Assessing compression algorithms on IoT sensor nodes

Emanuel Guberović, Fran Krišto, Petar Krivić and Igor Čavrak
University of Zagreb, Faculty of Electrical Engineering and Computing, Zagreb, Croatia
{emanuel.guberovic, fran.kristo, petar.krivic, igor.cavrak}@fer.hr

***Abstract* - With the ever-growing presence of IoT devices in everyday applications, the sheer volume of data exchanged is increasing rapidly. Communication channels' bandwidth used in these applications often presents a bottleneck and limits the frequency at which data can be exchanged, making application of compression beforehand a viable approach.**

In this paper, we analyze some of the well known lossless compression algorithms capable of seamless integration as software modules on embedded devices used in (near) real-time applications. Low processing and memory resources available on them as well as unsteadiness of aforementioned communication channels were taken into consideration in algorithm selection.

Attained compression ratio is evaluated on data harvested from smart meters, whilst time and energy consumed during processing were analyzed within a measurement framework purposefully built for this research. We show that considerable gains can be attained in the volume of data that can be sent from devices based on commonly used microcontrollers, with only a modest toll on time and overall energy used.

Keywords* - *compression algorithms, stream of data, real-time systems, sensor nodes, internet of things, smart meters

I. INTRODUCTION

Emergence of Internet of Things (IoT) connecting not only people, but vast number of ubiquitous objects that harvest environmental information and act on their surroundings, proved as the next evolutionary step of the Internet [1]. Smart metering applications present a significant and fast-growing segment of the IoT universe, being the enabler technology for smart grids and smart cities [2-4]. Those devices are characterized by a relatively small amount of data periodically collected from a single end-device, but resulting in a high volume traffic due to the large number of employed end-devices possibly spread over sizeable physical space and sharing the same communication channels.

Requirements for different smart metering applications can exhibit a significant variance in many important aspects such as amount and frequency of collected data, communication costs, and energy consumption. Data collection strategies can vary from just a few bytes a day delivered from end-device on a best-effort basis, to frequent and reliable data collection over a wireless channel in dense and interference-prone environments [5]. Limitations of wireless communication channel capacity can present a significant challenge for such applications where frequent and regular data collection is required in densely populated areas [6].

Employing different data compression techniques on exchanged data can significantly decrease its length, resulting in shortening on-air time of individual end-devices, reducing individual device's energy spent on communication, reducing the cost of exchanged data and allowing the increased frequency of data exchange over shared wireless communication channels.

To assess the usability of different lossless data compression algorithms on IoT sensor nodes, with particular focus on smart metering applications using wireless communication channels, we selected several algorithms - representative of the major lossless data compression methods - and tested their performance from two major aspects: compression ratio efficiency and resource consumption on embedded devices. Compression ratio efficiency was evaluated on utility consumption measurement data sets originating from publicly available data repositories. On the other hand, purposefully built test environment was used to measure the energy consumption of tested algorithms on embedded devices, among other performance measures such as memory and processing power used, the time required to process the exchanged data packet, etc.

Section II of this paper briefly describes the main data compression algorithms and defines algorithm selection criteria. Section III details on the test environment used to assess the suitability of a particular compression method to the studied application context, including performance measures, test datasets, compression method parameters, and execution platforms. Section IV presents the test results and discussion, and Section V concludes the paper.

II. COMPRESSION ALGORTIHMS

In IoT sensor nodes, especially when delta (difference) encoding preprocessing is applied, smaller valued readings can be expected to be more frequent. Compression algorithms used in this paper were selected for their excellent performance on compression of a stream of such data, where its value probability distribution is not known beforehand. In addition to good compression ratio, they provide a lossless encoding process that is friendly for use in embedded devices as algorithms are computationally simple with low space and time complexity.

Some of the assessed algorithms are static, with their symbol to codewords distribution model staying unchanged during the whole encoding process, others are adaptive and change the codewords distribution according to the incoming symbols. Although adaptive algorithms generally have better compression efficiency, static algorithms are more robust in unreliable communication channels as a single wrongly received package can create a discrepancy between adaptive coder and decoder.

