Refined Lightweight Temporal Compression for Energy-Efficient Sensor Data Streaming

Omid Sarbishei
Research and Development Department, Motsai Research, Saint Bruno, QC, Canada
o.sarbishei@motsai.com

***Abstract*— Lightweight Temporal Compression (LTC) is an energy-efficient lossy compression algorithm that maintains a memory usage and per-sample computational cost in O(1). The method provides a trade-off between compression ratio and accuracy using an error bound. In this paper, we present the Refined LTC (RLTC) algorithm, which uses a binning approach to widen the search space and increase the LTC's compression ratio and reduce its dynamic energy consumption, which is characterized by CPU computations and radio transmissions, without compromising the error bound. The proposed RLTC algorithm adds negligible overhead to the memory usage and latency of LTC. Experimental results on an environmental sensor dataset have shown that the LTC's compressed byte stream can be further reduced in size by up to 18%, while the dynamic energy consumption is reduced by 9.5% on average.**

I. INTRODUCTION

Recent technological advances in Internet of Things (IoT) applications have led to the prediction of launching possibly more than one billion connected objects worldwide by 2025[1]. With plethora of data being generated by these connected objects, it is often beneficial to utilize low-cost compression algorithms on such devices to save storage and power consumption on radio transmissions, while maintaining low CPU computations [2]. In industrial domains, the low-power operation is crucial, since the sensing systems are expected to run in the field for days or weeks without being recharged. As a common application, environmental sensor systems are designed to send temperature, humidity, wind, etc., to a host system over a low-power radio transmission protocol, such as Bluetooth Low Energy (BLE) or a wide area network.

Several applications require lossless data compression. However, in the context of IoT sensing systems, the sensor readings intrinsically involve noise and measurement errors, which can be treated as a configurable tolerance for a lossy compression algorithm. Lossy compression methods that are resource-intensive, such as the ones based on polynomial interpolation, discrete cosine and Fourier transforms, or auto-regression [1], are not well-suited for low-power connected objects due to their limited available memory (typically a few KB), and the energy consumption associated with CPU usage. The research studies in [2,8] have shown that most computationally heavy lossy compression algorithms suffer from a high energy consumption by CPU, which is often comparable to the energy consumed by radio transmissions. Consequently, [2,8] have demonstrated that for smoothly changing environmental data, the Lightweight Temporal Compression (LTC) in [3] is an energy-efficient solution. LTC estimates data points using a piece-wise linear function that guarantees an upper bound on the maximum absolute error between the reconstructed signal and the original one, while maintaining a memory usage and per-sample latency in $O(1)$. LTC is extended to multi-dimensional signals in [5]. For applications that target non-smooth signals with small error bounds on compression, the use of adaptive and predictive methods, such as the Generalized Predictive Coding (GPC) framework in [7], is more beneficial.

This paper presents the Refined LTC (RLTC) algorithm that analogous to the conventional LTC method in [3] moves each sample up or down by a small amount within an acceptable error bound to compress the data stream. The proposed RLTC method utilizes a binning approach to widen the search space for LTC and reach a higher Compression Ratio (CR) and a lower energy without compromising the error bound. The RLTC algorithm adds negligible overhead to the memory usage and latency-per-sample of LTC.

II. REFINED LIGHTWEIGHT TEMPORAL COMPRESSION

This section presents the proposed RLTC algorithm.

