computing Machinery, New York, NY, pp. 593-596.

Farnell (2021), "2021 Global IoT trends report", available at: https://uk.farnell.com/iot-trends-2021 (accessed 6 October 2021).

Giorgi, G. (2017), "A combined approach for real-time data compression in wireless body sensor networks", *IEEE Sensors Journal*, Vol. 17 No. 18, pp. 6129-6135, doi: 10.1109/JSEN.2017.2736249.

Harrison, D., Burmester, D., Seah, W. and Rayudu, R. (2016), "Busting myths of energy models for wireless sensor networks", *Electronics Letters*, Vol. 52 No. 16, pp. 1412-1414, available at: https://doi.org/10.1049/el.2016.1591

Hung, N.Q.V., Jeung, H. and Aberer, K. (2013), "An evaluation of model-based approaches to sensor data compression", *IEEE Transactions on Knowledge and Data Engineering*, Vol. 25 No. 11, pp. 2434-2447, doi: 10.1109/TKDE.2012.237.

Jawad, H.M., Nordin, R., Gharghan, S.K., Jawad, A.M. and Ismail, M. (2017), "Energy-efficient wireless sensor networks for precision agriculture: a review", *Sensors*, Vol. 17 No. 8, p. 1781, doi: 10.3390/s17081781.

Klus, L., Klus, R., Lohan, E.S., Granell, C., Talvitie, T., Valkama, M. and Nurmi, J. (2021), "Direct lightweight temporal compression for wearable sensor data", *IEEE Sensors Letters, 2021*, doi: 10.1109/LSENS.2021.3051809.

Lavric, A. and Popa, V. (2018), "Performance evaluation of LoRaWAN communication scalability in large-scale wireless sensor networks", *Wireless Communications and Mobile Computing*, Vol. 2018, pp. 1-9, doi: 10.1155/2018/6730719.

Li, Q., Lu, D., Zheng, J., Jiao, S., Luo, L., Wang, C. and Xu, W. (2017), "Li+-desolvation dictating Lithium-Ion battery's low-temperature performances", *ACS Applied Materials & Interfaces*, Vol. 9 No. 49, pp. 42761-42768, available at: https://doi.org/10.1021/acsami.7b13887

Lin, J.W., Liao, S.W. and Leu, F.Y. (2019), "Sensor data compression using bounded error piecewise linear approximation with resolution reduction", *Energies*, Vol. 12 No. 13, p. 2523, doi: 10.3390/en12132523.

Lin, D., Wang, Q., Min, W., Xu, J. and Zhang, Z. (2021), "A survey on energy-efficient strategies in static wireless sensor networks", *ACM Transactions on Sensor Networks*, Vol. 17 No. 1, pp. 1-48, doi: 10.1145/3414315.

Li, B., Sarbishei, O., Nourani, H. and Glatard, T. (2018), "A multi-dimensional extension of the lightweight temporal compression method", *IEEE International Conference on Big Data (Big Data), Seattle, WA, USA*, pp. 2918-2923, doi: 10.1109/BigData.2018.8621946.

Lu, S., Xia, Q., Tang, X., Zhang, X., Lu, Y. and She, J. (2021), "A reliable data compression scheme in sensor-cloud systems based on edge computing", *IEEE Access*, Vol. 9, pp. 49007-49015, doi: 10.1109/ACCESS.2021.3068753.

Maudet, S., Andrieux, G., Chevillon, R. and Diouris, J. (2021), "Refined node energy consumption modeling in a LoRaWAN network", *Sensors*, Vol. 21 No. 19, p. 6398, doi: 10.3390/s21196398.

Morin, E., Maman, M., Guizzetti, R. and Duda, A. (2017), "Comparison of the device lifetime in wireless networks for the internet of things", *IEEE Access*, Vol. 5, pp. 7097-7114, doi: 10.1109/ACCESS.2017.2688279.

Parker, D., Stojanovic, M. and Yu, C. (2013), "Exploiting temporal and spatial correlation in wireless sensor networks", *2013 Asilomar Conference on Signals, Systems and Computers*, pp. 442-446, doi: 10.1109/ACSSC.2013.6810315.

Prauzek, M., Konecny, J., Borova, M., Janosova, K., Hlavica, J. and Musilek, P. (2018), "Energy harvesting sources, storage devices and system topologies for environmental wireless sensor networks: a review", *Sensors*, Vol. 18 No. 8, p. 2446, doi: 10.3390/s18082446.

Rehman, A., Saba, T., Kashif, M., Fati, S.M., Bahaj, S.A. and Chaudhry, H. (2022), "A revisit of internet of things technologies for monitoring and control strategies in smart agriculture", *Agronomy*, Vol. 12 No. 1, p. 127, doi: 10.3390/agronomy12010127.

Săcăleanu, D.I., Popescu, R., Manciu, I.P. and Perişoară, L.A. (2018), "Data compression in wireless sensor nodes with LoRa", *2018 10th International Conference on Electronics, Computers and Artificial Intelligence (ECAI), 2018*, pp. 1-4, doi: 10.1109/ECAI.2018.8679003.

Sarbishei, O. (2019), "Refined lightweight temporal compression for energy-efficient sensor data streaming", *IEEE 5th World Forum on Internet of Things (WF-IoT), Limerick, Ireland*, pp. 550-553, doi: 10.1109/WF-IoT.2019.8767351.

Schoellhammer, T., Greenstein, B., Osterweil, E., Wimbrow, M. and Estrin, D. (2004), "Lightweight temporal compression of microclimate datasets [wireless sensor networks]", *29th Annual IEEE International Conference on Local Computer Networks*, pp. 516-524, doi: 10.1109/LCN.2004.72.

Ullo, S.L. and Sinha, G.R. (2020), "Advances in smart environment monitoring systems using IoT and sensors", *Sensors*, Vol. 20 No. 11, p. 3113, doi: 10.3390/s20113113.

Väänänen, O. and Hämäläinen, T. (2019), "Compression methods for microclimate data based on linear approximation of sensor data", *The 19th International Conference on Next Generation Wired/Wireless Advanced Networks and Systems NEW2AN 2019, August 26 – 28, 2019, St.Petersburg, Russia*.

Väänänen, O. and Hämäläinen, T. (2020), "Sensor data stream on-line compression with linearity-based methods", *IEEE International Conference on Smart Computing (SMARTCOMP), Bologna, Italy*, pp. 220-225, doi: 10.1109/SMARTCOMP50058.2020.00049.

Väänänen, O. and Hämäläinen, T. (2021), "LoRa-based sensor node energy consumption with data compression", *2021 IEEE International Workshop on Metrology for Industry 4.0 & IoT (MetroInd4.0&IoT)*, pp. 6-11, doi: 10.1109/MetroInd4.0IoT51437.2021.9488434.

Further reading

Arduino MKR WAN 1310 (2022), available at: https://store.arduino.cc/mkr-wan-1310 (accessed 6 October 2021).

Digita (2022), "IoT LoRaWAN network coverage in Finland", available at: www.digita.fi/en/iot-lorawan-network-coverage-map/ (accessed 6 October 2021).