A. Elias gamma (γ) and delta codes (δ)

Peter Elias introduced gamma and delta codes in his work on universal codes [7]. Universal codes are prefix codes of codeword length within a constant factor of distance to that of the optimal code if the probability distribution of symbols is monotonically decreasing for ascending integers.

The idea behind Elias codes is to prefix the positive integer being encoded with a representation of an order of its magnitude M, so that $2^M \leq n < 2^{M+1}$. For every n, we can denote n as $n = 2^M + L$, where offset L is a positive integer as well. In γ code, the magnitude part, M is represented in unary code, followed by offset L displayed in its M-bit binary representation. In δ encoding, the magnitude part is represented in γ code instead of unary.

B. Golomb-Rice code

Unlike Elias codes that do not have an external parameter and have a fixed codewords-symbols distribution, codewords of Golomb codes [8] are parameterized by the value m (modulo). Coded symbols are grouped into same length sets (m codewords in one set), where latter sets map bigger integers and have longer codewords. Resulting codeword of integer n consists of quotient $q(n) = \lfloor n/M \rfloor$ and remainder $(n) = n \bmod M$, where q(n) is represented in unary and r(n) in $\lfloor \log_2 M \rfloor$ -bit and $\lceil \log_2 M \rceil$-bit binary codes.

The particular case when m is a power of 2 ($m = 2^k$) is interesting as it allows simplified computation of codewords by using shift operations. Golomb code in this form is widely known as Golomb-Rice [9]. Golomb-Rice code with k=0 is identical to unary code.

Golomb codes are interesting because they are optimal for geometric distribution of symbols, which is similar to some values of natural phenomena sensors are measuring.

C. Adaptive Golomb-Rice code

JPEG2000 [10] specification defines Adaptive Golomb-Rice code algorithm that changes the amount of parameter k using the accumulator of the last encoded values. With estimated value in the accumulator being E [X], k is selected as $k = \max\left\{0, \left\lfloor \log_2\left(\frac{1}{2} E[X]\right)\right\rfloor\right\}$.

D. Exponential-Golomb code

In the Exponential Golomb Code (Exp-Golomb), introduced by Teuhola in [11], sizes of codeword sets grow exponentially, which introduces robustness of code efficiency when the symbol distribution deviates from optimal geometric distribution.

Exp-Golomb code of integer n (of order k=0) is identical to Elias-Gamma code of n+1. Further generalization for higher orders is attained by encoding $n+2^{k-1}$ in Exp-Golomb (k=0) and removing k leading zero bits.

E. Adaptive Exponential-Golomb code

The algorithm for the adaptive Exp-Golomb code was suggested in [12], stating that codeword length using optimal parameter value k is always within interval $[k + 2.8, k + 3.8]$. The adaptive step decreases the value k if the average codeword length in the accumulator of previous values is below the lower limit of that interval or increases the parameter k if it is above the upper limit.

F. Variable byte code

Variable byte codes use a different number of bytes to represent integers. Usually, these algorithms form bytes in a way where every byte has a descriptor bit, and 7 payload bits, descriptor bit carries the information whether the received byte is the last. One of the commonly used algorithms with this form is known as LEB128 (Little-Endian Base 128). A slight deviation from this form is used in varint30, another variable byte algorithm, that carries two descriptor bits in the first byte, with any additional bytes carrying only payload bits.

III. ASSESMENT METHODS

In the first part of our assessment, we analyzed compression efficiency and the processing time of algorithms described in Section II. Using UMass Smart* Dataset-2017, a dataset on electricity consumption containing the total electricity consumption for 114 family apartments during a three year period 2014-2016 [13]. The data were sampled every 15 minutes and saved using the differential code (recording only differences between previous and current consumption readings).

Values in the dataset are stored with a precision up to the tenth decimal, but since practical usage of these sensor values in smart metering often doesn't require precision higher than second decimal, values were normalized to integer values by multiplying with 100 before compression. The distribution of frequencies of normalized values resembles geometrical distribution with a few spikes, visible in Figure 1. The entropy of normalized dataset is 5.3135 b, which presents a theoretical minimal codeword length without any loss of information.