A. Notations

The algorithm receives a stream of data values $x_i \in \mathbb{R}$ at times t_i ($i \in \mathbb{N}$), and it transmits a stream of data values $y_i \in \mathbb{R}$ at times τ_i ($i \in \mathbb{N}$). We assume that

$$\forall k \in \mathbb{N}, \exists! \, i \in \mathbb{N} \;\; \tau_k = t_i,$$

That is, transmission times coincide with reception times. We denote the ordered pair $S = (t, x)$ as a data point with the value of x at time t. We also denote the two elements of S as

$$\pi_1(S) = t, \pi_2(S) = x.$$

B. Conventional LTC

The conventional LTC algorithm [3] maintains a high line initialized by the two data points $z = (t_0, x_0)$ and $(t_1, x_1 + \varepsilon)$ as well as a low line initialized by z and $(t_1, x_1 - \varepsilon)$, where ε is the error bound. For any upcoming sample, the slope of the

[1] https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide

978-1-5386-4980-0/19/$31.00 ©2019 IEEE

high (low) line is reduced (increased) monotonically to respect the error bound, until the high line falls below the low line. Finding the maximum (latest) timestamp t_T that keeps the high line above the low line is formally described as follows:

$$t_T = \max_{t>\pi_1(z),t\in\{t_i\}}(t): \forall l,k \in \mathbb{N}, t_k, t_l \in (\pi_1(z), t],$$
$$\min_k\left(\frac{x_k+\varepsilon-\pi_2(z)}{t_k-\pi_1(z)}\right) \geq \max_l\left(\frac{x_l-\varepsilon-\pi_2(z)}{t_l-\pi_1(z)}\right). \quad (1)$$

At the time the above condition is violated, i.e., $t = t_{T+1}$, the initial data point z is transmitted, and the midpoint of the last acceptable interval between the high line and the low line at $t = t_T$ is chosen as the new initial data point $z = (t_T, M)$, where M is the midpoint value. The high line and the low line are then reinitialized by connecting $z = (t_T, M)$ to $(t_{T+1}, x_{T+1} + \varepsilon)$ and $(t_{T+1}, x_{T+1} - \varepsilon)$, respectively.

C. Binning Approach

While at each iteration, LTC is initialized by a single data point z, we generalize the search space to an interval centered by z, which respects the error bound. This extension converts the problem into a constrained linear programming problem, whose solution majorly increases the latency/memory usage.

Since the error bound ε is a small value, we propose a simpler solution using a binning approach. At each iteration, considering the initial data point z, we consider the acceptable interval from the lower data point $(\pi_1(z), \pi_2(z) - \Delta)$ to the higher data point $(\pi_1(z), \pi_2(z) + \Delta)$, where Δ is defined by the last acceptable higher and lower lines in the previous iteration, and we initially have $\Delta = \varepsilon$ at $z = (t_0, x_0)$. We then encode the interval using $n > 1$ equally distanced bins:

$$z_j = (\pi_1(z), \pi_2(z) - \Delta + \tfrac{2j\Delta}{n-1}), j \in \{0,1,\dots,n-1\}. \quad (2)$$

For each individual bin z_j, we then find the longest compression time t_{T_j} analogous to Eq. (1). Next, the bin z_j with maximum t_{T_j}, i.e., longest temporal compression, will be transmitted. Finally, n new bins are reinitialized within the last interval between the high line and low line at $t = \max(t_{T_j})$.

Lemma 1: Assume an iteration of the LTC algorithm with the initial data point z, where the corresponding longest compression time t_T, is found by Eq. (1). Under such conditions, choosing $n = 2^k + 1$ bins ($k \in \mathbb{N}^+$) over the error interval centered by z in Eq. (2), delivers the same or a longer compression time compared to LTC, i.e., $\max(t_{T_j}) \geq t_T$.

Proof: Since $n = 2^k + 1$ $(k \in \mathbb{N}^+)$ and $j < n$, one possible value for j is 2^{k-1}. Putting $j = 2^{k-1}$ in Eq. (2) gives:

$$z_{j=2^{k-1}} = \left(\pi_1(z), \pi_2(z)\right) = z \overset{Eq.(1)}{\Longrightarrow} t_{T_{j=2^{k-1}}} = t_T, \quad (3)$$
$$\max\left(t_{T_j}\right) \geq t_{T_{j=2^{k-1}}} \overset{Eq.(3)}{\Longrightarrow} \max\left(t_{T_j}\right) \geq t_T. \quad \blacksquare$$

Lemma 1 indicates that for each iteration with identical initial conditions, choosing $n = 3,5,9$, etc., bins, compresses the same or more samples compared to the conventional LTC.