DHT22 temperature and humidity sensor (2022), available at: www.sparkfun.com/datasheets/Sensors/Temperature/DHT22.pdf (accessed 6 October 2021).

LoRa Alliance (2022a), "What is LoRaWAN specification", available at: https://lora-alliance.org/about-lorawan/ (accessed 6 October 2021).

LoRa Alliance (2022b), "RP002-1.0.0 LoRaWAN regional parameters", available at: https://lora-alliance.org/wp-content/uploads/2019/11/rp_2-1.0.0_final_release.pdf (accessed 6 October 2021).

Semtech, (2022) "What are LoRa and loRaWAN", available at: https://lora-developers.semtech.com/documentation/tech-papers-and-guides/lora-and-lorawan (accessed 6 October 2021).

Semtech, (2022) "What is an adaptive data rate", available at: https://lora-developers.semtech.com/documentation/tech-papers-and-guides/understanding-adr/ (accessed 6 October 2021).

Corresponding author

Olli Väänänen can be contacted at: olli.vaananen@jamk.fi

VIII

LINEARITY-BASED SENSOR DATA ONLINE COMPRESSION METHODS FOR ENVIRONMENTAL APPLICATIONS

by

Olli Väänänen & Timo Hämäläinen, 2023

Proceedings of the 6th Conference on Cloud and Internet of Things (CIoT)

https://doi.org/10.1109/CIoT57267.2023.10084892

Linearity-based Sensor Data Online Compression Methods for Environmental Applications

Olli Väänänen
School of Technology
JAMK University of Applied Sciences
Jyväskylä, Finland
ORCID: 0000-0002-7211-7668

Timo Hämäläinen
Faculty of Information Technology
University of Jyväskylä
Jyväskylä, Finland
ORCID: 0000-0002-4168-9102

Abstract— **Environmental monitoring is a typical Internet of Things (IoT) application. Environmental monitoring plays a significant role, for example, in smart farming and smart city applications. Environmental magnitudes are usually measured using wireless sensor nodes, which are often battery-powered, and the number of sensing nodes can be large. One effective method for reducing the energy consumption of a sensor node is to use data compression to reduce the amount of data required for transmission via a wireless connection. Compressing the sensor data means fewer transmission periods, and thus, lower energy consumption. Compression methods should be effective for compressing environmental magnitudes and be computationally light to be suitable for constrained sensor nodes. A compression algorithm should be able to compress an online data stream. In this paper, we review some compression algorithms suitable for environmental monitoring and present two new versions of those algorithms. The algorithms were evaluated, tested, and compared. The main parameters used for the comparisons were compression ratio, root mean square error, and inherent latency. The simulation results obtained using real datasets demonstrate that simple linearity-based compression algorithms are effective and suitable for compressing environmental data. Two new compression algorithm versions proved to be effective for compressing sensor data with reasonable compression quality and predictable inherent latency.**

Keywords—compression algorithm, data compression, edge computing, Internet of Things, sensor data

I. INTRODUCTION

In environmental monitoring the wireless sensor nodes can be located in wide area and the number of nodes can be large. Wireless sensor nodes are often battery powered and replacing empty batteries can be costly, as the nodes may be located in a wide area and thus require manpower to complete the replacement. Thus, minimizing the energy consumption and lengthening the lifetime of the sensor node can be a cost-effective solution. Compressing the sensor data stream in online mode can reduce the transmission periods needed via wireless connection. Wireless transmission is known to be the most energy consuming operation in wireless sensor node. Between sensing and transmission phases the node can be in sleep mode.

In this paper, some basic linearity-based compression algorithms are presented, and two new versions are developed and evaluated. The algorithms are compared to each other by compression ratio, root mean square error and algorithm inherent latency. The remainder of this paper is organized as follows: present algorithms are presented in Section II. New algorithm versions are presented in Section III. Compression algorithms inherent latency considerations are in Section IV. Algorithms' ability to compress environmental datasets is in Section V. Section VI is the summary of the results and finally section VII presents the conclusions.

II. LINEARITY-BASED TEMPORAL COMPRESSION METHODS FOR SENSOR DATA

A. Linear Regression based Temporal Compression

Linear Regression based Temporal Compression (LRbTC) algorithm is based on basic linear regression and it is designed to compress the sensor data in online mode. Thus, the dataset is not already available, but as a function of time, the new data values come in sequence with constant frequency, and the LRbTC algorithm compresses the data value by value [1]. The algorithm waits until the first N measurement values are available and then the regression line described by the N values is calculated. These N values are expected to predict the future values of the sensor data. If the data behave linearly, then the regression line can predict the consecutive measurement values with a certain error bound (ε) allowed. N has a minimum of three values, but four and five values were also tested in [1].

Calculating the linear regression of N values yields the linear line that best fits the N values used. The calculation of the regression line is based on the least-squares method, which minimizes the sum of squares of the deviation between data points. After calculating the regression line of the first N measured values, the algorithm stores and/or sends the starting point of the regression line. The inherent latency of the algorithm is $(N\text{-}1)\Delta t$ at this point when the regression line is calculated. Then when a new measured value is achieved after one measurement interval (Δt), the algorithm compares the value to the regression line value at that timestamp. If the value is within one error bound from the line, the algorithm waits for the next measured value and makes a new comparison. When the new measured value falls off from the regression line prediction (differs more than one error bound from the line), the algorithm stores and/or sends the linear regression line value in one timestamp before (last timestamp when the measured value was still within one error bound from the line) as an end point of the linear segment. Then the algorithm waits for N-1 new measured values (because one value is already available; the one that was in more than one error-bound distance from the line and ended the linear segment). After calculating the new regression line, the algorithm stores and/or sends the new regression line starting point.

The weakness of this basic form of LRbTC is the possibility that the values used to calculate the regression line may differ more than one error bound (ε) from the regression line. Thus, the values derived from the compressed dataset may differ more than one error bound from the original values and therefore the error bound requirement is not guaranteed [1]. A modified version was developed to solve the aforementioned weakness in the basic version. The modified LRbTC (M-LRbTC) is analogous to LRbTC, except that the comparison between the raw values and regression line is also made for the values used to calculate the regression line [1]. If the difference between the calculated regression line and the raw value or values is larger than the error bound, then the first two raw values are retained (stored/sent), and a new regression line is calculated when the next two new values are available. The algorithm works if $N = 3$ or more [1],[2].

The output of the algorithm can be presented as a compressed dataset, $M\text{-}LRbTC(S) = <(c_1, \tau_1), (c_2, \tau_2),\ldots, (c_k, \tau_k)>$. The compressed data values (c_i, τ_i) are either the starting points or end points of the linear segments, or the raw data values if the difference has been too big between the line and the raw values that were used to calculate the line.