Algorithms were implemented using Python v3.7 and assessed on Windows 10 OS(build 17134.407) PC with Core i5-7300HQ@2.5Ghz CPU. Before evaluating the complete dataset, preliminary evaluation was conducted on 10% of the dataset and interesting parameter space for k value in Golomb-Rice and Exp-Golomb algorithm was found to be k=0-8, as compression efficiency started to deteriorate for values of k bigger than 6.

Customizations were done on the payload size of LEB128 variable byte algorithm and evaluations were conducted for payload sizes 5 and 6 bits. Payload sizes

bigger than 7 and smaller than 4 bits gave worse results in preliminary evaluation.

Adaptive Exp-Golomb and Adaptive Golomb-Rice had counter N initialized at 0, with adaptive parameters refreshment done together with accumulator and counter normalization on every 100th iteration. Accumulator and counter normalizations were done by having their values halved.

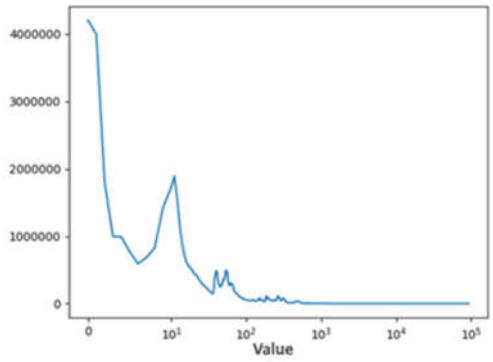

Figure 1. The frequency of values for UMass Smart* Dataset-2017 apartments readings normalized by multiplying with 100

For the second part of our assessment, we selected some of the algorithms analyzed in the first part which appeared suited for use on an embedded device that sends the compressed data over unreliable communication channels. Selected algorithms were rewritten in C++ with $\log_2$ and 2^x operations speed up using bit-shifts and lookup tables. After we implemented selected algorithms on the ESP32-WROOM32 embedded device, processing time along with power consumption of the device during compression was analyzed. It should be noted that although ESP32 is a low powered device, it is quite powerful for an embedded device due to its Xtensa 240Mhz CPU.

Since the test dataset is quite large and it cannot completely fit in ESP32 memory, a special test environment was built which sends chunks of the dataset in predefined batch sizes over serial communication. When the complete batch was received and parsed into the buffer of values a two-way readiness handshake was done (by setting ENCODING_STATUS and ENCODING_ACK pins high) between ESP device and test environment, marking their readiness to start the compression and measuring. When compression of the whole batch finishes, pins used in the handshake are set low, and the next iteration starts.

Code which controls the complete process was developed in Python v3.7 and uses GPIO interrupts and processing time and power consumption measuring was separated into two threads, this allowed for minimum influence communication within different devices had on the overall results. The whole measuring process is visible on the state diagram in Figure 2.

Power consumption was measured using INA219 sensor which relies on the I2C interface to communicate with the test environment. RaspberryPi device has both enough memory needed to store the whole test dataset and all of the needed communication interfaces (serial communication interface, I2C interface, and digital input and output pins for readiness handshake) and proved to be a natural choice for the core of the test environment.

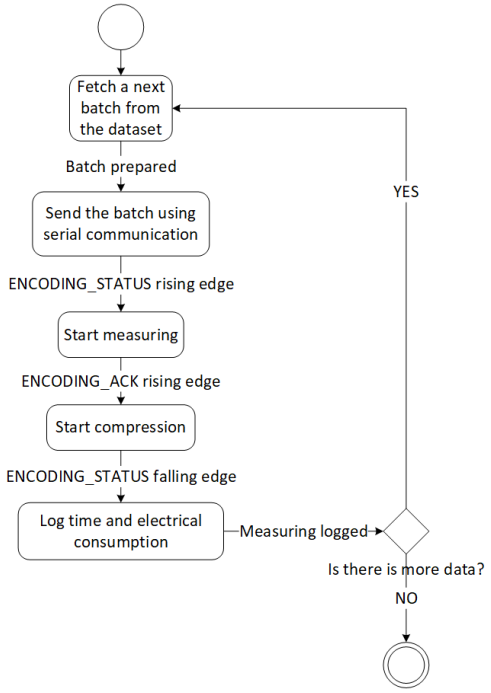

Figure 2. State diagram of the measurement process

IV. ASSESMENT RESULTS

Results from the first part of the assessment described in Section III are visible in Table I. Compression efficiency and processing time are evaluated on the complete UMass Smart-2017 dataset.

