Enabling Compression in Tiny Wireless Sensor Nodes

Francesco Marcelloni[1] and Massimo Vecchio[2]
[1]Dipartimento di Ingegneria dell'Informazione, University of Pisa, Via Diotisalvi 2, 56122 Pisa, Italy, e-mail: f.marcelloni@ing.unipi.it
[2]INRIA Saclay, Ile de France sud, France, e-mail: massimo.vecchio@inria.fr

1. Introduction

A Wireless Sensor Network (WSN) is a network composed of sensor nodes communicating among themselves and deployed in large scale (from tens to thousands) for applications such as environmental, habitat and structural monitoring, disaster management, equipment diagnostic, alarm detection, and target classification. In WSNs, typically, sensor nodes are randomly distributed over the area under observation with very high density. Each node is a small device able to collect information from the surrounding environment through one or more sensors, to elaborate this information locally and to communicate it to a data collection centre called *sink* or *base station*. WSNs are currently an active research area mainly due to the potential of their applications. However, the deployment of a large scale WSN still requires solutions to a number of technical challenges that stem primarily from the features of the sensor nodes such as limited computational power, reduced communication bandwidth and small storage capacity. Further, since sensor nodes are typically powered by batteries with a limited capacity, energy is a primary constraint in the design and deployment of WSNs.

Datasheets of commercial sensor nodes show that data communication is very expensive in terms of energy consumption, whereas data processing consumes significantly less: the energy cost of receiving or transmitting a single bit of information is approximately the same as that required by the processing unit for executing a thousand operations. On the other hand, the energy consumption of the sensing unit depends on the specific sensor type. In several cases, however, it is negligible with respect to the energy consumed by the communication unit and sometimes also by the processing unit. Thus, to extend the lifetime of a WSN, most of the energy conservation schemes proposed in the literature aim to minimize the energy consumption of the communication unit (Croce et al., 2008). To achieve this objective, two main approaches have been followed: power saving through duty cycling and in-network processing. Duty cycling schemes define coordinated sleep/wakeup schedules among nodes in the network. A detailed description of these techniques applied to WSNs can be found in (Anastasi et al., 2009). On the other hand, in-network processing consists in reducing the amount of information to be transmitted by means of aggregation (Boulis et al., 2003) (Croce et al., 2008) (Di Bacco et al., 2004) (Fan et al., 2007)

(Intanagonwiwat et al., 2003) (Lindsey et al., 2002) (Madden et al., 2002) and/or compression techniques. In this chapter, we do not consider aggregation: the interested reader can refer to (Fasolo et al., 2007) for a brief discussion and classification of aggregation approaches.

Data compression algorithms fall into two broad classes: lossless and lossy algorithms. Lossless algorithms guarantee the integrity of data during the compression/decompression process. On the contrary, lossy algorithms generate a loss of information, but generally ensure a higher compression ratio.

Due to the limited resources available in sensor nodes, to apply data compression in WSNs requires specifically designed algorithms. Two approaches have been followed:

1. to distribute the computational cost on the overall network (Chen et al., 2004) (Ciancio & Ortega, 2005) (Ciancio et al., 2006) (Deligiannakis et al., 2004) (Ganesan et al., 2003) (Gastpar et al., 2006) (Girod et al., 2005) (Guestrin et al., 2004) (Lin et al., 2006) (Pradhan et al., 2002) (Rebollo-Monedero, 2007) (Tang & Raghavendra, 2004), (Wagner et al., 2007) (Zixiang et al., 2004);
2. to exploit the statistical features of the data under monitoring so as to adapt some existing algorithms to the constraints imposed by the limited resources available on the sensor nodes (Ganesan et al., 2003) (Lynch et al., 2004) (Sadler & Martonosi, 2006).

The first approach is natural in cooperative and dense WSNs, where data measured by neighbouring nodes are correlated both in space and in time. Thus, we can apply distributed transforms or estimate distributed models which allow decorrelating the data measured by sensors, and, therefore, representing these data by using fewer bits. Obviously, the models are generally only approximations of the data. Thus, distributed compression algorithms are intrinsically lossy.

To the best of our knowledge, only a few papers have discussed the second approach. Examples of compression techniques applied to the single node adapt some existing dictionary-based compression algorithms to the constraints imposed by the limited resources available on the sensor nodes. For instance, in (Sadler & Martonosi, 2006) and (LZO, 2008), the authors have introduced two lossless compression algorithms, namely S-LZW and miniLZO, which are purposely adapted versions of LZW (Welch, 1984) and LZ77 (Ziv & Lempel, 1977), respectively. Since S-LZW outperforms miniLZO, as shown in (Sadler & Martonosi, 2006), we will consider only S-LZW as comparison in this chapter. The Lightweight Temporal Compression (LTC) algorithm proposed in (Schoellhammer et al., 2004) is an efficient and simple lossy compression technique for the context of habitat monitoring. LTC introduces a small amount of error into each reading bounded by a control knob: the larger the bound on this error, the greater the saving by compression.

The choice of the algorithm type (lossless or lossy) depends on the specific application domain. Typically, applications, which are not particularly critical, tolerate the use of sensors that, though very cheap, collect data affected by a non-negligible noise. In this context, lossy compression algorithms can provide a double advantage: to reduce noise and to compress data (Ganesan et al., 2003). On the other hand, the criticality of some application domains demands sensors with high accuracy and cannot tolerate that measures, are corrupted by the compression process. In Body Area Networks, for instance, sensor nodes permanently monitor and log vital signs: each small variation of these signs have to be captured because it might provide crucial information to make a diagnosis. Thus, we believe

that both lossless and lossy compression algorithms suitable to WSNs have to be deeply investigated. Since sensor nodes are typically equipped with a few kilobytes of memory and a 4-8MHz microprocessor, embedding classical data compression schemes in these tiny nodes is practically infeasible (Barr & Asanović, 2006) (Kimura & Latifi, 2005).
To overcome these problems, in a previous paper (Marcelloni & Vecchio, 2009), we have proposed a Lossless Entropy Compression algorithm (LEC), which exploits the natural correlation that exists in data typically collected by WSNs and the principles of entropy compression. We have shown how its low complexity and the small amount of memory required for its execution make the algorithm particularly suited to be used on available commercial sensor nodes. Other important features of LEC are i) its ability to compute a compressed version of each value on the fly and ii) to exploit a very short fixed dictionary, whose size depends on the precision of the analog-to-digital converter (ADC).
The LEC algorithm follows a scheme similar to the one used in the baseline JPEG algorithm for compressing the so-called DC coefficients of a digital image: the basic idea is to divide the alphabet of values into groups whose sizes increase exponentially and consequently to implement the codewords as a hybrid of entropy and binary codes (Pennebaker & Mitchell, 1992). In particular, the entropy code (a variable-length code) specifies the group, while the binary code (a fixed-length code) represents the index within the group. In (Marcelloni & Vecchio, 2009), we have adopted the Huffman table proposed in JPEG to entropy encoding the groups.
In this chapter, first we briefly introduce the LEC algorithm and, by using two real datasets, we discuss how the LEC algorithm outperforms S-LZW and three well-known compression algorithms, namely gzip, bzip2 and rar. We used these three algorithms as benchmarks, but actually these algorithms are not embeddable in tiny sensor nodes. Second, we analyze how the correlation between consecutive samples affects the performance of LEC by downsampling the datasets with different downsampling factors and by evaluating how much the compression ratio decreases. Third, we investigate the use of semi-adaptive and adaptive Huffman coding to increase the performance in the case of reduced correlation between consecutive samples. Fourth, we discuss how LEC can be transformed into a lossy compression algorithm and show how the lossy version considerably outperforms the lossless version in terms of compression ratios without introducing a significant error. Finally, we compare the lossy version of LEC with LTC.
The Chapter is organized as follows. Section 2 introduces the LEC algorithm. In Section 3, we assess the performance of LEC in terms of compression ratios and complexity. Section 4 introduces the lossy version of LEC and shows some preliminary experimental results. Finally, Section 5 gives some conclusions.