Fig. 1 illustrates the proposed binning approach using an example with $n = 3$ bins. The blue bin at t_0 delivers $\max\left(t_{T_j}\right)$, and thus, is transmitted at t_4. Furthermore, three new and equally distanced initial bins are generated at t_3 based on the higher and lower points of the last acceptable blue region. This process continues for the next upcoming samples.

Algorithm 1 describes the RLTC method, which is initially

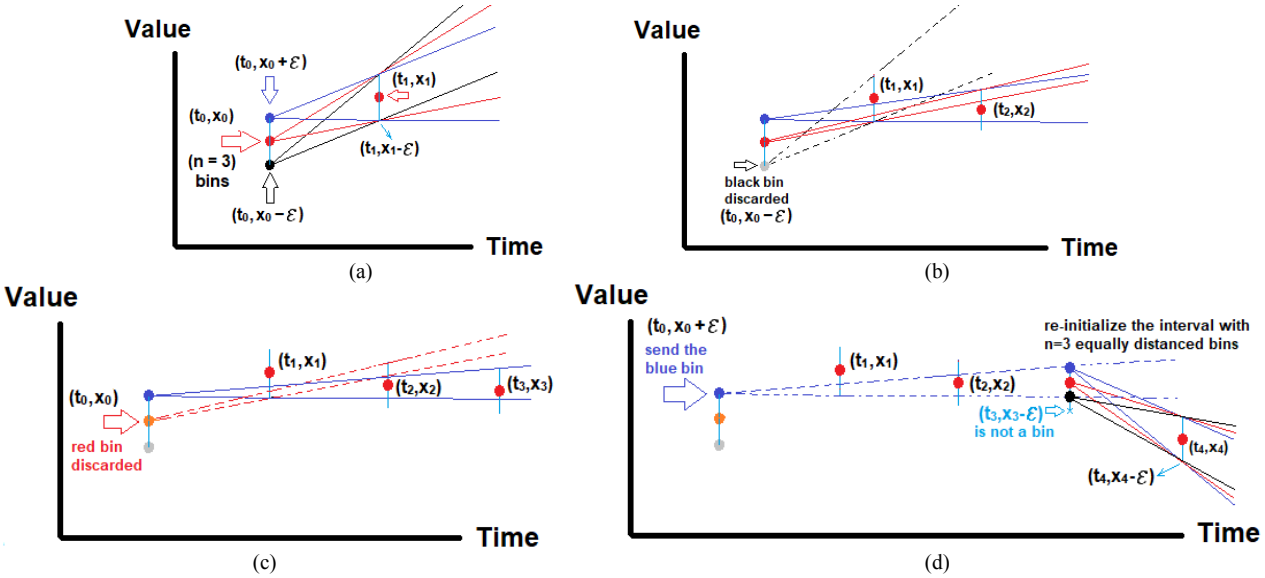

Figure 1. Refined Lightweight Temporal Compression with $n = 3$ initial bins. a) the three bins at time t_0 (black, blue and red) are equally distanced from each other over a vertical interval, where the centered value is equal to the value of the sample received at time t_0, i.e., x_0, and the upper and lower bounds are defined by the given error bound, i.e., $x_0 + \varepsilon$ and $x_0 - \varepsilon$. The higher and lower lines connecting each individual bin to the upper/lower bounds of the second sample (t_1, x_1) generate three regions represented by the black, blue and red colors. b) the third sample (t_2, x_2) has an acceptable interval from $(t_2, x_2 - \varepsilon)$ to $(t_2, x_2 + \varepsilon)$, which is outside the black region, so the black bin is discarded, while the red and blue regions are squeezed. c) the forth sample's interval at t_3 is outside the red region, so the red bin is discarded, while the blue region is further squeezed. d) the next sample's interval at t_4 is outside the last valid bin (blue bin) region, so the corresponding initial blue bin $(t_0, x_0 + \varepsilon)$ is transmitted as the first output sample. Furthermore, three new and equally distanced initial bins are generated at t_3 based on the higher and lower points of the the last acceptable blue region. This process continues for the next upcoming samples.

