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.

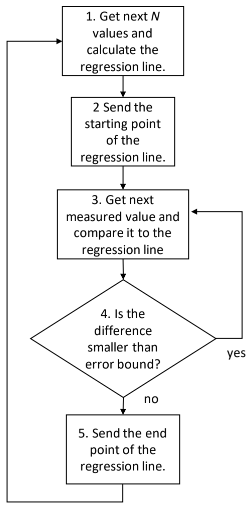
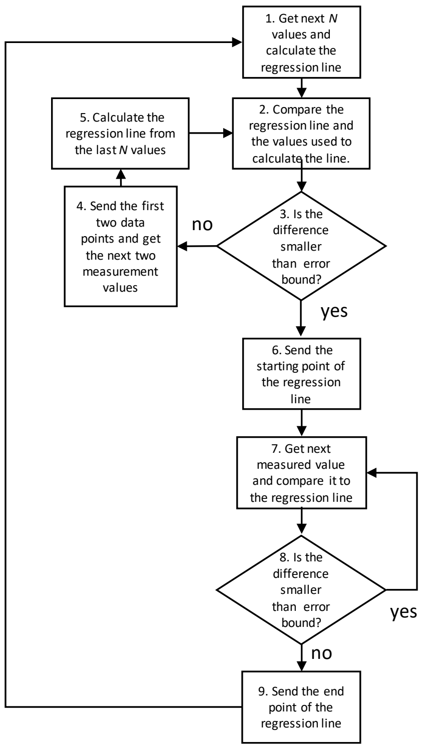

1. Get next N values and calculate the regression line.
2 Send the starting point of the regression line.
3. Get next measured value and compare it to the regression line
4. Is the difference smaller than error bound?
yes
no
5. Send the end point of the regression line.

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 $S = \langle(v_1,t_1), (v_2,t_2),\dots (v_n,t_n)\rangle$, 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 $LRbTC(S) = \langle(c_1,\tau_1), (c_2,\tau_2),\dots (c_n,\tau_n)\rangle$.

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.
3. Is the difference smaller than error bound?
no
4. Send the first two data points and get the next two measurement values
5. Calculate the regression line from the last N values
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

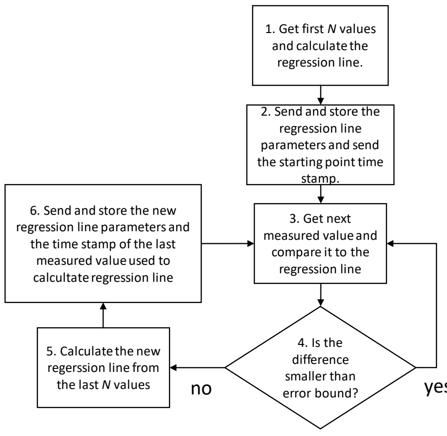
Fig. 2. Modified LRbTC (M-LRbTC).

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 $\langle(v_1,t_1),(v_2,t_2)\rangle$ 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: $\langle(a_1,b_1,\tau_1),(c_2,\tau_2),(a_3,b_3,\tau_3),(c_4,\tau_4),\dots (a_{n-1},b_{n-1},\tau_{n-1}), (c_n,\tau_n)\rangle$.

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

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

The raw data of sensor signal is $S = \langle(v_1,t_1),(v_2,t_2),\dots (v_n,t_n)\rangle$. 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 $\langle(v_1,t_1),(v_2,t_2),(v_3,t_3)\rangle$ 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 $t_3 - t_1 = 2 \times \Delta t$. 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.

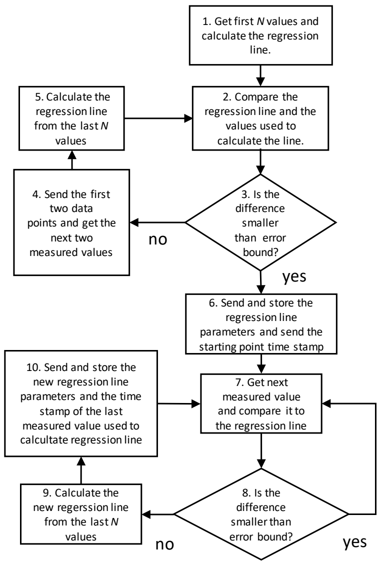

1. Get first N values and calculate the regression line.
2. Compare the regression line and the values used to calculate the line.
3. Is the difference smaller than error bound?
4. Send the first two data points and get the next two measured values
5. Calculate the regression line from the last N values
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?
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
no
yes
no
yes

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 $\tau_2 - \Delta t = t_{10}$.