2. The LEC Algorithm

Figure 1 shows the block scheme of the LEC algorithm. In the sensing unit of a sensor node, each measure m_i acquired by a sensor is converted by an ADC to a binary representation r_i on R bits, where R is the resolution of the ADC, that is, the number (2^R) of discrete values the ADC can produce over the range of analog values.
For each new acquisition m_i, LEC computes the difference $d_i = r_i - r_{i-1}$, which is input to an entropy encoder (in order to compute d_0 we assume that r_{-1} is equal to the central value among the 2^R possible discrete values). The entropy encoder performs compression

losslessly by encoding differences d_i more compactly based on their statistical characteristics. LEC exploits a modified version of the Exponential-Golomb code (Exp-Golomb) of order 0 (Teuhola, 1978), which is a type of universal code. The basic idea is to divide the alphabet of numbers into groups whose sizes increase exponentially. Like in Golomb coding (Golomb, 1966) and Elias coding (Elias, 1975), a codeword is a hybrid of unary and binary codes: in particular, the unary code (a variable-length code) specifies the group, while the binary code (a fixed-length code) represents the index within the group. Indeed, each nonzero d_i value is represented as a bit sequence bs_i composed of two parts $s_i | a_i$, where s_i codifies the number n_i of bits needed to represent d_i (that is, the group to which d_i belongs) and a_i is the representation of d_i (that is, the index position in the group). When d_i is equal to 0, the corresponding group has size equal to 1 and therefore there is no need to codify the index position in the group: it follows that a_i is not represented.

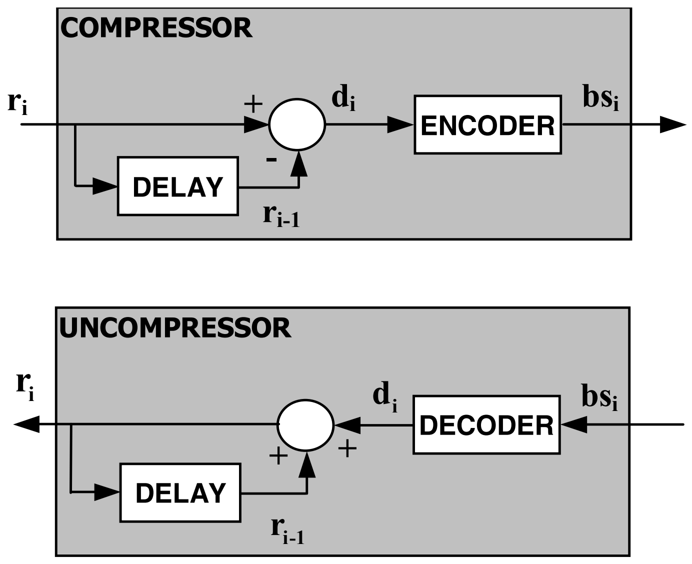

Fig. 1. Block diagram of the encoding/decoding schemes.

For any nonzero d_i, n_i is trivially computed as $\lceil \log_2(|d_i|) \rceil$: at most n_i is equal to R. Thus, in order to encode n_i a prefix-free table of R + 1 entries has to be specified. This table depends on the distribution of the differences d_i: more frequent differences have to be associated with shorter codes. From the observation that, in typical data collected by WSNs, the most frequent differences are those close to 0, in (Marcelloni & Vecchio, 2009) we adopted Table 1, where the first 11 lines coincide with the table used in the baseline JPEG algorithm for compressing the DC coefficients (Pennebaker & Mitchell, 1992). On the other hand, these coefficients have statistical characteristics similar to the measures acquired by the sensing unit. Of course, whether the resolution of the ADC is larger than 14 bits, the table has to be appropriately extended.
In order to manage negative d_i, LEC maps the input differences onto nonnegative indexes, using the following bijection:

$$index = \begin{cases} d_i, & d_i \geq 0 \\ 2^{n_i} - 1 - |d_i|, & d_i < 0 \end{cases} \tag{1}$$

Finally, s_i is equal to the value at entry n_i in the prefix-free table and a_i is the binary representation of *index* over n_i bits. Since d_i is typically represented in two's complement notation, when $d_i < 0$, a_i is equal to the n_i low-order bits of $d_i - 1$.

The procedure used to generate a_i guarantees that all possible values have different codes. Using Table 1, we have, for instance, that d_i = 0, d_i = +1, d_i = -1, d_i = +255 and d_i = -255 are encoded as 00, 010|1, 010|0, 111110|11111111 and 111110|00000000, respectively. Once bs_i is generated, it is appended to the bitstream which forms the compressed version of the sequence of measures m_i.

In the uncompressor, the bit sequence bs_i is analyzed by the decoder block which outputs difference d_i. Difference d_i is added to r_{i-1} to produce r_i.

n_i	s_i	d_i
0	00	0
1	010	-1,+1
2	011	-3,-2,+2,+3
3	100	-7,…,-4,+4,…,+7
4	101	-15,…,-8,+8,…,+15
5	110	-31,…,-16,+16,…,+31
6	1110	-63,…,-32,+32,…,+63
7	11110	-127,…,-64,+64,…,+127
8	111110	-255,…,-128,+128,…,+255
9	1111110	-511,…-256,+256,…,+511
10	11111110	-1023,…,-512,+512, …,+1023
11	111111110	-2047, …,-1024,+1024, …,+2047
12	1111111110	-4095, …,-2048,+2048, …,+4095
13	11111111110	-8191, …,-4096,+4096, …,+8191
14	111111111110	-16383, …,-8192,+8192, …,+16383

Table 1. The default dictionary table.

3. Performance Assessment Results

In our experiments, we have used the temperature and relative humidity measurements collected from a randomly extracted node (NODE ID= 84) of the WSN SensorScope LUCE deployment (SensorScope, 2009), within the time interval from 23/11/2006 to 17/12/2006. The resulting temperature and relative humidity datasets are composed by 64913 samples and we will refer to them as LU_ID84_T and LU_ID84_H, respectively. The WSN adopted in the deployment employs a TinyNode node type (TinyNode, 2009), which uses a TI MSP430 microcontroller, a Xemics XE1205 radio and a Sensirion SHT75 sensor module (Sensirion, 2009).

This module is a single chip which includes a capacitive polymer sensing element for relative humidity and a bandgap temperature sensor. Both the sensors are seamlessly coupled to a 14-bit ADC and a serial interface circuit on the same chip. The Sensirion SHT75 can sense air temperature in the $[-20°C, +60°C]$ range and relative humidity in the $[0\%, 100\%]$ range. The outputs *raw_t* and *raw_h* of the ADC for temperature and relative humidity are represented with resolutions of 14 and 12 bits, respectively. The outputs *raw_t*

and *raw_h* are converted into measures *t* and *h* expressed, respectively, in Celsius degrees (°C) and percentage (%) as described in (Sensirion, 2009). The datasets corresponding to the deployments store measures *t* and *h*. On the other hand, the LEC algorithm works on *raw_t* and *raw_h*. Thus, before applying the algorithm, we extracted *raw_t* and *raw_h* from *t* and *h*, respectively, by using the inverted versions of the conversion functions in (Sensirion, 2009).
Table 2 shows some statistical characteristics of the two datasets. In particular, we have computed the mean $\bar{s}$ and the standard deviation $\sigma_{\bar{s}}$ of the samples, the mean $\bar{d}$ and the standard deviation $\sigma_{\bar{d}}$ of the differences between consecutive samples, the information entropy $H = -\sum_{i=1}^{N} p(x_i) \cdot \log_2 p(x_i)$ of the original signal, where N is the number of possible values x_i (the output of the ADC) and $p(x_i)$ is the probability mass function of x_i, and the information entropy $H_d = -\sum_{i=1}^{N} p(d_i) \cdot \log_2 p(d_i)$ of the differentiated signal.