One drawback of M-LRbTC (and in basic LRbTC) is that two data pairs are required for each linear segment; starting point and end point. Another drawback of this algorithm is the inherent latency. Latency is predicted when the new regression line is calculated, and it is determined by N. After the regression line is calculated, the latency is not known and is not predictable. The better the regression line predicts future values, the longer is the latency. The drawback of unpredictable latency can be overcome by sending the regression line parameters a (slope) and b (base) with the line starting point timestamp. Thus, in this case, the compressed data can be represented as $LRbTC(S) = <(a_1,b_1,\tau_1),(c_2,\tau_2),(a_3,b_3,\tau_3),(c_4,\tau_4),\ldots,(a_{k-1},b_{k-1},\tau_{k-1}),(c_k,\tau_k)>$. The latency in this version is the N–1 measurement intervals when calculating the regression line. When the line parameters are sent, the receiver knows that the values follow the line with one measurement interval latency until the line-end parameters (c_i and τ_i) are received [2].

B. *Real-Time Linear Regression Based Temporal Compression*

RT-LRbTC was presented in [2] as a modification of M-LRbTC to achieve a predictable and shorter inherent latency. RT-LRbTC uses already available sensor values to calculate a new regression line. Thus, the inherent latency is only one measurement interval, Δt. A flowchart of the RT-LRbTC is presented in Fig. 1. Initially, the algorithm works as an M-LRbTC, and the inherent latency is N–1 measurement intervals long, which is $(N\text{-}1)\Delta t$. In step 6, the algorithm stores and/or sends the regression line parameters and the timestamp of the regression line starting point instead of the line starting point value. In step 7, when the algorithm is in the linear section, the inherent latency is one measurement interval (Δt). If the difference in step 8 is larger than the error bound allowed, the new regression line is calculated in step 9, but the line is calculated using the already available values. The compressed dataset is presented as $RT\text{-}LRbTC(S) = <(a_1,b_1,\tau_1), (a_2,b_2,\tau_2),\ldots, (a_k,b_k,\tau_k)>$, where a_i and b_i are the regression line parameters and τ_i is the line beginning timestamp. When the new line parameters with the timestamp are received, it is known that the previous line ended one measurement interval earlier. Thus, the inherent latency of the algorithm was described by the measurement frequency [2].

The compression efficiency is dependent on the data characteristics, and in most cases, RT-LRbTC has a lower inherent compression ratio (CR) than M-LRbTC [2]. As RT-LRbTC generally has a lower CR, it means that there are more linear regression lines after compression with RT-LRbTC than with M-LRbTC. An advantage of RT-LRbTC is that the line parameters must be sent only once for each linear section, thus resulting in a better compression ratio compared to M-LRbTC [2]. In the basic version of M-LRbTC, the starting and endpoint values with timestamps need to be sent for each linear segment. RT-LRbTC benefits from the fact that, compared to $(N\text{-}1)\Delta t$ latency with M-LRbTC, there is no inherent latency when the new regression line is calculated.

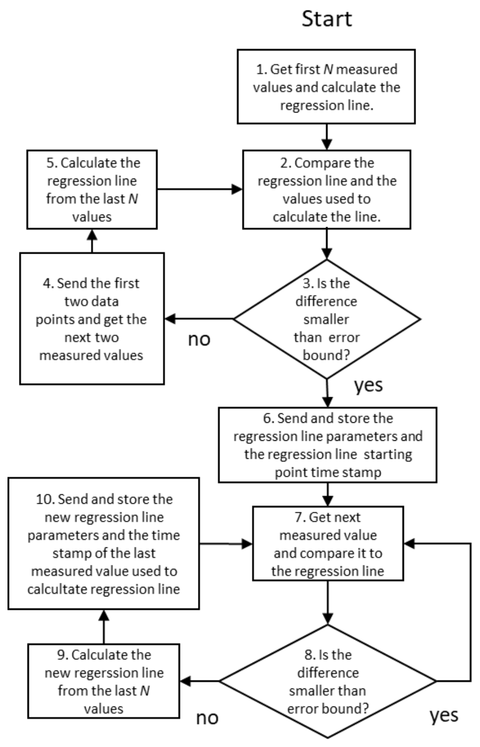

Start
1. Get first N measured values and calculate the regression line.
5. Calculate the regression line from the last N values
2. Compare the regression line and the values used to calculate the line.
4. Send the first two data points and get the next two measured values
no
3. Is the difference smaller than error bound?
yes
6. Send and store the regression line parameters and the regression line starting point time stamp
10. Send and store the new regression line parameters and the time stamp of the last measured value used to calcultate regression line
7. Get next measured value and compare it to the regression line
9. Calculate the new regerssion line from the last N values
8. Is the difference smaller than error bound?
no
yes

Fig. 1. RT-LRbTC flowchart

C. *Lightweight Temporal Compression*

LTC is a well-known and simple compression algorithm. It was first presented in [3], but a similar algorithm, called Fan, was actually presented before in [4] for electrocardiogram (ECG) data. LTC is a very effective compression algorithm, especially for environmental data that behave rather linearly when the observation time window is short. The compression ratio depends on the data characteristics and error bound used. LTC can achieve a compression ratio as high as 20 when compressing the environmental temperature data with a 10-minute measurement interval and 1.0 °C error bound [1].

The LTC has an unpredictable latency and is dependent on each linear section length. Thus, the higher the CR, the longer the latency. If the data behave very linearly, a long latency is derived. When the new linear segment starts, the starting point is known, but the direction of the following values remains unknown until the linear segment ends, and the end point is stored in the compressed dataset. Due to unpredictable latency, LTC is not suited for real-time applications. In this study, LTC has been used as a comparison for the other algorithms.

Some slight variations of the original LTC algorithm have been developed. In [5], a modification of the LTC algorithm was used. Other variations include Adaptive Lightweight Temporal Compression [6], Refined Lightweight Temporal Compression (RLTC) [7], multidimensional extension of the LTC method [8], Direct Lightweight Temporal Compression (DLTC) [9] and DFan [10]. These modified versions were developed either to minimize the data reconstruction error or improve the compression efficiency.

III. New Versions of the RT-LRbTC Algorithm

In this study, two new versions of the RT-LRbTC algorithm were developed. One variation of the basic RT-LRbTC is the RT-LRbTC with 2Δ*t* inherent latency in the new regression line calculation (RT-LRbTC-2Δ*t*). This version is the same as the basic RT-LRbTC (with $N = 3$), but the values used to calculate the new regression line are the last point in the previous linear section, the first value that fell off from the previous section, and one new measurement value. The need to wait for one new value adds the inherent latency to 2Δ*t* when the information of the previous line ends, and new line parameters are obtained. In Fig. 1, this means that in step 9, there is a need to wait for one measurement interval and then calculate the new regression line from the last three values. This new version is a compromise between M-LRbTC and RT-LRbTC; having the inherent latency between those two algorithms.