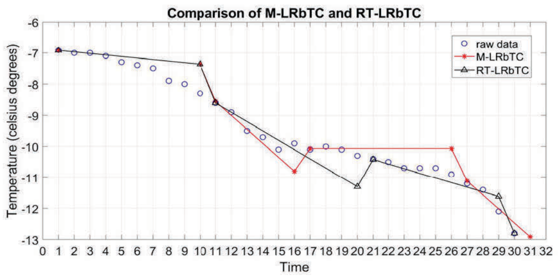


Fig. 5. Comparison of M-LRbTC and RT-LRbTC.

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.

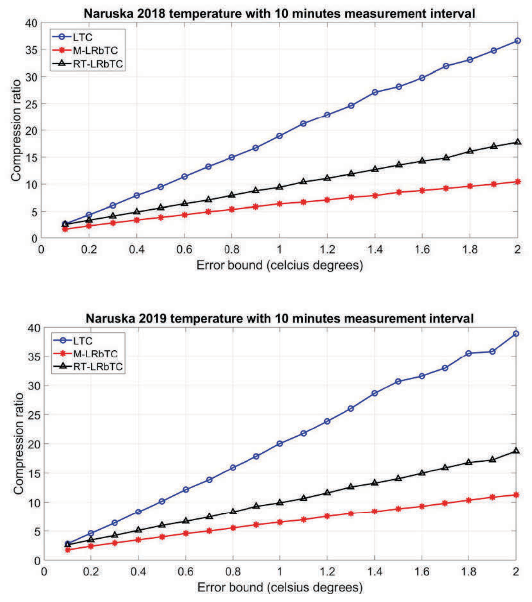


Fig. 6. Comparison on the algorithms with temperature data.

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.

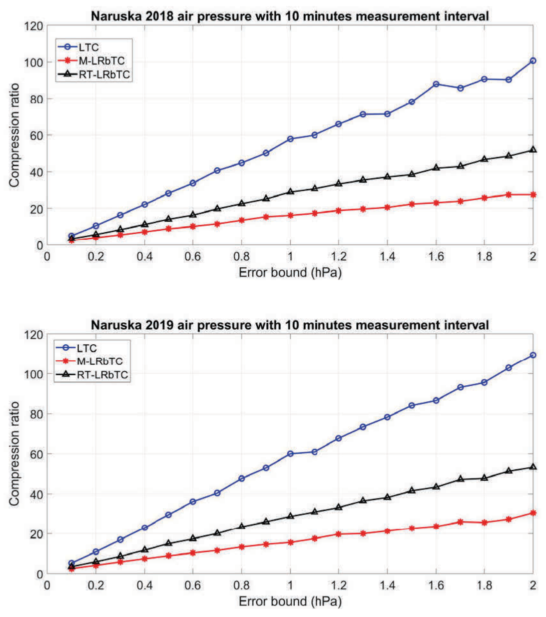


Fig. 7. Comparison of the algorithms with air pressure data.

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].

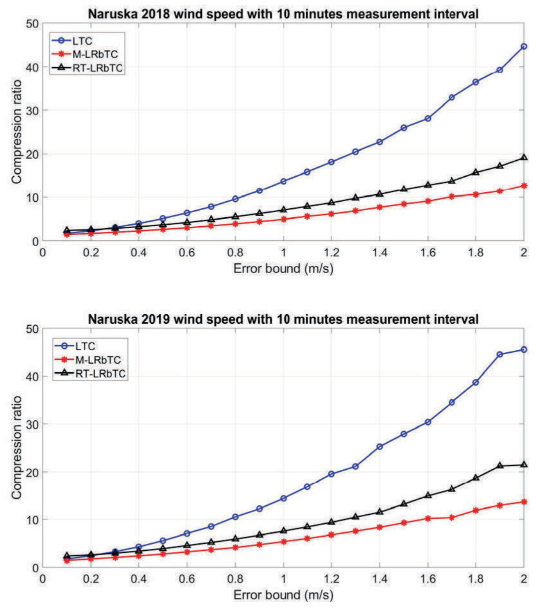


Fig. 8. Comparison of the algorithms with wind speed data.

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	$(N-1) \times \Delta t$	$(N-1) \times \Delta t$	$(N-1) \times \Delta t$
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