Dataset	Samples	$\bar{s} \pm \sigma_{\bar{s}}$	$\bar{d} \pm \sigma_{\bar{d}}$	H	H_d
LU_ID84_T	64913	7.21±3.16	$-2.87 \cdot 10^{-5} \pm 0.05$	10.07	4.05
LU_ID84_H	64913	87.04±8.04	$1.12 \cdot 10^{-4} \pm 0.55$	10.08	5.85

Table 2. Statistical characteristics of the datasets.

In the following, we first show the compression ratios achieved by LEC and compare them with the ones achieved by S-LZW and three well-known compression algorithms. We also discuss the complexity of LEC and S-LZW. Then, we investigate the dependence of the compression performance of LEC on the correlation between consecutive samples of the signal to be compressed and show how semi-adaptive and adaptive Huffman coding can help LEC to increase the compression ratios. Finally, we introduce a problem that affects LEC and in general all the differential compression algorithms and discuss how this problem can be solved without considerably penalizing the compression ratios achieved by LEC.

3.1 Compression ratios and complexity

The performance of a compression algorithm is usually computed by using the compression ratio (*CR*) defined as:

$$CR = 100 \cdot \left(1 - \frac{comp\_size}{orig\_size}\right) \tag{2}$$

where *comp_size* and *orig_size* are, respectively, the sizes in bits of the compressed and the uncompressed bitstreams. Considering that uncompressed samples are normally byte-aligned, both temperature and relative humidity samples are represented by 16-bit unsigned integers. Thus, from Table 2, it is easy to compute *orig_size* for the given datasets.
Moreover, assuming that all samples have to be transmitted to the sink by using the lowest number of messages so as to have power saving (Mainwaring et al., 2002) and supposing

that each packet can contain at most 29 bytes of payload (Croce et al., 2008), we can define the packet compression ratio as:

$$PCR = 100 \cdot \left(1 - \frac{comp\_pkt}{orig\_pkt}\right) \tag{3}$$

where *comp_pkt* and *orig_pkt* represent the number of packets necessary to deliver the compressed and the uncompressed bitstreams, respectively.

Table 3 shows the results obtained by LEC in terms of CR and PCR on the two datasets. As expected, the LEC algorithm achieves higher compression ratios on the temperature dataset which is characterized by a lower entropy H_d and, in general, a low variability between consecutive samples (that is, low values of the mean and standard deviation of the differences between consecutive samples).

	Dataset	***orig_size***	***comp_size***	***CR(%)***	***orig_pkt***	***comp_pkt***	***PCR(%)***
LEC	LU_ID84_T	1038608	303194	70.81	4477	1307	70.81
	LU_ID84_H	1038608	396442	61.83	4477	1709	61.83

Table 3. Compression ratios obtained by LEC on the two datasets.

To assess the goodness of the results shown in Table 3, we have also applied the S-LZW algorithm and three well-known compression methods to the same datasets. S-LZW is a lossless compression algorithm purposely developed to be embedded in sensor nodes. S-LZW splits the uncompressed input bitstream into fixed size blocks and then compresses separately each block. During the block compression, for each new string, that is, a string which is not already in the dictionary, a new entry is added to the dictionary. For each new block, the dictionary used in the compression is re-initialized by using the 256 codes which represent the standard character set. Due to the poor storage resources of sensor nodes, the size of the dictionary has to be limited. Thus, since each new string in the input bitstream produces a new entry in the dictionary, the dictionary might become full. If this occurs, an appropriate strategy has to be adopted. For instance, the dictionary can be frozen and used as-is to compress the remainder of the data in the block (in the worst case, by using the code of each character), or it can be reset and started from scratch. To take advantage of the repetitive behaviour of sensor data, a mini-cache is added to S-LZW: the mini-cache is a hash-indexed dictionary of size N, where N is a power of 2, that stores recently used and created dictionary entries. Further, the repetitive behaviour can be used to pre-process the raw data so as to build appropriately structured datasets, which can perform better with the compression algorithm.

In (Sadler & Martonosi, 2006), the authors show that the use of structured datasets and the introduction of the mini-cache increase the compression ratios without introducing appreciable computational overhead. It follows that S-LZW has to balance four major inter-related parameters: the size (BLOCK_SIZE) of the data block, the maximum number (MAX_DICT_ENTRIES) of dictionary entries, the strategy (DICTIONARY_STRATEGY) to follow when the dictionary is full and the number (MINI-CACHE_ENTRIES) of mini-cache

entries. With the aim of putting S-LZW in its best situation, we adopted the values suggested in (Sadler & Martonosi, 2006): a block size of 528 bytes, a dictionary of 512 entries that is maintained once full and a mini-cache of 32 entries.
As regards the well-known compression methods, we have considered gzip, bzip2 and rar. These methods have a parameter which allows setting the compression level. This parameter is between 1 and 9 (default 6) for gzip and bzip2, and between 1 and 5 (default 3) for rar. We fixed this paramater to the maximum possible compression (9 for gzip and bzip2 and 5 for rar).
Table 4 shows the results obtained by the four algorithms. We can observe that LEC outperforms the other algorithms. In the table, we have not shown the PCRs for gzip, bzip2 e rar. Indeed, these algorithms have been used only as benchmarks to validate the compression ratios obtained by applying LEC. Actually, as already observed in (Ganesan et al., 2003) (Kimura & Latifi, 2005) (Sadler & Martonosi, 2006), these algorithms cannot be executed in a sensor node, due to memory requirements and computational power needed for their execution. Indeed, the executable codes are too large to be embedded in tiny sensor nodes. Further, the compression ratios are obtained after collecting all the samples and therefore all the samples have to be stored in memory. This implies that large datasets cannot be managed. In addition, the compression cannot be performed on the fly. Finally, during their execution, these algorithms require a large memory to manage some step of the execution.

Dataset	**Algorithm**	***CR(%)***	***PCR(%)***
LU_ID84_T	S-LZW	48.99	48.98
	gzip	48.87	-
	bzip2	69.24	-
	rar	69.16	-
LU_ID84_H	S-LZW	31.24	31.22
	gzip	37.86	-
	bzip2	57.82	-
	rar	59.03	-

Table 4. Compression ratios obtained by S-LZW, gzip, bzip2 e rar algorithms on the two datasets.

As regards complexity of LEC and S-LZW, we have performed a comparative analysis on the number of instructions required by both the algorithms to compress data. To this aim, we have adopted the Sim-It Arm simulator (Sim-It, 2009), since there already exists a free available version of S-LZW implemented for this simulator by the same authors of this compression algorithm. Sim-It Arm is an instruction-set simulator that runs both system-level and user-level ARM programs. Since S-LZW compresses each dataset block by block, we executed the two algorithms on Sim-It Arm simulator to compress the first block of each dataset. A block consists of 528 bytes (corresponding to 264 samples of 16 bits). Table 5 shows the number of instructions required for compressing one block, the number of saved bits and the number of instructions per saved bit for the temperature and relative humidity datasets, respectively.

We note that, though the LEC algorithm achieves a higher compression ratio than S-LZW, it requires a lower number of instructions. In particular, we observe that, the LEC algorithm executes, for instance, 15.33 instructions for each saved bit against 29.93 executed by S-LZW for compressing the first block of the temperature dataset.