Another variation of the basic RT-LRbTC and RT-LRbTC-2Δ*t* is the use of weighted linear regression instead of ordinary linear regression. This version of the algorithm is called RT-WLRbTC (Real-Time Weighted Linear Regression-based Temporal Compression) with 2Δ*t* inherent latency (RT-WLRbTC-2Δ*t*). Weighted linear regression (or weighted least-squares, *WLS*) is used in statistics and data analysis instead of simple linear regression when the variation in the samples (values) is not constant. This heterogeneous nature of the values can be addressed by *WLS* using heterogeneous weights w_i in the normal linear regression equations [11]. The sum of squares of the deviation with weights is [12]:

$$S_w = \sum_{i=1}^{n} w_i[y_i - (ax_i + b)]^2 \tag{1}$$

The *WLS* estimates of *a* and *b* (line parameters) were obtained by minimizing (1). In this study, the idea of *WLS* is utilized as some of the samples are used more than once while calculating the linear regression to give more weight to a specific individual value (single values used twice means double weight). RT-WLRbTC-2Δ*t* is similar to RT-LRbTC-2Δ*t*, except that the last value used to calculate the linear regression line is used twice, thus having a weight value of 2 compared to 1 for the other two values. Hence, the computational complexity is similar to that of the M-LRbTC with $N = 4$. The main idea behind using this type of weighted linear regression is that when the linear section ends and the new regression line is calculated, it is expected that the direction of the values is changing. Thus, the latest value is expected to predict future values better than other values used to calculate the regression line, and thus the latest value has a larger effect on the linear regression line calculation. Different versions of weighted linear regression can be developed and used; however, only this one example was tested in this study.

IV. Temporal Compression Methods' Inherent Latency

In Table I, all the presented methods are compared in the order of the algorithm's inherent latency. This comparison does not consider the latency caused by the computational time. Because the measurement interval in typical environmental applications is rather long (minutes or even hours in some cases), the time needed for calculations is negligible, even with the most constrained end devices.

LTC and M-LRbTC (basic version) are not well suited for compressing sensor data value by value in the online mode. LTC has unpredictable inherent latency, which is dependent on how well the values fit in the linear section. When the linear section ends, the endpoint and the new linear section starting point are at the same point. That information is achieved in one measurement interval after the linear section ends. The basic version of the M-LRbTC has an inherent latency of (*N*-1)Δ*t* in the beginning, when the algorithm waits until there are *N* measurement values to be used to calculate the regression line. The starting point line value is stored and/or sent, but it is not known in which direction the values are moving since this until the line ends and the end point value is stored and/or sent. If the linear regression line parameters are sent, then the inherent latency is constant Δ*t* in the linear section. Only M-LRbTC[b] (Table I) and the three RT-LRbTC-based algorithms have fixed and predictable latencies. Of the presented algorithms, RT-LRbTC has the shortest overall latency, Δ*t*, in the linear section, and no latency in calculating a new line. The new versions have double inherent latency (2Δ*t*) and no latency in calculating a new regression line.

V. Linearity-based Methods' Compression Quality and Ability to Compress Environmental Datasets

It was demonstrated in [13] that the average absolute change between consecutive measurements (*AC*) can be used to predict the selected linearity-based algorithm's ability to compress datasets (compression ratio, *CR*). *AC* is defined as:

$$AC = \frac{\sum_{i=1}^{n-1}|x_{i+1} - x_i|}{n-1} \tag{2}$$

Additionally, the standard deviation (*SD*) of the change between consecutive measurements can also be used to predict the *CR*, but the *AC* provides a better estimation [13]. *SD* is defined as (3):

$$SD = \sqrt{\frac{\sum_{i=1}^{n-1}\left((x_{i+1} - x_i) - (\overline{x_{i+1} - x_i})\right)^2}{n-2}} \tag{3}$$

where,

$$(\overline{x_{i+1} - x_i}) = \frac{1}{n-1}\sum_{i=1}^{n-1}(x_{i+1} - x_i) \tag{4}$$

TABLE I. Compression Algorithms' Latencies in Different Phases of Compression

Phase of the Compression	Compression Algorithm					
	LTC	***M-LRbTC*[a]**	***M-LRbTC*[b]**	***RT-LRbTC***	***RT-LRbTC-2Δt***	***RT-WLRbTC-2Δt***
At the beginning	0	(*N*-1)Δ*t*	(*N*-1)Δ*t*	(*N*-1)Δ*t*	(*N*-1)Δ*t*	(*N*-1)Δ*t*
In linear section	Length of the linear section	Length of the linear section	Δ*t*	Δ*t*	2Δ*t*	2Δ*t*
Calculating new line	NA	(*N*-1)Δ*t*	(*N*-1)Δ*t*	0	0	0

[a]M-LRbTC: The linear regression line start and endpoint values are sent.
[b]M-LRbTC: The linear regression line parameters are sent with the starting timestamp. The endpoint of the linear line is sent when the first value falls off from the linear segment.

The suitability of a compression algorithm depends on the characteristics of the sensor data. Many environmental magnitudes are quasi-linear in a short time window, and some compression algorithms are more suitable and effective for this type of linearly behaving data than for other types of data. In this study, the *AC* and *SD* values of the datasets were used to compare the datasets' characteristics and to estimate the compression algorithms' ability to compress those datasets effectively.

The common parameters used to compare different compression algorithms are the compression ratio (*CR*) and the root mean square error (*RMSE*). The compression ratio is calculated as *CR = (original data)/(compressed data)* and the root mean square error [14]:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(v_i - c_i)^2} \quad (5)$$

where v_i is the raw data value and c_i is the corresponding value derived from the compressed dataset. The compression ratio indicates how effectively the algorithm reduces the size of the original data and is most widely used to express the efficiency of the compression algorithm. The *RMSE* indicates the compression quality. Because the methods presented in this paper are lossy, the reconstructed data differ from the original data. Thus, some information is lost. The *RMSE* provides information about how much the data reconstructed from the compressed data differs from the original data. The smaller the *RMSE* value, the smaller is the deviation from the original data.

A. The Selected Compression Algorithms' Efficiency to Compress Real Environmental Datasets

LTC, M-LRbTC, and three versions of RT-LRbTC algorithms were tested with real datasets and compared with each other. All tested algorithms are suitable for constrained IoT devices. In [15] and [16], LTC and RT-LRbTC (with *N* values 3 and 4) were implemented on a LoRa sensor node and no significant energy consumption from the algorithm calculations was discovered. It was discovered that using these lossy compression algorithms led to significant reduction on energy consumption and thus it can be effective solution for lengthening the battery powered sensor node lifetime. The energy saving was due to reduction of the wireless transmitting periods [15], [16].

In this study, the M-LRbTC algorithm was tested using *N* values of 3, 4, and 5. The RT-LRbTC algorithm was tested with the original version and two newly developed versions: RT-LRbTC-2Δ*t* and RT-WLRbTC-2Δ*t*. All algorithms were programmed and tested using MATLAB. The datasets were already available, and thus, this situation does not correspond to the situation for compressing the real-time sensor data stream. This testing situation demonstrates how the algorithm would have compressed the data when the dataset was collected. As the temperature at the same geological location behaves rather similarly year by year, this testing indicates how the algorithm could possibly compress the data in that situation. Similar behavior at certain geological locations for other environmental magnitudes is also expected.