Algorithm 1: Refined LTC.
Parameters: n: Number of bins ($n > 1$), ε: Error bound
Inputs: (t_i, x_i): Received data points (Lines 1 and 5),
Outputs: tr: Transmitted/compressed data points (Line 23)
1: ReadTwoSamples(); *//block to receive* (t_0, x_0) *and* (t_1, x_1)
2: $T = 1$; *//current timestamp index*
//reset higher/lower points $hp_{0:n-1}$ *and* $lp_{0:n-1}$ *and all bins*
3: $hp_{0:n-1} = x_1 + \varepsilon$; $lp_{0:n-1} = x_1 - \varepsilon$; *//high & low points*
4: $z_{0:n-1}$ = bin$((t_0, x_0), \varepsilon)$; *//Eq. (2) with* z=(t_0, x_0) *and* Δ=ε
5: ReadOneSample(); *//block to receive* (t_{T+1}, x_{T+1})
6: T += 1; *//increase the current timestamp index*
//find the bin index with the widest region/interval
7: $m = \underset{j}{\arg\max}\,(hp_j - lp_j)$; *//priority to central bins*
8: z_lp = lp_m; z_hp = hp_m; *//back up widest low/high points*
9: no_bins_flag = True; *//no good bins found yet*
10: **for** j = 0 to $n-1$ *//with* $n > 1$
11: if $(hp_j < lp_j)$ continue; *//ignore discarded bins*
//extend the line from bin z_j *through the higher/lower point*
12: new_hp = line$(t_T, z_j, (t_{T-1}, hp_j))$;
13: new_lp = line$(t_T, z_j, (t_{T-1}, lp_j))$;
14: **if** (new_hp $< x_T - \varepsilon$) or (new_lp $> x_T + \varepsilon$)
15: $hp_j = -1$; $lp_j = 1$; *//discard the bin*
16: **else** *//squeeze the acceptable region*
17: hp_j = min$(x_T + \varepsilon$, new_hp);
18: lp_j = max$(x_T - \varepsilon$, new_lp);
19: no_bins_flag = False; *//one good bin was found*
20: **end if**
21: **end for**
22: **if** (no_bins_flag) *//all bins are discarded*
23: tr = z_m; *//transmit the bin* z_m *with the widest region*
24: $lp_{0:n-1} = x_T - \varepsilon$; $hp_{0:n-1} = x_T + \varepsilon$; *//reset lp, hp*
25: $z_{0:n-1}$= bin$\left(\left(t_{T-1}, \frac{(\text{z_lp}+\text{z_hp})}{2}\right), \frac{(\text{z_hp}-\text{z_lp})}{2}\right)$; *//Eq.(2)*
26: **end if**
27: Go back to Line 5 to read a new input data point

configured by the parameters n and ε. It then continuously reads the samples (t_i, x_i), and outputs the compressed ones.

At Line 1, the initial two samples (t_0, x_0) and (t_1, x_1) are read using a blocking function. Line 3 sets all the n higher points $hp_{0:n-1}$ to $x_1 + \varepsilon$, and sets all the n lower points $lp_{0:n-1}$ to $x_1 - \varepsilon$. The function *bin*() at Line 4 and Line 25 sets the n individual bins $z_{0:n-1}$ according to Eq. (2), where z is the 1st argument and Δ is the 2nd argument of *bin*().

Line 7 finds the bin index m that represents the widest acceptable region, i.e., higher point minus lower point. If more than one bin is found, the index m closer to the midpoint index $(n-1)/2$ is selected, i.e., priority is given to the central bins. Line 8 backs up the lower/higher points of the widest region.