	LEC		**S-LZW**	
	Temperature	**Relative Humidity**	**Temperature**	**Relative Humidity**
number of instructions	44784	62817	63207	63207
number of saved bits	2922	2086	2112	96
number of instructions per saved bit	15.33	30.11	29.93	658.41

Table 5. Complexity of LEC and S-LZW.

3.2 Compression ratio versus correlation between consecutive samples

In the previous section we have adopted the default Huffman table for compressing the two datasets so as to show the effectiveness of the LEC algorithm when dealing with high correlated datasets. In this section, we analyze the behaviour of LEC when the correlation between consecutive samples decreases. To this aim, we performed the following experiment: we simulated different lengths of the sampling interval by downsampling the sequence of data. Since in the original datasets, samples are obtained by measuring temperature and relative humidity at intervals of 30 seconds along 25 days, we considered downsampling factors of 2, 4, 8, 16, 60 and 120, which correspond, respectively, to consider time intervals of 1, 2, 4, 8, 30 and 60 minutes. We expect that, like for all compression algorithms based on differential coding, the sampling rate affects the achievable compression ratio: when the sampling interval is long, the correlation between consecutive samples typically decreases, thus reducing the performance of the LEC algorithm. The significance of this reduction depends on the variability of the signal.

Figure 2 shows the results obtained by compressing the downsampled temperature and relative humidity datasets. We can observe that the compression ratios decrease with the increase of the downsampling factors. For instance, for the temperature, we pass from a compression ratio of 70.81% with the original data (downsampling factor equal to 0) to a compression ratio of 41.38% with a downsampling factor equal to 120. As expected, the decrease of the compression ratios is therefore quite relevant. On the other hand, the Huffman table shown in Table 1 has been proposed for data with high correlation, where high probabilities are associated with differences between consecutive samples very close to 0. Actually, these differences are characterized by a high occurrence frequency. By downsampling the original signal with factors from 16 to 120, this assumption is not true anymore. To be fair in the experiment, we should compute again the occurrence frequencies of the differences between consecutive samples (this approach is known in the literature as semi-adaptive Huffman coding (Salomon, 2007)) and modify appropriately the Huffman table used in the compression.

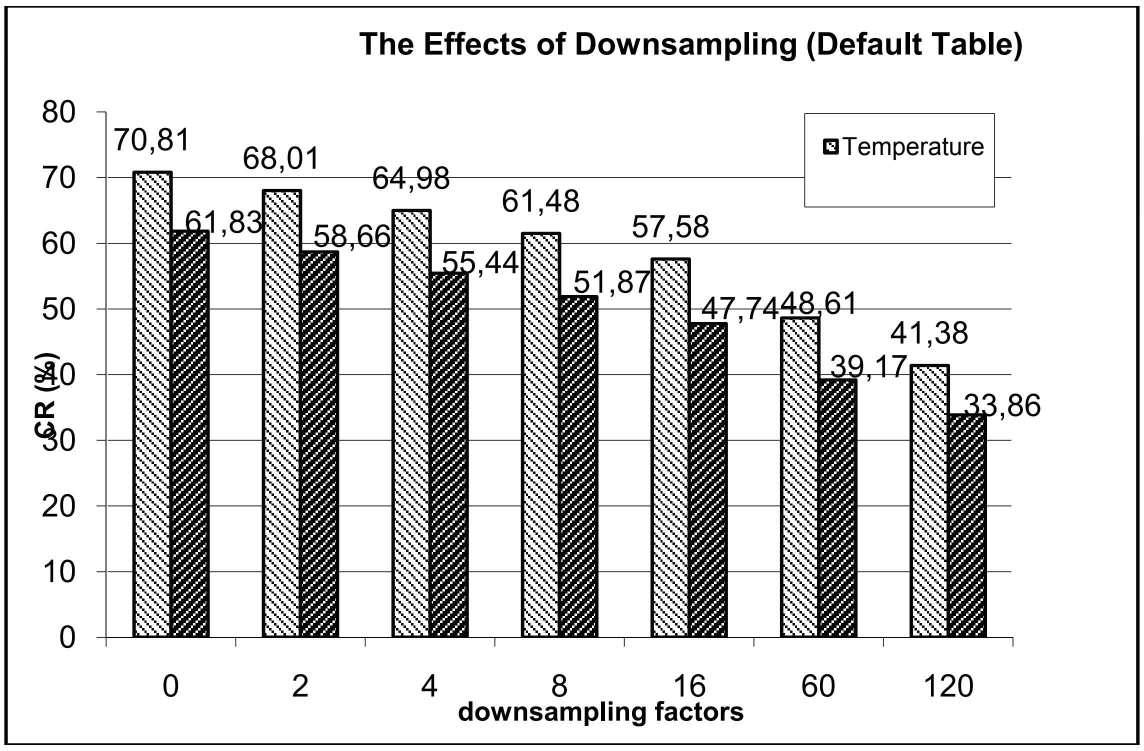

Fig. 2. Compression ratios obtained by using the default Huffman table on the temperature and humidity datasets sampled with different downsampling factors.

Figure 3 shows the results obtained by semi-adaptive Huffman coding. We can observe that the compression ratios still decrease with respect to increasing downsampling factors (on the other hand, the correlation between consecutive samples is lower), but now this decrease is less significant. To take on-line these variations of difference distributions into account, in the literature adaptive Huffman coding has been proposed. The method was originally developed by Faller (Faller, 1973) and Gallager (Gallager, 1978) with substantial improvements by Knuth (Knuth, 1985).

Figure 4 shows the results obtained by using the adaptive Huffman coding in LEC. Obviously, the use of the adaptive coding increases the compression ratios with respect to the use of a fixed table, but does not allow outperforming the use of the semi-adaptive Huffman coding. On the other hand, unlike fixed table and adaptive Huffman coding, semi-adaptive Huffman coding exploits the knowledge of all data. Obviously, this knowledge cannot be assumed in real applications. Thus, the compression ratios obtained by using the semi-adaptive Huffman coding can be considered as an upper limit. However, we observe that the compression ratios achieved with the adaptive Huffman coding are very close to the ones obtained with the semi-adaptive Huffman coding. On the other hand, we have to consider that the use of the adaptive Huffman coding increases the complexity of LEC. Further, since the adaptive coding/decoding scheme is symmetric, a possible loss of one packet makes the decompression process completely unreliable. Thus, in real applications of WSNs the use of a fixed table is certainly desirable and practically mandatory.