The environmental magnitudes were temperature, air pressure and wind speed. The datasets used were obtained from the Finnish Meteorological Institute's open data service [17]. The datasets tested were Salla Naruska measurement station data and Hanko Tulliniemi measurement station data. All datasets were for the full year 2019 with 10-minute measurement intervals. The temperature was measured in degrees Celsius with 0.1 degrees resolution, air pressure was measured in hPa with 0.1 hPa resolution and wind speed was measured in 10-minute average value with 0.1 m/s resolution.

The Salla Naruska measurement station is in the eastern part of Finnish Lapland. It is one of the coldest locations in Finland. The Hanko Tulliniemi measurement station is in the southernmost part of Finland, 100 m from the sea. These two locations have very different climates. The Salla Naruska dataset was also used in [2] but the Hanko Tulliniemi dataset is a new experiment. In [2] LTC, M-LRbTC, and RT-LRbTC compression ratios with different error bound values were tested using the Salla Naruska dataset (temperature, air pressure, and wind speed). In this study, the same compression ratio simulations for the Salla Naruska datasets were repeated, but the RT-LRbTC algorithm's MATLAB version was further developed to give the *CR* value calculated as only line parameters are needed for each linear section and possible single values (not included in any linear segment) were taken into account. In [2] the same situation was achieved by doubling the *CR* values achieved from the compressed datasets, which included the start and end points of the linear segments. The method in [2] gives the same values for *CR* when the *CR* is high but gives erroneous results if the compressed dataset also includes single values that do not belong to linear segments. In this study, the *RMSE* values for each algorithm were calculated using different error-bound values. In addition, the *CR* and *RMSE* values were compared, and two new algorithms were tested for each environmental dataset.

1) Temperature Datasets

The Salla Naruska dataset contains 52 463 values and the Hanko Tulliniemi dataset 51 961 measurement values with 10-minute measurement intervals. For the full year, the dataset should contain 52 560 values but both datasets have some periods with missing values. The individual missing values and short periods with missing values in the original datasets were linearly interpolated. Longer periods with missing values were removed from the dataset. The temperature values in the Salla Naruska dataset varied between -37.2 °C and +30 °C. The temperature values in the Hanko Tulliniemi dataset varied between -12.3 °C and +27.6 °C.

Table II presents a comparison between these two temperature datasets. Both the *AC* and *SD* values were higher for the Salla Naruska temperature dataset than for the Hanko Tulliniemi dataset. It indicates a lower compression ratio for Salla Naruska dataset.

TABLE II. TEMPERATURE DATASETS

	Salla Naruska	***Hanko Tulliniemi***
Number of values	52 463	51 961
Average change (AC)	0.2077	0.1282
Standard deviation (SD)	0.3473	0.2221

The results achieved in [13] can be used to estimate the compression ratios with error bound of 0.5 °C for the LTC and M-LRbTC algorithms when the *AC* and *SD* values are known. With *AC* values from Table II the estimation gives *CR* = 10.4 (Salla Naruska dataset) and *CR* = 15.8 (Hanko Tulliniemi dataset) for LTC when the error bound is 0.5 °C. The real *CR* values achieved in this paper were 10.2 and 14.1 respectively. For M-LRbTC ($N = 3$) the estimation gives *CR* values 4.0 and 5.8 compared to the results achieved here which are 4.0 and 5.4 respectively. The estimations from [13] are close to the real compression ratios achieved in this study.

Fig. 2 shows the performance results of the different compression algorithms. Fig. 2 (a) and (b) show the compression ratios as a function of error bound (from 0.1 to 1.0 degrees Celsius). The LTC has a superior compression ratio compared to the linear regression-based algorithms. The difference between the different linear-regression-based algorithms is not very large. The compression performance difference between M-LRbTC $N = 3$ and $N = 4$ was slightly larger than that between $N = 4$ and $N = 5$. RT-LRbTC has the lowest compression ratio in terms of linear lines (start and end points), but it benefits from the fact that only one transmission period is required for each linear line. In Fig. 2 (a) and (b), for different RT-LRbTC (RT-LRbTC, RT-LRbTC-2Δt, and RT-WLRbTC-2Δt) algorithms, the compression ratio values are presented as only the line parameters are stored/sent at the beginning of each linear line.

The new variations in RT-LRbTC (RT-LRbTC-2Δt and RT-WLRbTC-2Δt) have a slightly better compression performance than the basic RT-LRbTC. This can be observed in Fig. 2 (a) and (b) for the compression ratio. In general, all the tested compression algorithms performed better for the Hanko Tulliniemi dataset than for the Salla Naruska dataset, as indicated by the *AC* and *SD* values in Table II. A typical error bound for temperature data in many applications can be approximately ± 0.5 degrees Celsius. One possibility to choose the error bound is to use the margin of error of the temperature sensor, which can be found in the sensor's data sheet [3].

In Fig. 2 (c) and (d) the *RMSE* values with different error bounds can be seen. The LTC has the largest *RMSE* values, which means that the drawback of a better compression ratio is lower compression quality. Among linear-regression-based methods, the results are similar between them. As RT-LRbTC has a higher compression ratio, it also has a lower compression quality than the M-LRbTC methods. The advantage of RT-LRbTC is its shorter latency and moderate compression ratio, but its drawback is the larger average reconstruction error after compression than that of M-LRbTC.

The compression quality measurements (*RMSE*) are similar level with all linear regression-based algorithms, as can be seen in Fig. 2 (c) and (d). RT-WLRbTC-2Δt does not show any better performance than RT-LRbTC-2Δt in any measurements for these temperature datasets. In Fig. 2 (e) and (f), the *RMSE* values are compared in terms of the compression ratio. In this comparison, the LTC has the best performance, even though the previous comparisons in (c) and (d) indicate a lower compression quality. The LTC benefits from its superior compression ratio compared to the other tested methods.

2) Air Pressure Datasets

The same algorithms were tested for the air pressure datasets. The datasets are listed in Table III. The characteristics of both datasets were very similar, indicating