TABLE I. COMPRESSION EFFICIENCY AND TOTAL PROCESSING TIME

Algorithm	Compression efficiency	Time[s]
Elias γ	0.5189	220.86
Elias δ	0.5509	326.28
Golomb-Rice(k=6)	0.6333	216.45
Exp-Golomb(k=6)	0.6375	322.62
Adaptive Golomb-Rice	0.6466	418.94
Adaptive Exp-Golomb	0.5293	521.96
LEB128	0.5178	115.86
Varint30	0.4832	183.01

It is interesting to note from these results that Exponential-Golomb (k=6) code has the highest compression efficiency out of all of the static algorithms, Adaptive Golomb-Rice has the best compression efficiency, and LEB128 has the shortest processing time out of all the analyzed algorithms.

On Figure 3 we can observe how selected algorithms compare to the entropy of the UMass apartments dataset. Mean codewords lengths are mostly under the two times

TABLE II. CODEWORD LENGTHS

Algorithm	Mean length	Median length	Percentil-25 length	Percentil-75 length	Minimal length	Maximal length
Elias γ	10.23	11	7	15	1	33
Elias δ	9.64	10	8	14	1	25
Golomb-Rice(k=6)	8.39	7	7	9	7	1456
Exp-Golomb(k=6)	8.34	7	7	9	7	27
Adaptive Golomb-Rice	8.22	8	7	9	1	530
Adaptive Exp-Golomb	10.04	10	10	10	1	25
LEB128	10.26	8	8	16	8	24
Varint30	11.0	8	8	16	8	24

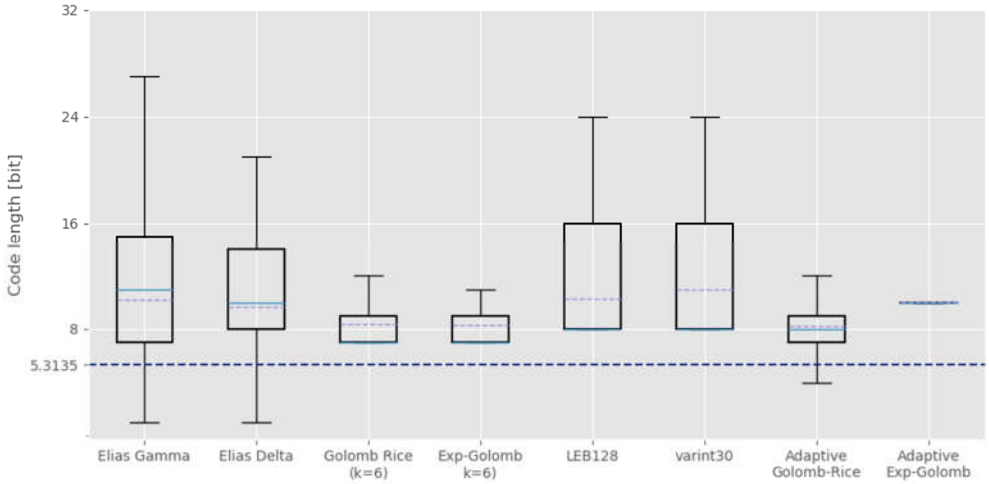

Figure 3. Codeword lengths compared to the dataset entropy (dashed line)

the entropy size, and well under the common 16-bit and 32-bit integer sizes.

Table II displays mean, median, minimal, maximal, 25th percentile and 75th percentile codewords lengths for assessed algorithms. It is interesting to note that Adaptive Golomb-Rice method gave the best result in terms of smallest mean codewords length, but both static and adaptive Golomb-Rice methods have extremely big maximal values, rendering them very troublesome for practical use with the UMass apartments dataset.