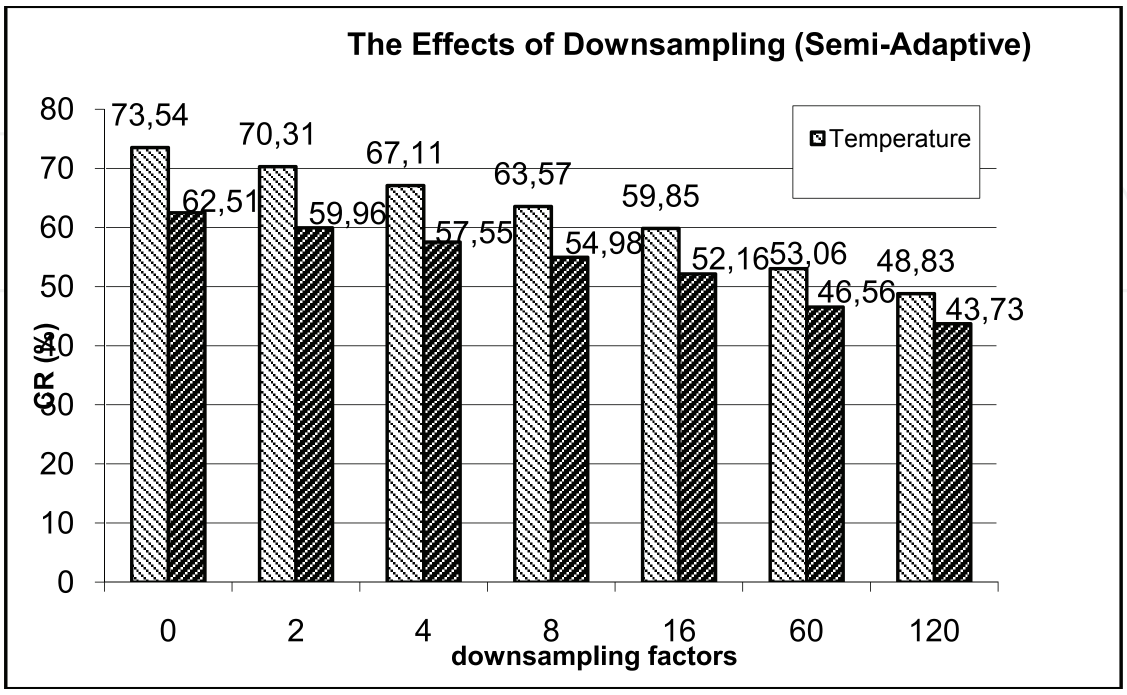

Fig. 3. Compression ratios obtained by using the semi-adaptive Huffman coding on the temperature and humidity datasets sampled with different downsampling factors.

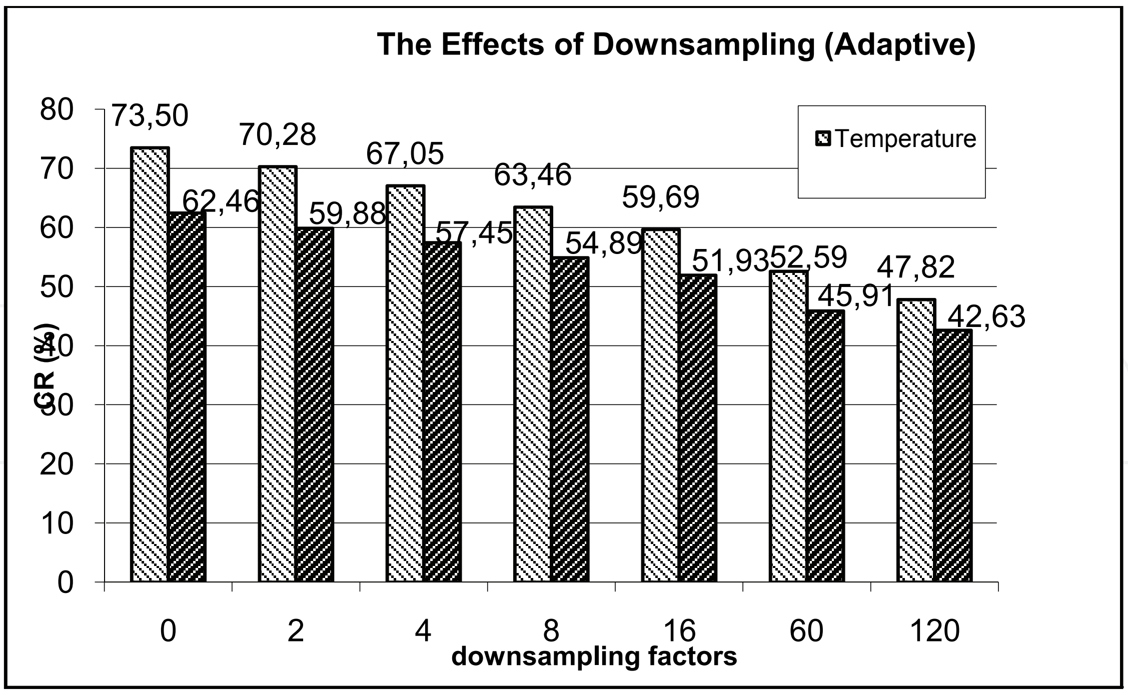

Fig. 4. Compression ratios obtained by using the adaptive Huffman coding on the temperature and humidity datasets sampled with different downsampling factors.

The fixed Huffman table used in the original version of LEC can guarantee satisfactory performance when the correlation between consecutive samples is high. However, when the correlation is not high, we can find a fixed Huffman table suitable for the specific application. Indeed, we would like to remark that, in real habitat monitoring applications, the sampling rate is a parameter of the application domain: once fixed, rarely it is modified. Since the trend of the environmental signals is generally known, this allows us to make quite reliable assumptions on the distributions of the differences, thus permitting us to generate fixed Huffman tables which guarantee high compression ratios. We could also consider to adopt a two-phase approach. In the first phase, we collect an appropriate number of samples so as to perform an analysis of occurrence frequency of the differences. Then, in the second phase, we use the fixed Huffman table generated by the analysis performed in the first phase to compress the data on the fly.
To highlight that the lack of sample correlation does not affect only LEC, but in general all the compression algorithms, we have also applied S-LZW to the temperature and humidity datasets sampled with downsampling factors of 2, 4, 8, 16, 60 and 120. Figure 5 compares the compression ratios obtained by S-LZW with the ones achieved by the LEC algorithm executed by using the default table. As expected, we can observe that also the performance of S-LZW are considerably affected by downsampling.

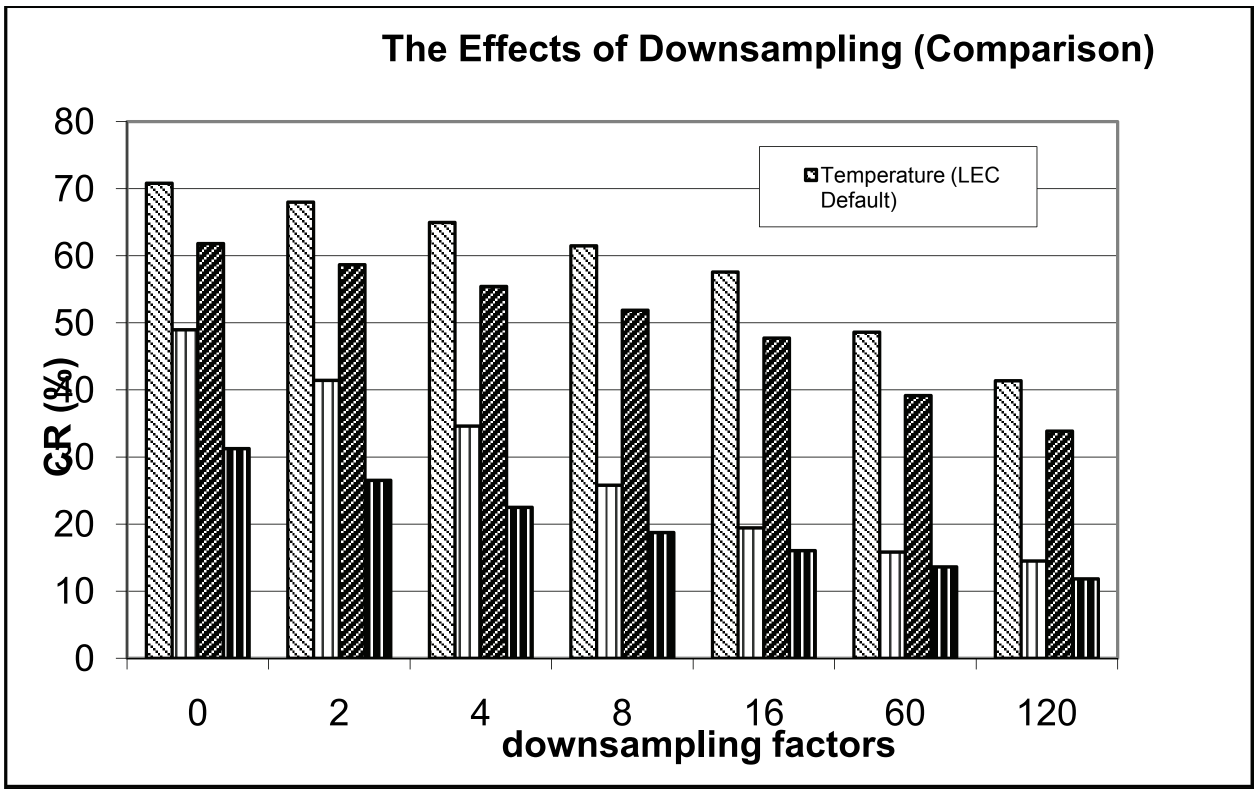

Fig. 5. Comparison between S-LZW and LEC executed with default table on the temperature and humidity datasets sampled with different downsampling factors.