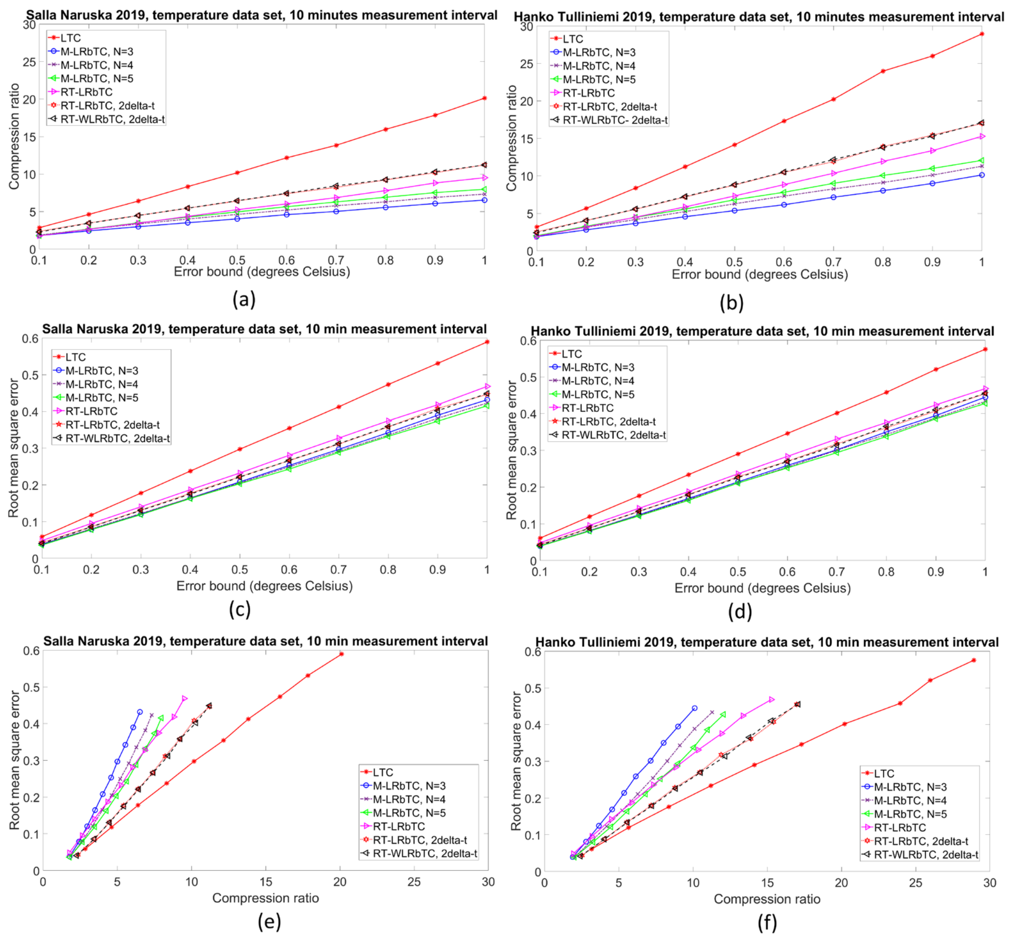


Fig. 2. Compression algorithms' performance for temperature datasets

very similar behavior for both datasets. There was only a small difference between the *AC* and *SD* values. The air pressure values in the Salla Naruska dataset varied between 967.2 hPa and 1039.5 hPa. In Hanko Tulliniemi dataset the values were between 970.5 hPa and 1043.1 hPa.

TABLE III. AIR PRESSURE DATASETS

	Salla Naruska	*Hanko Tulliniemi*
Number of values	52 463	51 961
Average change (AC)	0.0856	0.0836
Standard deviation (SD)	0.1234	0.1249

In Fig. 3, all the results for compressing the two tested air pressure datasets are shown. Both datasets are full-year datasets with 10-minute measurement intervals.

As shown in Fig. 3 (a) and (b), the compression ratios were significantly higher than for the temperature data (Fig. 2). These are not directly comparable, but as both magnitudes are measured with 0.1 (degree and hPa) resolution, the error bounds of 0.1 – 1.0 can thus be compared to each other sufficiently. In general, the air pressure data changes rather slowly; thus, it is well suited for linearity-based compression algorithms. The *AC* and *SD* values in Table III are significantly lower than those for the temperature datasets in Table II, thus indicating better compression performance. From the results obtained in [13] it is possible to estimate the compression ratios for the LTC and M-LRbTC algorithms. The data from [13] estimated the compression ratio with 0.5 hPa error bound for LTC to be 27.9 for the Salla Naruska dataset and 28.3 for the Hanko Tulliniemi dataset. The results from MATLAB simulations provided *CR* values 29.6 for the Salla Naruska dataset and 29.8 for the Hanko Tulliniemi dataset. These results are very close to the estimations.

The results for RT-LRbTC-2Δ*t* and RT-WLRbTC-2Δ*t* were very close to the RT-LRbTC values in terms of the compression ratio (Fig. 3 (a) and (b)). All linear regression-based algorithms are very close to each other in terms of the quality metrics (*RMSE*), as can be seen in Fig. 3 (c) and (d). Again, the LTC has a superior compression ratio but also the largest average construction error. The difference between the various linear regression-based algorithms is small in terms of the quality metrics and compression ratio. The three different RT-LRbTC algorithms presented the best compression performance among the linear regression-based algorithms. In Fig. 3 (e) and (f), the *RMSE* values of the different algorithms are compared in terms of compression ratio. The performance order between the algorithms is quite similar to that of the temperature data, as seen previously.

3) *Wind Speed Datasets*

The two wind speed datasets have different characteristics. As the measurement stations are located in very different places, one close to the sea and the other in Lapland in the lowland, the wind conditions are different. The *AC* and *SD* values are listed in Table IV. The *AC* and *SD* values were higher for the Hanko Tulliniemi dataset, which indicates lower compression ratios for that dataset. The wind speed was measured as 10-minute average values. Otherwise, the wind speed is gusty if measured in instantaneous values, and thus,

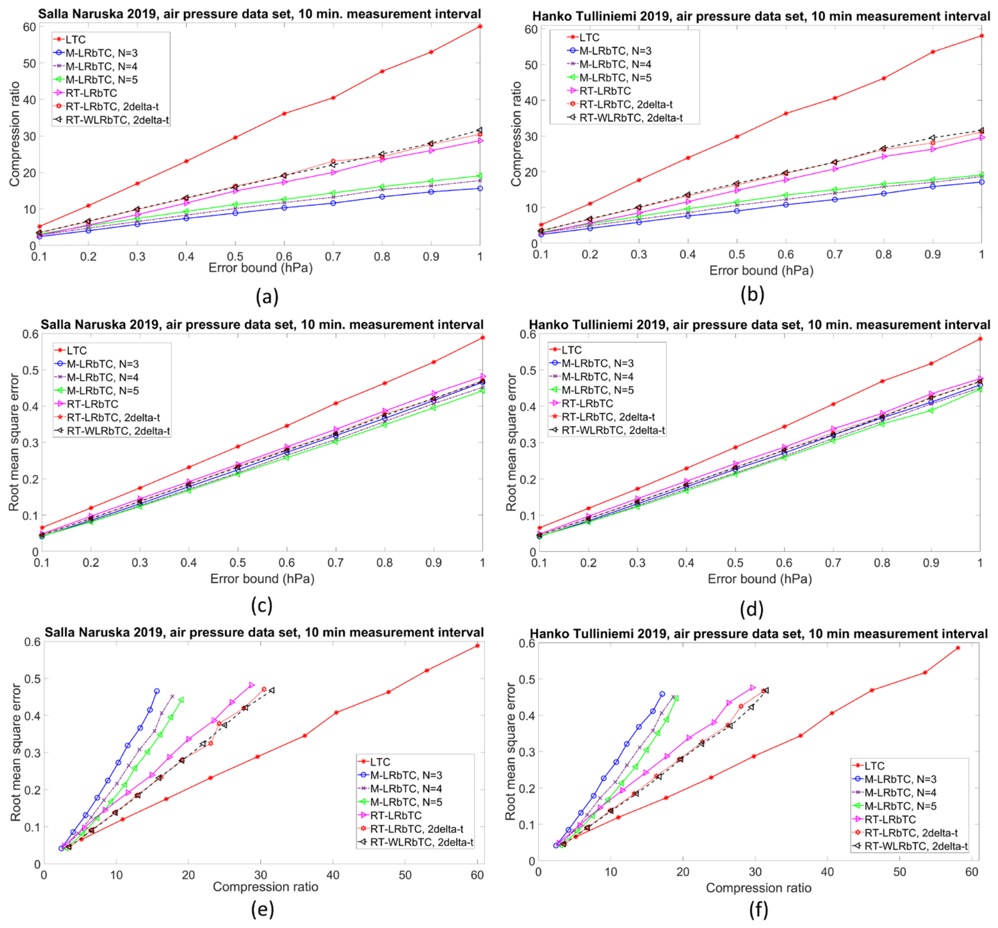


Fig. 3. Compression algorithms' performance for air pressure datasets

it does not exhibit linear behavior. The values in the Salla Naruska dataset were 0 m/s - 9.3 m/s. The values in the Hanko Tulliniemi dataset were 0 m/s - 21.8 m/s.