Line 10 provides a loop to update the acceptable region of the individual bins. The function $line()$ at Line 12-13 delivers the data point value at its first argument, i.e., $t = t_T$, based on the line defined by the data points in the 2nd and 3rd arguments. If all the bins are discarded, at Line 23, we transmit the bin z_m, where m is set at Line 7. Next, we reinitialize $lp_{0:n-1}$, $hp_{0:n-1}$, and the bins $z_{0:n-1}$ (Lines 24-25). Note that when the transmission latency is negligible compared to the sampling period, we are not required to transmit the timestamps t_i.

Algorithm 1 requires $3n$ global variables to store the higher and lower points and all the bins, i.e., $hp_{0:n-1}$, $lp_{0:n-1}$, $z_{0:n-1}$. Therefore, the memory usage complexity is $O(n)$. The latency-per-sample also involves iterations over the individual bins with some scalar arithmetic operations. This gives the complexity of $O(n)$ for the per-sample latency. Fortunately, given a small error bound, the number of bins n required to achieve a high enough resolution is often very small. In fact, our experiments have shown that using typical error bounds for sensors, over 98% of the overall improvements in CR are found with only $n = 2$ bins. This corresponds to a negligible overhead in memory/latency compared to LTC. Furthermore, a small value of n indicates that the global memory usage and latency per sample can be re-stated with the complexity of $O(1)$, which matches the conventional LTC algorithm.

III. Experimental Results

In this section, we evaluate the CR and energy-efficiency of the proposed RLTC method compared to previous work using the outside temperature, humidity and wind data from [4], which have been filtered and sampled every 15 minutes for two months. We consider different error bounds ε and number of bins n. CR is also a percentage defined as follows:

$$\text{CR} = 100(1 - compressed\_bytes/total\_bytes).$$

A. Compression Ratios Compared to LTC

First, we compare the CRs found by RLTC and the LTC in [3] for temperature data using $\varepsilon = \pm 0.1, \pm 0.25, \pm 0.5^{\circ}\text{C}$. The value of $n = 1$ corresponds to the conventional LTC, while $n \geq 2$ bins, refers to the proposed RLTC. Beyond $n = 2$ bins, CR is not improved majorly. At $\varepsilon = \pm 0.1^{o}$, the use of 2 bins increases CR by ~1.1% compared to LTC, i.e., a ~7.1% reduction in the size of the LTC-compressed byte stream.

Next, the CRs for relative humidity are evaluated as shown in Fig. 3. With $\varepsilon = \pm 0.1\%, \pm 0.25\%, \pm 0.5\%$, using $n \geq 2$ bins, delivers similar CR results. Depending on ε, the CR is increased between 1.4% and 4% using 3 bins. This reduces the size of the LTC compressed stream between 11% and 12%.

Fig. 4 depicts the CR for wind data with the error bounds $\varepsilon = \pm 0.25, \pm 0.5, \pm 0.8$ in mph. At $n = 3$, depending on ε, the CR is increased between 5.22% and 6.08%. With $\varepsilon = \pm 0.8$ mph, the LTC-compressed stream is reduced by ~18%.

The mean-square-error for LTC is comparable to RLTC at $n > 2$. Choosing $n = 3$ is energy-efficient and it involves the midpoint bin unlike $n = 2$ (Lemma 1). We keep the midpoint bin to reach a zero error, under no compression (See Line 7).

B. Compression Ratios Compared to Predictive Methods

Even though RLTC outperforms LTC in all cases, for the non-smooth wind data at lower errors, e.g., $\varepsilon = \pm 0.25$mph, RLTC will perform poorly against predictive methods [7]. We compared RLTC at $n = 3$ with the GPC/S-LEC in [7] with the packet size of 32 samples and found that the CR by RLTC is higher than GPC across all error bounds for temperature and humidity (12% higher on average), but for wind data, the CR

by GPC beats RLTC at $\varepsilon = \pm 0.25, \pm 0.5$mph (15% higher CR on average), but it becomes worse than RLTC at $\varepsilon \geq 0.8$mph.