4.3 The problem of the first sample

LEC, as all the differential compression algorithms, suffers from the following problem. In order to reconstruct the original samples, the decoder must know the value of the first

sample: if the first sample has been lost or corrupted, all the other samples are not correctly decoded. In our case, the compressed bitstream is sent by wireless communication to the collector, which takes the decompression process in charge. Since the transmission can be non-reliable, the first packet could be lost and thus also the first value, making correct reconstruction of samples impossible.
To make communication reliable, a number of solutions have been proposed. In general, these solutions involve protocols based on acknowledgements which act at Transport layer. Obviously, these protocols require a higher number of message exchanges between nodes and this increases the power consumption. A review of these algorithms is out of the scope of this chapter. Anyway, a solution to this problem can be also provided at the application layer without modifying the protocols of the underlying layers: when we insert the first sample into the payload of a new packet, we do not insert the difference between the current and the previous sample, but rather the difference between the current sample and a reference value known to the decoder (for instance, the central value of the ADC). Thus, the decoding of each packet is independent of the reception of the previous packets. Table 6 compares the PCRs obtained by using this expedient (this PCR will be denoted as PCR*) with those shown in Table 3: we can note that the decrease of PCR is not high. Further, the PCR*s are still higher than those achieved by S-LZW. Thus, we can conclude that the LEC scheme can be made more robust without significantly affecting its performance.

Dataset	***PCR(%)***	***PCR*(%)***
LU_ID84_T	70.81	68.19
LU_ID84_H	61.83	58.21

Table 6. PCRs obtained without (PCR) and by (PCR*s) transmitting the first value in each packet.

5. From Lossless to Lossy

In some WSN applications, like environmental monitoring, the accurateness of the measures is less important than the sensor cheapness. Thus, often commercial wireless nodes are equipped with sensors which, though cheap, collect measures affected by considerable noise. In this context, the use of lossless compression algorithms can be penalising. Indeed, noise increases the entropy of the signal and therefore hinders the lossless compression algorithm to achieve considerable compression ratios. The ideal solution would be to adopt on the sensor node, a lossy compression algorithm in which the loss of information would be just the noise. Thus, we could achieve high compression ratios without losing relevant information. To this aim, we exploit the observation that data typically collected by WSNs are strongly correlated. Thus, differences between consecutive samples should be regular and generally very small. If this does not occur, it is likely that samples are affected by noise. To de-noise and simultaneously compress the samples, we introduce a lossy version of LEC. In this version, the difference $d_i = r_i - r_{i-1}$ is not directly encoded, but is first quantized and then encoded following the Differential Pulse Code Modulation (DPCM) scheme often used for digital audio signal compression. The schemes of the lossy versions of the compressor and uncompressor are shown in Fig. 6.

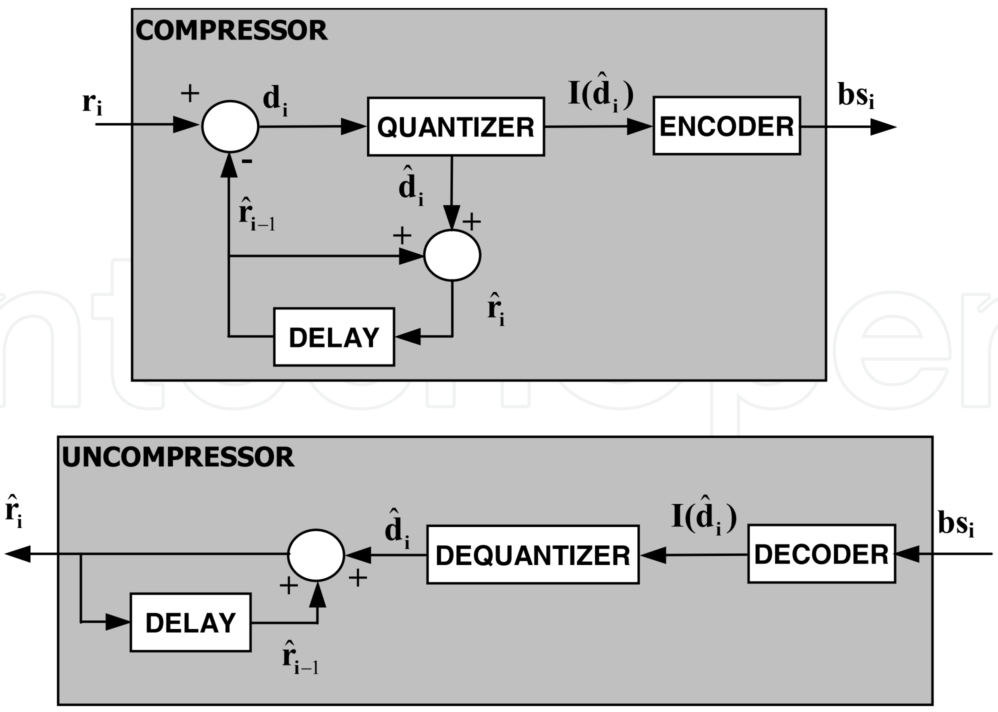

Fig. 6. Block diagram of the encoding/decoding schemes.

Actually to avoid the well-known problem of the *accumulation of the error* (Salomon, 2007), we quantize the difference between sample r_i and the most recent reconstructed value $\hat{r}_{i-1}$. The problem originates from the following consideration: the compressor can compute the exact differences d_i from the original data samples r_i and r_{i-1}, while the uncompressor can work only with quantized differences $\hat{d}_i$. The uncompressor uses $\hat{d}_i$ to generate the reconstructed samples $\hat{r}_i$ ($\hat{r}_i = \hat{r}_{i-1} + \hat{d}_i$) rather than the original samples r_i. The generic *n*th reconstructed sample $\hat{r}_n$ at the uncompressor will contain the sum of the quantization errors accumulated during the reconstruction of the previous *n*-1 samples plus the quantization error of the current sample:

$$\hat{r}_n = r_n + \sum_{i=1}^{n} q_i \tag{3}$$

where q_i is the quantization error.
To overcome this problem, the compressor is modified so as to compute the generic difference $d_i = r_i - \hat{r}_{i-1}$, that is, to calculate difference d_i by subtracting the most recent reconstructed value $\hat{r}_{i-1}$ (which both the compressor and the uncompressor have) from the current original sample r_i. Thus, the uncompressor first decodifies r_0. Then, when it receives the first quantized difference $\hat{d}_1$, it computes $\hat{r}_1 = r_0 + \hat{d}_1 = r_0 + d_1 + q_1 = r_1 + q_1$. When it receives the second quantized difference $\hat{d}_2$, it computes $\hat{r}_2 = \hat{r}_1 + \hat{d}_2 = \hat{r}_1 + d_2 + q_2 = \hat{r}_1 + r_2 - \hat{r}_1 + q_2 = r_2 + q_2$. The decoded value $\hat{r}_2$ contains just the single quantization error q_2, and in general, the decoded value $\hat{r}_i$ contains just the quantization error q_i.

