A Simple Algorithm for Data Compression in Wireless Sensor Networks

Francesco Marcelloni, *Member, IEEE,* and Massimo Vecchio, *Member, IEEE*

***Abstract*— Power saving is a critical issue in wireless sensor networks (WSNs) since sensor nodes are powered by batteries which cannot be generally changed or recharged. As radio communication is often the main cause of energy consumption, extension of sensor node lifetime is generally achieved by reducing transmissions/receptions of data, for instance through data compression. Exploiting the natural correlation that exists in data typically collected by WSNs and the principles of entropy compression, in this Letter we propose a simple and efficient data compression algorithm particularly suited to be used on available commercial nodes of a WSN, where energy, memory and computational resources are very limited. Some experimental results and comparisons with, to the best of our knowledge, the only lossless compression algorithm previously proposed in the literature to be embedded in sensor nodes and with two well-known compression algorithms are shown and discussed.**

***Index Terms*— Wireless sensor networks, data communication, signal processing, energy efficiency.**

I. Introduction

ENERGY is a primary constraint in the deployment of WSNs, since sensor nodes are typically powered by small batteries, which cannot be generally changed or recharged. In most cases, the radio transceiver onboard sensor nodes is the main cause of energy consumption. Thus, power saving is generally achieved by reducing radio communication through mainly two approaches: duty cycling [1] and in-network processing [2]. Duty cycling schemes define coordinated sleep/wakeup schedules among nodes in the network. On the other hand, in-network processing consists in reducing the amount of data to be transmitted by means of compression [3] and/or aggregation techniques [4]. Due to limited processing and storage resources of sensor nodes, data compression in sensor nodes requires the use of ad-hoc algorithms. Only a few papers have discussed the possibility of embedding lossless compression algorithms into sensor nodes. Actually, this solution is very effective in sparse WSNs consisting of a number of static sensor nodes, placed in pre-fixed points in an area under monitoring, and one or more data collectors, which come into contact with the static sensors at approximately regular intervals and collect values measured by them. This model is characterised by a number of advantages in comparison with the traditional approach based on multi-hop communication: network lifetime and network capacity increase, while packet loss probability and node synchronization error decrease. On the other hand, data latency and costs of the network infrastructure might increase [5]. Since this model is typically applied in environmental monitoring applications, latency is not an issue, while the additional cost for data collectors can be maintained low exploiting the mobility of external agents.

Obviously, compressing data can be a valuable help in power saving only if the execution of compression algorithms does not require an amount of energy greater than the one saved in reducing transmission: in [6] it is shown that compression prior to transmission in wireless battery-powered devices may actually cause an overall increase of power consumption, if no energy awareness is introduced, because compression algorithms are aimed at saving storage and not energy.

We propose a simple compression algorithm particularly suited to the reduced memory and computational resources of a WSN node. The compression scheme exploits the high correlation that typically exists between consecutive samples collected by a sensor onboard a node. Thanks to this characteristic and following the principles of entropy compression [6], the algorithm is able to compute a compressed version of each value acquired from a sensor on-the-fly, using a very small dictionary whose size is determined by the resolution of the analog-to-digital converter (ADC). Finally, the compression algorithm is lossless. To achieve these goals, we have followed a scheme similar to the one used by the baseline JPEG algorithm for compressing the *DC coefficients* of a digital image [7]. Indeed, such coefficients are characterized by a high correlation, very similar to that characterizing data collected by WSNs.

II. The Compression Algorithm

In 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. For each new acquisition m_i, the compression algorithm 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. Each d_i 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 and a_i is the representation of d_i. If $d_i = 0$, then $n_i = 0$, else $n_i = \lceil log_2(|d_i|) \rceil$. Thus, at most n_i is equal to R. Code s_i is a variable-length symbol generated from n_i by using Huffman coding. The basic idea of Huffman coding is to map an alphabet to a representation

Manuscript received February 27, 2008. The associate editor coordinating the review of this letter and approving it for publication was C. Charalambous. 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)".

F. Marcelloni is with the Dipartimento di Ingegneria dell'Informazione, University of Pisa, Via Diotisalvi 2, 56122 Pisa, Italy (e-mail: f.marcelloni@iet.unipi.it).

M. Vecchio is with the IMT Lucca Institute for Advanced Studies, P.zza S. Ponziano 6, 55100 Lucca, Italy (e-mail: m.vecchio@imtlucca.it).

Digital Object Identifier 10.1109/LCOMM.2008.080300.

TABLE I
THE HUFFMAN VARIABLE LENGTH CODES USED IN THE EXPERIMENTS.