C. Dynamic Energy Consumption by CPU and BLE Radio

Next, we evaluate the memory usage, per-sample latency and dynamic energy consumption for LTC and RLTC with $n = 3$ on Neblina [6] using the nRF52832's CPU/BLE-radio power models and Nordic's Power Profiler Tool. Neblina uses a normal 0dB transmission power at 1.8V supply voltage. The dynamic energy is characterized by the CPU usage for compression and radio transmissions on nRF52, i.e., we do not consider the sensor or the static energy within idle periods. We use the temperature data with $\varepsilon = \pm 0.25^o$. The system is assumed to wake up every 15 minutes to read the data. If a transmission is required, a BLE advertisement phase and then a BLE connection phase occur in 100ms intervals to transmit the temperature packet, and then the system goes back to sleep. The maximum BLE packet size is 20 bytes. This means that involving a packet header, we cannot pack many samples in a single packet. This is particularly a limitation for certain predictive methods, such GPC in [7], which requires several encoded samples to be sent in one packet. As shown in Table I, the CPU power is much lower than BLE for both LTC and RLTC methods. RLTC improves the dynamic energy by 7%.

Table II also presents the results comparing RLTC with

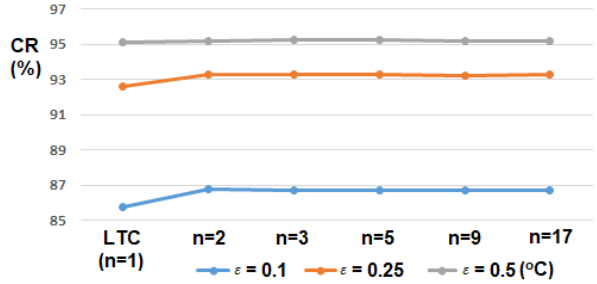

Figure 2. Compression ratios (%) found by LTC ($n = 1$) and the proposed RLTC ($n \geq 2$) for temperature data using different error bounds in °C.

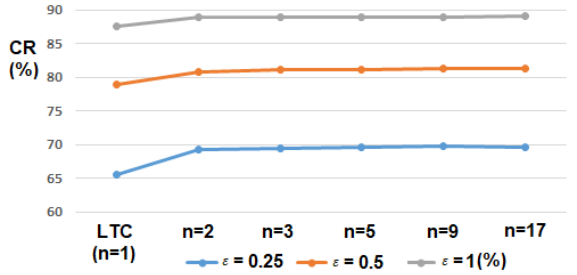

Figure 3. CR (%) found by LTC and RLTC for humidity data with ε in %.

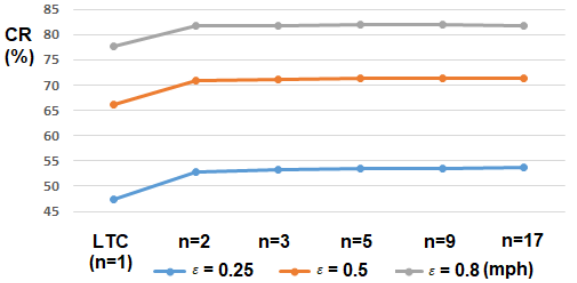

Figure 4. CR (%) found by LTC and RLTC for wind data with ε in mph.