Difference d_i is input to the block QUANTIZER that outputs the quantization level $\hat{d}_i$ assigned to d_i and the index $I\left(\hat{d}_i\right)$ of $\hat{d}_i$. The index $I\left(\hat{d}_i\right)$ is input to the ENCODER block, which generates the codeword bs_i using the same bijection defined in (1) for mapping integer inputs to natural values, and the same combination of unary and binary codes described in Section 2. The ENCODER block, therefore, encodes the quantization index corresponding to the quantized difference rather than the difference as in LEC. Again, the dictionary table used to produce the codes should be generated based on the occurrence frequency of the quantization indexes. In these preliminary experiments, we have decided to adopt the same dictionary used in Table 1, where in place of d_i, the reader should read $\hat{d}_i$. Since the number of quantization levels $\hat{d}_i$ is lower than the number of possible d_i, the table might have a lower number of entries.
In the uncompressor, the codeword bs_i is analyzed by the DECODER block which outputs the index $I\left(\hat{d}_i\right)$, exploiting the same dictionary table. This index is elaborated by the block DEQUANTIZER to produce $\hat{d}_i$ which is added to $\hat{r}_{i-1}$ to output $\hat{r}_i$.
Currently, we are simply adopting a uniform quantization. In this case, the unique parameter to be fixed is the difference *D* between two consecutive levels. This parameter is very important because it affects the value of the quantization error and indirectly the compression ratio. To show the performance of the lossy version of LEC, we set *D* to six different values: 10%, 20%, 30%, 40%, 50% and 60% of the Manufactured Error (ME) of the sensor used to collect data. In the case of the sensors (Sensirion SHT75) used in our experiments, ME = ± 0.3 ^{o}C and ME = ± 1.8% for temperature and relative humidity, respectively (Sensirion, 2009). Table 7 shows the compression ratios and the root mean squared errors (RMSEs) obtained on the temperature and relative humidity datasets.
RMSE is computed as:

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(r_i - \hat{r}_i\right)^2} \tag{5}$$

where r_i is the original sample, $\hat{r}_i$ is the reconstructed sample and *N* is the number of samples of the signal. We observe that, as expected, the compression ratios are higher than the ones obtained by the original version of LEC. On the other hand, the lossy version introduces an error on the reconstructed signal. Anyway, this error is lower than ME, which represents a sort of uncertainty of the measure.
To assess the results shown in Table 7, we have applied LTC to the same datasets. LTC is an efficient and simple lossy compression technique for the context of habitat monitoring. LTC generates a set of line segments which form a piecewise continuous function. This function approximates the original dataset in such a way that no original sample is farther than a fixed error *e* from the closest line segment. Thus, before executing the LTC algorithm, we have to set error *e*. To perform a fair comparison with the lossy version of LEC, we have set *e* to the 10%, 20% and 30% of the ME of the sensor. This allows obtaining RMSEs comparable with the ones obtained by the lossy version of LEC when *D* is equal to the 20%, 40% and 60% of the ME. Table 8 shows the compression ratios and the RMSEs obtained on the

temperature and relative humidity datasets. We can observe that the lossy version of LEC outperforms LTC in terms of CR for comparable RMSEs, thus proving the good characteristics of the proposed lossy compression algorithm.

Dataset	**Algorithm**	***CR(%)***	***RMSE***
LU_ID84_T	0.1 ·ME	78.18	0.0082
	0.2 ·ME	81.26	0.0171
	0.3 ·ME	83.45	0.0256
	0.4 ·ME	83.46	0.0353
	0.5 ·ME	84.94	0.0428
	0.6 ·ME	86.14	0.0517
LU_ID84_H	0.1 ·ME	74.65	0.0450
	0.2 ·ME	78.83	0.0872
	0.3 ·ME	80.89	0.1296
	0.4 ·ME	82.13	0.1721
	0.5 ·ME	82.97	0.2166
	0.6 ·ME	83.61	0.2598

Table 7. Compression ratios obtained by the lossy version of LEC on the two datasets.

Dataset	**Algorithm**	***CR(%)***	***RMSE***
LU_ID84_T	0.1 ·ME	55.00	0.0190
	0.2 ·ME	77.53	0.0348
	0.3 ·ME	86.12	0.0502
LU_ID84_H	0.1 ·ME	26.49	0.0824
	0.2 ·ME	55.97	0.1681
	0.3 ·ME	70.99	0.2496

Table 8. Compression ratios obtained by the LTC algorithm on the two datasets.

6. Conclusions

In this chapter, we have discussed how enabling compression helps in wireless sensor nodes. First, we have briefly introduced LEC, a lossless compression algorithm we proposed in a previous paper. LEC divides the alphabet of differences between consecutive samples into groups whose sizes increase exponentially. Each codeword is a hybrid of unary and binary codes: in particular, the unary code (a variable-length code) specifies the group, while the binary code (a fixed-length code) represents the index within the group. In the original version, we used the Huffman table proposed in JPEG for coding the groups. Here, we have investigated semi-adaptive and adaptive Huffman coding and carried out a comparison in terms of compression ratios with the LEC algorithm with fixed Huffman table. We have shown that semi-adaptive and adaptive Huffman coding can increase the compression ratios when the correlation between consecutive samples decreases. We have compared all the approaches with S-LZW, a compression algorithm specifically proposed for sensor nodes, and with three classical compression algorithms, namely gzip, bzip2 and rar, though these algorithms are not embeddable in tiny sensor nodes. We have shown that the different versions of LEC can achieve considerable compression ratios in all the datasets considered in the experiments. Finally, we have discussed how LEC can be transformed into a lossy compression algorithm and have shown that this lossy version outperforms LTC, a lossy compression algorithm specifically designed for being embedded in tiny sensor nodes.

7. Acknowledgements

This work was supported by the Italian Ministry of University and Research (MIUR) under the PRIN project #2005090483_005 “Wireless sensor networks for monitoring natural phenomena” and the FIRB project “Adaptive Infrastructure for Decentralized Organization (ArtDecO)”.

8. References

Anastasi, G., Conti, M., Di Francesco, M. & Passarella, A. (2009) Energy conservation in wireless sensor networks: A survey. *Ad Hoc Networks*, Vol. 7, 537-568.

Barr, K. C. and Asanović, K. (2006) Energy-aware lossless data compression. *ACM Trans. Comput. Syst.*, Vol. 24, 250-291.

Boulis, A., Ganeriwal, S. & Srivastava, M.B. (2003) Aggregation in sensor networks: an energy– trade-off. *Ad Hoc Networks*, Vol. 1, 317–331.

Chen, H., Li, J. & Mohapatra, P. (2004) RACE: time series compression with rate adaptivity and error bound for sensor networks. Proceedings of the First IEEE International Conference on Mobile Ad-hoc and Sensor Systems, pp. 124-133, Fort Lauderdale, FL, USA, 24-27 October,. IEEE, Piscataway, NJ, USA.

Ciancio, A. & Ortega, A. (2005) A distributed wavelet compression algorithm for wireless multihop sensor networks using lifting. Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing, pp. 825-828, Philadelphia, PA, USA, 18-23 March,. IEEE, Piscataway, NJ, USA.

Ciancio, A., Pattem, S., Ortega, A. & Krishnamachari, B. (2006) Energy-efficient data representation and routing for wireless sensor networks based on a distributed wavelet compression algorithm. Proceedings of the Fifth international Conference on Information Processing in Sensor Networks, pp. 309-316, Nashville, TN, USA, 19-21 April, ACM, New York, NY, USA.

Croce, S., Marcelloni, F. & Vecchio, M. (2008) Reducing power consumption in wireless sensor networks using a novel approach to data aggregation. *The Computer Journal*, Vol. 51, No. 2, 227–239.