A. Customizations to LEB128 code

Adjusting payload size of LEB128 code can improve its compression efficiency, referenced in this paper as varint6p and varint5p, variable byte codes with payload sizes of 6 and 5 bits have compression efficiency comparable to other algorithms while maintaining the short computation time. Codeword lengths of these customizations along with that of the original version can be seen in Figure 4. Both varint6p and varitn5p dropped mean codeword lengths to 9.621 and 9.2 respectively, with varint6p having a lowest maximal value of 21 bits. Interestingly lowering payload size to 5 bits increased maximal value to 24 bits.

Good results with the payload size customizations encourage further assessment of practical usage, with emphasis on decompression, as a possible drawback to these customizations could be the lack of SIMD instructions enhanced decompression that the original LEB128 has, as data is no longer byte-rounded.

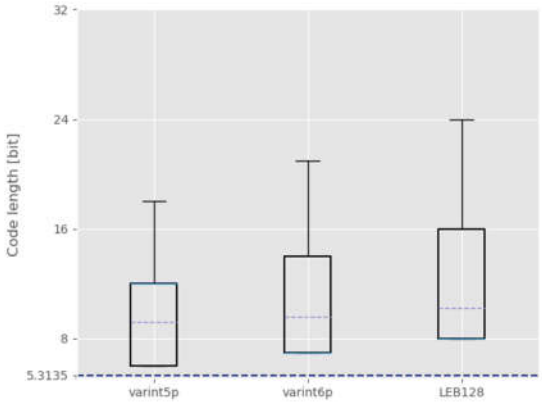

Figure 4. Codeword lengths compared to the dataset entropy (dashed line) – customizations to LEB128

B. Influence of k- parameter in Golomb-Rice and Exp-Golomb algortithms

It is interesting to see how codeword lengths change with different values of parameter *k* in both Golomb-Rice and Exp-Golomb compression algorithms. With bigger *k*-

values, extremes deviate less from the mean value and increasing value *k* from 0 to 6 improves compression efficiency. Enlarging parameter *k* further doesn't improve compression efficiency and introduces a longer processing times. Compression efficiency and total processing time for both algorithms for values k=0-8 are displayed on Table III.

TABLE III. COMPRESSION EFFICIENCY AND TOTAL PROCESSING TIME FOR DIFFERENT K-PARAMETER VALUES IN GOLOMB-RICE AND EXP-GOLOMB ALGORITHMS

Algorithm	Compression efficiency	Time[s]
Golomb-Rice (k=0)	0.0474	179.05
Exp-Golomb (k=0)	0.5189	220.86
Golomb-Rice (k=1)	0.0927	165.39
Exp-Golomb (k=1)	0.5598	287.27
Golomb-Rice (k=2)	0.1746	165.85
Exp-Golomb (k=2)	0.5955	291.44
Golomb-Rice (k=3)	0.3035	167.63
Exp-Golomb (k=3)	0.5569	302.32
Golomb-Rice (k=4)	0.4609	178.70
Exp-Golomb (k=4)	0.5762	309.62
Golomb-Rice (k=5)	0.5800	198.06
Exp-Golomb (k=5)	0.5790	316.80
Golomb-Rice (k=6)	0.6333	216.45
Exp-Golomb (k=6)	0.6375	322.62
Golomb-Rice (k=7)	0.6186	235.55
Exp-Golomb (k=7)	0.6091	329.43
Golomb-Rice (k=8)	0.5767	245.69
Exp-Golomb (k=8)	0.5675	341.77

Both Golomb-Rice and Exp-Golomb algorithms gave best results for $k = 6$, with Exp-Golomb code having bigger mean codeword length values, but much smaller maximal extremes. Golomb-Rice ($k = 0$) has a towering maximal value of 92759 bits, and although small in comparison maximal value is still very big 371 bits even at parameter $k = 8$. Meanwhile, Exp-Golomb algorithm has a maximal value of 33 bits at $k = 0$, down to 25 bits at $k = 8$. Golomb-Rice and Exp-Golomb codeword lengths for different k-parameter values can be seen in Figures 5 and 6 respectively.