TABLE I. MEMORY USAGE, AVERAGE LATENCY AND ENERGY PER SAMPLE FOR LTC AND RLTC USING TEMPERATURE DATA ON NEBLINA

Method	Latency	RAM (bytes)	Energy (CPU+BLE) (uJ)
LTC	6.9us	24	0.055 + 2.67 = 2.73
RLTC ($n = 3$)	14.4us	52	0.115 + 2.42 = ***2.54***

TABLE II. COMPRESSION RATIO AND AVERAGE CPU+RADIO ENERGY PER SAMPLE USING LTC AND RLTC WITH $n = 3$ BINS ON NEBLINA

Scenario	LTC CR (%)	RLTC CR (%)	LTC Energy (uJ)	RLTC Energy (uJ)
T: $\varepsilon = 0.5^o$	95	*95.22*	1.86	*1.84*
T: $\varepsilon = 0.25^o$	92.58	*93.27*	2.73	*2.54*
T: $\varepsilon = 0.1^o$	85.78	*86.69*	5.18	*4.91*
H: $\varepsilon = 1\%$	87.59	*88.97*	4.53	*4.09*
H: $\varepsilon = 0.5\%$	78.87	*81.15*	7.67	*6.91*
H: $\varepsilon = 0.25\%$	65.52	*69.54*	12.48	*11.1*
W: $\varepsilon = 0.8$mph	77.78	*81.77*	8.06	*6.68*
W: $\varepsilon = 0.5$mph	66.14	*71.17*	12.26	*10.5*
W: $\varepsilon = 0.25$mph	47.5	*53.22*	18.97	*16.97*
Average Improvement (%)		***4.2%***	-	***9.5%***

LTC in terms of CR and average per-sample (every 15 minutes) dynamic energy (CPU + radio). Across all error bounds ε and sensor types, i.e., temperature (T), humidity (H) and wind (W), RLTC with $n = 3$ reaches a lower dynamic energy consumption and a higher CR compared to LTC.

IV. CONCLUSION AND FUTURE WORK

This paper presented the RLTC lossy compression algorithm, as an alternative to the classic LTC approach [3]. Using an environmental dataset, RLTC was shown to improve the CR and energy consumption compared to LTC, while adding negligible overhead to the memory usage and latency. As the next step, we aim to implement the proposed algorithm on a low-power distributed wireless network, such as a BLE mesh network, and evaluate the energy savings in the field. We also plan to combine the benefits of RLTC with adaptive methods to achieve higher CR and lower energy consumption.

ACKNOWLEDGEMENT

We acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC).

REFERENCES

[1] J. Lu, F. Valois, M. Dohler, and M. Y. Wu, "Optimized data aggregation in WSNs using adaptive ARMA", in *IEEE Conference on Sensor Technologies and Applications*, 2010, pp. 115-120.

[2] D. Zordan, B. Martinez, I. Vilajosana, M. Rossi, "On the performance of lossy compression schemes for energy constrained sensor networking", *ACM Trans. on Sensor Networks*, 11 (1), p. 15, 2014.

[3] T. Schoellhammer, E. Osterweil, B. Greenstein, M. Wimbrow, D. Estrin, "Lightweight temporal compression of micro-climate data sets", *IEEE Conf. on Local Computer Networks*, 2004, pp. 516-524.

[4] F. Zamora-Martínez, P. Romeu, P. Botella-Rocamora, J. Pardo, "On-line learning of indoor temperature forecasting models towards energy efficiency", *Energy and Buildings*, Vol. 83, Nov. 2014, pp. 162-172.

[5] B. Li. O. Sarbishei, H. Nourani, T. Glatard, "A multi-dimensional extension of the Lightweight Temporal Compression method", *IEEE Workshop on Real-Time & Stream Analytics in Big Data*, Dec. 2018.

[6] Motsai page: www.motsai.com/products/neblina.

[7] Y. Li, Y. Liang, "Temporal Lossless and Lossy Compression in Wireless Sensor Networks", *ACM Trans. On Sensor Networks*, Vol. 12, No. 4, Article 37, Oct. 2016, pp. 1-35.

[8] M. Arioua, S. al Fallah, J. Elasari, A. E. Oualkadi, "On the performance of piecewise linear approximation techniques in WSNs", *IEEE CommNet Conference*, May 2018, pp.1-6.