Deligiannakis, A., Kotidis, Y. & Roussopoulos, N. (2004) Compressing historical information in sensor networks. Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data, pp. 527-538, Paris, France, 13-18 June, ACM, New York, NY, USA.

Di Bacco, G., Melodia, T. & Cuomo, F. (2004) A MAC protocol for delay-bounded applications in wireless sensor networks. Proceedings of the Third Annual Mediterranean Ad Hoc Networking Workshop, pp. 208-220, Bodrum, Turkey, 27-30 June, available on-line: http://www.ece.osu.edu/medhoc04/.

Elias, P. (1975) Universal codeword sets and representations of the integers. *IEEE Transaction on Information Theory*, Vol. 21, No. 2, 194–203.

Faller, N. (1973) An adaptive system for data compression. Proceedings of the 7th Asilomar Conference on Circuits, Systems, and Computers, pp. 593–597, Pacific Grove, CA, USA, November, IEEE Press, Piscataway, NJ, USA.

Fan, K-W, Liu, S. & Sinha, P. (2007) Structure-free data aggregation in sensor networks. *IEEE Transactions on Mobile Computing*, Vol. 6, 929-942.

Fasolo, E., Rossi, M., Widmer, J. & Zorzi, M. (2007) In-network aggregation techniques for wireless sensor networks: a survey. *Wireless Communications*, Vol. 14, 70-87.

Gallager, R. G. (1978) Variations on a theme by Huffman. *IEEE Transactions on Information Theory*, Vol. 24, No. 6, 668-674.

Ganesan, D., Estrin, D. & Heidemann, J. (2003) DIMENSIONS: why do we need a new data handling architecture for sensor networks?, *SIGCOMM Comput. Commun. Rev.*, Vol. 33, 143-148.

Gastpar, M., Dragotti, P. L. & Vetterli, M. (2006) The distributed Karhunen-Loève transform. *IEEE Transactions on Information Theory*, Vol. 52, No. 12, 5177-5196.

Girod, B., Aaron, A., Rane, S. & Rebollo-Monedero, D. (2005) Distributed video coding. *Proceedings of the IEEE*, Special Issue Advances Video Coding, Delivery, Vol. 93, No. 1, 71–83.

Golomb, S. W. (1966) Run-length encodings. *IEEE Transactions on Information Theory*, Vol. 12, No. 3, 399-401.

Guestrin, C., Bodi, P., Thibau, R., Paski, M. & Madden, S. (2004) Distributed regression: an efficient framework for modelling sensor network data. Proceedings of the Third International Symposium on Information Processing in Sensor Networks, pp.1-10, Berkeley, CA, USA, 26-27 April, ACM, New York, NY, USA.

Intanagonwiwat, C., Govindan, R., Estrin, D., Heidemann, J. & Silva, F. (2003) Directed diffusion for wireless sensor networking. *IEEE/ACM Trans. Netw.*, Vol. 11, 2-16.

Kimura, N. & Latifi, S. (2005) A survey on data compression in wireless sensor networks. Proceedings of the International Conference on Information Technology: Coding and Computing, pp. 8-13, Las Vegas, NV, USA, 4-6 April, IEEE Computer Society, Washington, DC, USA.

Knuth, D. E. (1985) Dynamic Huffman Coding. *Journal of Algorithms*, Vol. 6, 163-180.

Lin, S., Kalogeraki, V., Gunopulos, D. & Lonardi, S. (2006) Online information compression in sensor networks. Proceedings of the IEEE International Conference on Communications, pp. 3371-3376, Istanbul, Turkey, 11-15 June, IEEE Press, Piscataway, NJ, USA.

Lindsey, S., Raghavendra, C. & Sivalingam, K. M. (2002) Data gathering algorithms in sensor networks using energy metrics. *IEEE Trans. Parallel Distrib. Syst.*, Vol. 13, 924-935.

Lynch, J. P., Wang, Y., Sundararajan, A., Law, K. H. & Kiremidjian, A. S. (2004) Wireless sensing for structural health monitoring of civil structures. Proceedings of the International Workshop on Integrated Life-Cycle Management of Infrastructures, Hong Kong, 9-11 December.

LZO (2008) http://www.oberhumer.com/opensource/lzo/

Madden, S., Franklin, M. J., Hellerstein, J. M. & Hong, W. (2002) TAG: a Tiny AGgregation service for ad-hoc sensor networks. *SIGOPS Oper. Syst. Rev.*, Vol. 36, 131-146.

Mainwaring, A., Culler, D., Polastre, J., Szewczyk, R. & Anderson, J. (2002) Wireless sensor networks for habitat monitoring. Proceedings of the First ACM international Workshop on Wireless Sensor Networks and Applications, pp. 88-97, Atlanta, GA, USA, 28 September, ACM, New York, NY, USA.

Marcelloni, F. & Vecchio, M. (2009) An efficient lossless compression algorithm for tiny nodes of monitoring wireless sensor networks, *The Computer Journal*, Section B: Networks and Computer Systems, Advance Access, doi:10.1093/comjnl/bxp035.

Pennebaker, W. B. & Mitchell, J. L. (1992) *JPEG still image data compression standard*. Kluwer Academic Publishers, Norwell, MA, USA.

Pradhan, S. Kusuma, J. & Ramchandran, K. (2002) Distributed compression in a dense microsensor network, *IEEE Signal Processing Mag.*, Vol. 19, 51–60.

Rebollo-Monedero, D. (2007) Quantization and transforms for distributed source coding. PhD thesis The Department of Electrical Engineering and the Committee on Graduate Studies of Stanford University.

Sadler, C. M. & Martonosi, M. (2006) Data compression algorithms for energy-constrained devices in delay tolerant networks. Proceedings of the 4th ACM International Conference on Embedded networked sensor systems, pp. 265-278, Boulder, Colorado, USA, October 31 - November 3, ACM, New York, NY, USA.

Salomon, D. (2007) *Data Compression: The Complete Reference*, Springer Verlag, London, UK.

Schoellhammer, T., Osterweil, E., Greenstein, B., Wimbrow, M. & Estrin, D. (2004) Lightweight temporal compression of microclimate datasets. Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks, pp. 516-524, Tampa, FL, USA, 16-18 November, IEEE Computer Society, Washington, DC, USA.

Sensirion (2009) Sensirion homepage, www.sensirion.com

SensorScope (2009), SensorScope deployments homepage http://sensorscope.epfl.ch/index.php/Main_Page

Sim-It (2009) Sim-It Simulator homepage, http://simit-arm.sourceforge.net/

Tang, C. & Raghavendra, C. S. (2004) Compression techniques for wireless sensor networks. In: *Wireless Sensor Networks*, Raghavendra, C. S., Sivalingam, K. M. and Znati, T. (Ed.), Kluwer Academic Publishers, Norwell, MA, USA.

Teuhola, J. (1978) A Compression Method for Clustered Bit-Vectors. *Information Processing Letters*, Vol. 7, 308-311.

TinyNode (2009) TinyNode homepage, http://www.tinynode.com

Wagner, R. S., Baraniuk, R. G., Du, S., Johnson, D. B. & Cohen, A. (2006) An architecture for distributed wavelet analysis and processing in sensor networks. Proceedings of the Fifth International Conference on Information Processing in Sensor Networks, pp. 243-250, Nashville, TN, USA, 19-21 April, ACM, New York, NY, USA.

Welch, T.A. (1984) A technique for high-performance data compression, *Computer*, Vol. 17, 8-19.

Ziv, J. & Lempel, A. (1977) A universal algorithm for sequential data compression, *IEEE Transactions on Information Theory*, Vol. 23, 337-343.

Zixiang, X., Liveris, A.D. & Cheng, S. (2004) Distributed source coding for sensor networks. *IEEE Signal Processing Magazine*, Vol. 21, No. 5, 80-94.