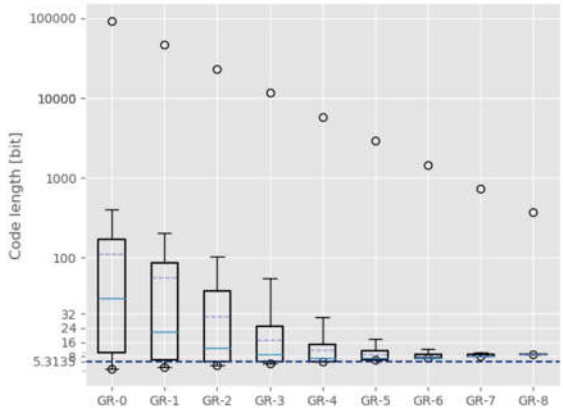

Figure 5. Codeword lengths compared to the dataset entropy (dashed line) – for different *k*-parameter values in Golomb-Rice

These results display the robustness that exponentially growing set sizes introduce in Exp-Golomb algorithm and although Golomb-Rice algorithm has better compression efficiency, spikes in the dataset values distribution that deviate from geometrical distribution make them virtually unusable because of the high maximal values. In practical use one could add a descriptor bit to the compressed payload which would indicate whether the value was compressed or sent in an original form if the compressed value exceeds a set threshold.

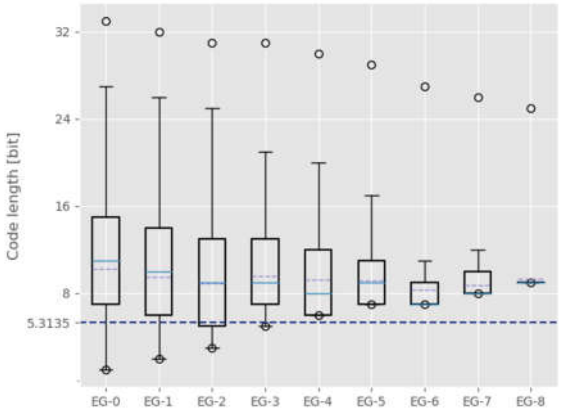

Figure 6. Codeword lengths compared to the dataset entropy (dashed line) – for different *k*-parameter values in Exp-Golomb

C. *Evaluation of selected algorithms on embedded device system ESP32*

In the second part of the assessment, to simulate a real-world scenario where these compression algorithms could be used, compression was evaluated on an embedded device system described in Section III.

Processing time and energy consumption were again assessed on the complete dataset, to minimize any influence measuring process described in Section III had on the overall result, batches of 1000 values were used, and every value was compressed 1000 times.

Overall results are even better than those from the first part of the assessment which was run on a much more powerful PC, this due to speedup attained by optimizing log2 and 2x operations with bit-shifts and lookup tables. Compression efficiency and total processing time for selected algorithms on ESP32 are displayed on Table IV.

TABLE IV. COMPRESSION EFFICIENCY AND TOTAL PROCESSING TIME ON ESP32

Algorithm	Time [s]	Energy [J]
Elias γ	28.25	5.72
Elias δ	45.62	8.13
Golomb-Rice(k=6)	16.719	3.59
Exp-Golomb(k=6)	46.84	10.06
LEB128	5.884	1.25

Although adaptive algorithms had the best compression efficiency in the first part of the assessment, the fact that one lost package can create inconsistency

between coder and decoder, resulting in incorrectly communicated sensor readings is why they are regarded too unreliable for usage in communication channels where package loss is a regular occurrence. Therefore, for further evaluation on sensor nodes following static algorithms were selected because of their excellent compression efficiency and short computation time: Elias γ, Elias δ, Golomb-Rice(k=6), Exp-Golomb(k=6) and LEB128.

Similarly to results in previous assessments, LEB128 was the algorithm that took the shortest processing time. It is interesting to note the energy consumption of ESP32 device during compression, as it could be a reference for energy savings in IoT sensors communication. For a difference in compression efficiency of 15.43%, Exp-Golomb algorithm took around 41 seconds more to compress UMass apartments dataset and while doing that spent 8.81 J of energy more.

V. CONLUSION

Assessment of commonly used variable length code algorithms in a real-world scenario of compressing smart meter readings on an embedded device empirically asserted some theoretical foundations known from before.