TABLE IV. WIND SPEED DATASETS

	Salla Naruska	*Hanko Tulliniemi*
Number of values	52 463	51 961
Average change (AC)	0.2844	0.4188
Standard deviation (SD)	0.4125	0.5917

The compression ratios for the Salla Naruska dataset are significantly higher than those for the Hanko Tulliniemi dataset, as can be seen in Fig. 4 (a) and (b). For example, LTC with error bound 0.5 m/s achieves *CR* = 5.54 for Salla Naruska dataset and *CR* = 3.98 for Hanko Tulliniemi dataset. The estimations obtained from the data in [13] provide the compression ratio estimations of 5.5 for the Salla Naruska dataset and 3.6 for Hanko Tulliniemi dataset for LTC with the same error bound. These estimations were close to the actual compression ratios achieved. The estimations for M-LRbTC, *N* = 3 give *CR* = 2.7 for Salla Naruska data and *CR* = 2.1 for Hanko Tulliniemi data. The MATLAB simulation yielded the same values. The compression ratios are clearly higher for the Salla Naruska dataset than for the Hanko Tulliniemi dataset, as indicated by the *AC* and *SD* values of the datasets. RT-LRbTC-2Δ*t* and RT-WLRbTC-2Δ*t* exhibited the higher *CR* values compared with the other linear regression-based methods.

The *RMSE* results are shown in Fig. 4 (c) and (d). The performance results with these quality measurements were at the same level for each linear-regression-based method. The differences were very limited. LTC has a significantly lower compression quality (higher *RMSE*), as can be seen in Fig. 4 (c) and (d). The different characteristics of the datasets did not appear to affect the quality metrics. The *RMSE* results were very similar for both the datasets. When comparing the *RMSE* values as a function of compression ratio (Fig. 4 (e) and (f)), the performances of the different algorithms differ less from each other than with temperature and air pressure datasets.

The new algorithms, RT-LRbTC-2Δ*t* and RT-WLRbTC-2Δ*t*, showed better overall performance than the other linear regression-based algorithms for wind speed datasets. Thus, these new algorithms are potential methods for compressing wind-speed data if the additional inherent latency is acceptable.

VI. SUMMARY OF THE RESULTS

LTC had the best compression efficiency for all datasets, but at the same time, it had the largest *RMSE* values with a certain error bound. The M-LRbTC algorithm benefits from increasing the *N* value from 3 to 4 or 5; however, it makes the algorithm more complex and increases the inherent latency. The new versions (RT-LRbTC-2Δ*t* and RT-WLRbTC-2Δ*t*) have slightly better compression performance than RT-LRbTC. These new algorithms have the same benefit as the RT-LRbTC in that only one transmitting period is needed for each linear segment. The weighted linear regression did not improve the compression performance compared with RT-LRbTC-2Δ*t*. Among the tested algorithms, RT-LRbTC is the best algorithm if a short inherent latency is required, as shown in Table I. If the predicted latency is required, then different versions of RT-LRbTC or M-LRbTC (when regression line parameters are sent) are suitable algorithms. LTC has superior compression performance, but unpredictable latency; thus, it

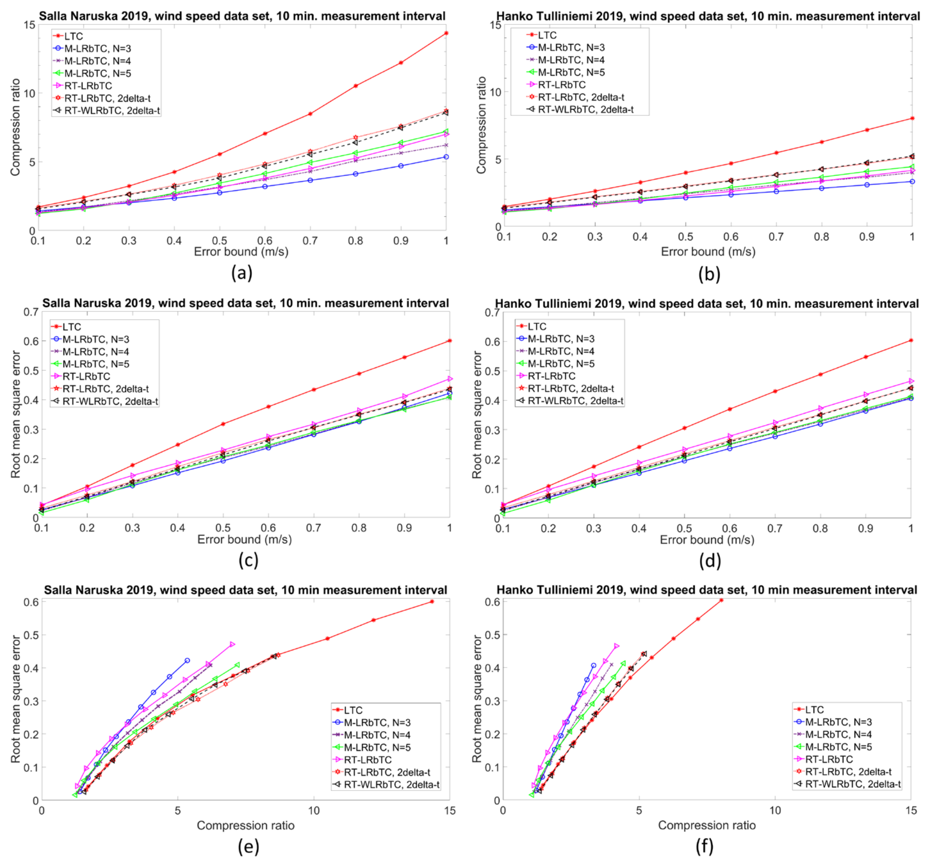


Fig. 4. Compression algorithms' performance for wind speed datasets

is not well suited for online data stream compression or near real-time applications. With a certain error bound, the LTC had the largest *RMSE* values, and thus had the lowest compression quality. Linear regression-based algorithms have very similar *RMSE* values, and thus have a similar performance in terms of compression quality. The two new algorithms proved to be suitable for compressing online environmental sensor data streams with a predictable but slightly longer inherent latency than the original RT-LRbTC.

VII. Conclusions

Different linearity-based sensor data compression algorithms were presented, and their efficiency to compress different environmental microclimate datasets was tested with real datasets. The environmental magnitudes tested were temperature, air pressure and wind speed. Algorithms were compared using the compression ratio (*CR*) and quality measurements as the root mean square error (*RMSE*). Inherent latency was used as a feature to compare the ability of different algorithms to compress data in the online mode.

The datasets used were real datasets acquired from the Finnish Meteorological Institute's Open Data Service. The datasets were used retrospectively and not in the online mode. Thus, the dataset characteristics can be used to compare the compression results of different compression algorithms. The datasets were measurement values from the year 2019, but the characteristics of those datasets can be used to predict the performance of different algorithms in the future in that measurement station and measurement setup. In different places (microclimates), different magnitudes have a behavior typical for that place. Thus, the available dataset for that specific location can be used to predict the characteristics of future datasets and thus predict the performance of the different algorithms.

Two new versions of the RT-LRbTC are presented in this paper. These new versions were tested and compared with other algorithms. LTC has a superior compression ratio compared to other methods but as a disadvantage, LTC has unpredictable inherent latency. The quality measurements demonstrate that the LTC also has the largest reconstruction error when a certain error bound is used. The quality measurements between the different versions of M-LRbTC and RT-LRbTC are very close to each other.

The new versions of the RT-LRbTC algorithm (RT-LRbTC-2Δt and RT-WLRbTC-2Δt) present better compression ratios than the basic version of the RT-LRbTC, but at the cost of a larger inherent latency. In addition, the quality measurements are slightly better for the new versions. Using weighted linear regression to calculate the regression line, weighting the last value used to calculate the line, did not yield any better results than the regular linear regression line. Thus, it only adds the computational complexity of the compression algorithm, without any significant benefits.

All the presented and tested linearity-based compression algorithms are very simple and thus suitable for use in constrained wireless sensor nodes to reduce the overall energy consumption and extend the battery lifetime. These compression methods can significantly reduce the number of wireless transmission periods and, consequently, lower the energy consumption of the sensor node.

References

[1] O. Väänänen and T. Hämäläinen, "Compression Methods for Microclimate Data Based on Linear Approximation of Sensor Data," *NEW2AN 2019, Lecture Notes in Computer Science*, vol 11660. Springer, Cham. https://doi.org/10.1007/978-3-030-30859-9_3

[2] O. Väänänen and T. Hämäläinen, "Sensor Data Stream on-line Compression with Linearity-based Methods," *2020 IEEE International Conference on Smart Computing (SMARTCOMP)*, 2020, pp. 220-225, doi: 10.1109/SMARTCOMP50058.2020.00049.

[3] T. Schoellhammer, B. Greenstein, E. Osterweil, M. Wimbrow and D. Estrin, "Lightweight temporal compression of microclimate datasets [wireless sensor networks]," *29th Annual IEEE International Conference on Local Computer Networks*, 2004, pp. 516-524, doi: 10.1109/LCN.2004.72.

[4] S. M. S. Jalaleddine, C. G. Hutchens, R. D. Strattan and W. A. Coberly, "ECG data compression techniques-a unified approach," in *IEEE Transactions on Biomedical Engineering*, vol. 37, no. 4, pp. 329-343, April 1990, doi: 10.1109/10.52340.

[5] D. Parker, M. Stojanovic, and C. Yu, "Exploiting temporal and spatial correlation in wireless sensor networks," *2013 Asilomar Conference on Signals, Systems and Computers*, Pacific Grove, CA, 2013, pp. 442-446, doi: 10.1109/ACSSC.2013.6810315.

[6] J. Azar, A. Makhoul, R. Darazi, J. Demerjian, and R. Couturier, "On the performance of resource-aware compression techniques for vital signs data in wireless body sensor networks," *2018 IEEE Middle East and North Africa Communications Conference (MENACOMM)*, Jounieh, 2018, pp. 1-6, doi: 10.1109/MENACOMM.2018.8371032.

[7] O. Sarbishei, "Refined Lightweight Temporal Compression for Energy-Efficient Sensor Data Streaming," *2019 IEEE 5th World Forum on Internet of Things (WF-IoT)*, Limerick, Ireland, 2019, pp. 550-553, doi: 10.1109/WF-IoT.2019.8767351.

[8] B. Li, O. Sarbishei, H. Nourani, and T. Glatard, "A multi-dimensional extension of the Lightweight Temporal Compression method," *2018 IEEE International Conference on Big Data (Big Data)*, Seattle, WA, USA, 2018, pp. 2918-2923, doi: 10.1109/BigData.2018.8621946.

[9] L. Klus *et al.*, "Direct Lightweight Temporal Compression for Wearable Sensor Data," In *IEEE Sensors Letters*, vol. 5, no. 2, pp. 1-4, Feb. 2021, doi: 10.1109/LSENS.2021.3051809.

[10] S. Lu, Q. Xia, X. Tang, X. Zhang, Y. Lu and J. She, "A Reliable Data Compression Scheme in Sensor-Cloud Systems Based on Edge Computing," in *IEEE Access*, vol. 9, pp. 49007-49015, 2021, doi: 10.1109/ACCESS.2021.3068753.

[11] W. W. Piegorsch, Statistical Data Analytics: Foundations for Data Mining, Informatics, and Knowledge Discovery. Chichester, West Sussex: Wiley, 2015.

[12] S. Chatterjee and A. S. Hadi, Regression Analysis by Example, John Wiley & Sons, 2012.

[13] O. Väänänen, M. Zolotukhin, and T. Hämäläinen, "Linear Approximation Based Compression Algorithms Efficiency to Compress Environmental Data Sets," In *Web, Artificial Intelligence and Network Applications. WAINA 2020. Advances in Intelligent Systems and Computing,* vol 1150. Springer, Cham. doi: 10.1007/978-3-030-44038-1_11

[14] N. Q. V. Hung, H. Jeung, and K. Aberer, "An Evaluation of Model-Based Approaches to Sensor Data Compression," in *IEEE Transactions on Knowledge and Data Engineering*, vol. 25, no. 11, pp. 2434-2447, Nov. 2013, doi: 10.1109/TKDE.2012.237.

[15] O. Väänänen and T. Hämäläinen, "LoRa-Based Sensor Node Energy Consumption with Data Compression," *2021 IEEE International Workshop on Metrology for Industry 4.0 & IoT (MetroInd4.0&IoT)*, 2021, pp. 6-11, doi: 10.1109/MetroInd4.0IoT51437.2021.9488434.

[16] O. Väänänen and T. Hämäläinen, "Efficiency of temporal sensor data compression methods to reduce LoRa-based sensor node energy consumption", In *Sensor Review*, Vol. 42 No. 5, pp. 503-516, 2022, https://doi-org.ezproxy.jyu.fi/10.1108/SR-10-2021-0360

[17] Finnish Meteorological Institute's open data–service. [Online] Available: https://en.ilmatieteenlaitos.fi/open-data