Results show how Exp-Golomb code introduces robustness to Golomb codes and that overall compression efficiency is heavily dependent on properly selecting the k parameter. Another assertion was that adaptive compression algorithms can have better compression efficiency than static counterparts as they change the k parameter according to the previously encoded values.

An interesting revelation from the assessments was the plausible usage variable byte algorithms might have in IoT sensor nodes as they introduce little loss in compression efficiency to a large gain in processing time and energy consumption. Results of processing time and total energy spent during compression on an embedded device could be used as a reference to the future evaluation of potential energy savings compression could introduce before sending packages over wireless communication channels.

Results show how optimizing common mathematical operations in compression algorithms shorten their processing time, which in combination with the fact that embedded devices nowadays often come with modestly strong CPU-s makes data transfer from IoT sensor nodes unhindered even if compression is used.

Future work might include assessing decompression algorithms on servers that aggregate data from sensor nodes, with emphasis on SIMD enhanced LEB128 decompression. Another area of research that might induce some positive findings could be self- recovery of adaptive compression methods in unreliable communication channels.

ACKNOWLEDGMENT

This work is partially supported by the HAMAG-BICRO through the project Helm Smart Grid.

REFERENCES

[1] L. Atzori, A. Iera, and G. Morabito, “The Internet of Things: A survey”, Comput. Netw., vol. 54, no. 15, pp. 2787–2805, 2010.

[2] A. Krishnan, "Smart Meter Rollouts From Water Utilities Gain Momentum As Total Installed Smart Meters Swarm to Reach 1.1 Billion Units by 2021”, Retrieved 2016, From ABIresearch. [Online]. Available: https://www.abiresearch.com/press/smart-meter-rollouts-water-utilities-gain-momentum

[3] S. Haben, C. Singleton, and P. Grindrod, “Analysis and clustering of residential customers energy behavioral demand using smart meter data,” IEEE Trans. Smart Grid, vol. 7, no. 1, pp. 136–144, Jan. 2016.

[4] A. Zanella, N. Bui, A. Castellani, L. Vangelista, and M. Zorzi, “Internet of Things for smart cities,” IEEE Internet Things J., vol. 1, no. 1, pp. 22–32, Feb. 2014.

[5] J. Jin, J. Gubbi, S. Marusic, and M. Palaniswami, “An information framework for creating a smart city through Internet of Things,” IEEE Internet Things J., vol. 1, no. 2, pp. 112–121, Apr. 2014.

[6] Kais Mekki, Eddy Bajic, Frederic Chaxel, Fernand Meyer, “A comparative study of LPWAN technologies for large-scale IoT deploymen”,ICT Express, 2018, ISSN 2405-9595, https://doi.org/10.1016/j.icte.2017.12.005.

[7] P. Elias, “Universal Codeword Sets and Representations of the Integers,” IEEE Trans. Inf. Theory, vol. 21, no. 2, pp. 194–203, 1975.

[8] S. W. Golomb, “Run Length Encodings,” IEEE Trans. Inf. Theory, vol. IT-12, no. 5, pp. 399–401, 1966.

[9] R. F. Rice. Some Practical Universal Noiseless Coding Techniques. Technical Report 79/22, Jet Propulsion Laboratory, 1979.

[10] D. S. Taubman, JPEG2000: Image Compression Fundamentals, Standards and Practice, vol. 11, no. 2. 2002.

[11] Teuhola, J. (1978) “A Compression Method for Clustered Bit-Vectors,” Information Processing Letters, 7:308–311, October.

[12] A. Kiely, M. Klimesh , "Generalized Golomb Codes and Adaptive Coding of Wavelet-Transformed Image Subbands", The Interplanetary Network Progress Report, vol. 42-154, p. 1-14, Jet Propulsion Laboratory, 2003.

[13] Smart*: An Open Data Set and Tools for Enabling Research in Sustainable Homes. Sean Barker, Aditya Mishra, David Irwin, Emmanuel Cecchet, Prashant Shenoy, and Jeannie Albrecht. Proceedings of the 2012 Workshop on Data Mining Applications in Sustainability (SustKDD 2012), Beijing, China, August 2012.