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

```
encode(di,Table)
  IF di = 0 THEN
    SET ni TO 0
  ELSE
    SET ni TO ⌈log2(|di|)⌉     //compute category
  ENDIF
  SET si TO Table[ni]          //extract si from Table
  IF ni = 0 THEN               //build bsi
    SET bsi TO si              //ai is not needed
  ELSE
    IF di > 0 THEN             //build ai
      SET ai TO (di)|ni
    ELSE
      SET ai TO (di − 1)|ni
    ENDIF
    SET bsi TO ≪ si,ai ≫       // build bsi
  ENDIF
RETURN bsi
```

Fig. 1. Pseudo-code of the *encode* algorithm.

for that alphabet, composed of sequences of bits of variable sizes, so that symbols that occur frequently have a smaller representation than those that occur rarely. In our case, the symbols are $R+1$ and the probabilities decrease with the increase of the values. Table I shows the set of Huffman codes used in our experiments. The first 11 lines coincide with the table used in the baseline JPEG algorithm for compressing the DC coefficients. On the other hand, these coefficients have statistical characteristics similar to the measures acquired by sensor nodes.

The a_i part of the bit sequence bs_i is a variable-length integer code generated as follows:

1) if $d_i > 0$, a_i corresponds to the n_i low-order bits of the two's complement representation of d_i;
2) if $d_i < 0$, a_i corresponds to the n_i low-order bits of the two's complement representation of $(d_i - 1)$;
3) if $d_i = 0$, s_i is coded as 00 and a_i is not represented.

The procedure used to generate a_i guarantees that all possible values have different codes. Once bs_i is generated, it is appended to the bitstream which forms the compressed version of the sequence of measures m_i. Fig. 1 summarises the algorithm used to encode d_i. Here, $\ll si, ai \gg$ denotes the concatenation of si and ai, while $v|_{ni}$ denotes the n_i low-order bits of v.

We observe that the compression algorithm described in Fig. 1 is very simple (it can be implemented in a few lines of code) and requires only to maintain the first two columns of Table I in memory. On simulations performed on Avrora [8], an instruction-level sensor network simulator, we have observed that the execution of the algorithm requires from 59 to 618 instructions. This difference is due to the execution of the binary logarithm, which is not generally available in the instruction set of microprocessors onboard sensor nodes and is, therefore, implemented by an algorithm that requires the execution of a number of instructions dependent on the value of the argument (the larger the value, the larger the number of instructions). For the environmental data collected in the experiments discussed in the next section, we verified that the execution of the algorithm in Fig. 1 requires in average 355 instructions. Since transmission of a bit needs an energy comparable to the execution of a thousand instructions, just saving only a bit by compressing original data corresponds to reduce power consumption.

TABLE II
RESULTS OBTAINED BY APPLYING THE PROPOSED COMPRESSION ALGORITHM.

	Temperature	Relative humidity
orig_size	23040 bits	23040 bits
comp_size	7605 bits	7527 bits
comp_ratio	66.99%	67.33%

III. EXPERIMENTAL RESULTS

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

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

where *comp_size* and *orig_size* are, respectively, the size of the compressed and the uncompressed bitstream. We deployed a WSN composed of *Moteiv*'s Tmote sky [9] sensor nodes: each node uses a Sensirion SHT11 module [10] to measure temperature and relative humidity. The module is in its turn coupled with a 14-bit ADC converter. From the datasheets of the SHT11, the default measurement resolutions result to be 14-bit and 12-bit for, respectively, temperature and relative humidity. In our experiment, we consider samples acquired by a node every 2 minutes during 48 hours (in total, 1440 samples).

Table II summarises the results obtained by applying the compression algorithm. We note that, though the compression algorithm is very simple, it is able to obtain considerable compression ratios.

Let us assume that all samples have to be transmitted to the data collector by using the lowest number of packets. Considering that uncompressed samples are normally byte-aligned, both temperature and relative humidity samples are represented by 16-bit unsigned integers. Supposing that each packet can contain at most 25 bytes of payload [11], the uncompressed versions of temperature and relative humidity data require 116 packets each to be forwarded to the data collector, while the compressed versions require only 38 packets each, thus allowing a considerable power saving.

To assess the goodness of our algorithm, we compared these results with a dictionary-based lossless compression

TABLE III
RESULTS OBTAINED BY APPLYING S-LZW, GZIP AND BZIP2 ALGORITHMS.

	Temperature		Relative humidity	
	comp_size	*comp_ratio*	*comp_size*	*comp_ratio*
S-LZW	16760 bits	27.25%	13232 bits	42.57%
gzip	15960 bits	30.73%	13320 bits	42.19%
bzip2	15992 bits	30.59%	13120 bits	43.05%

algorithm, called S-LZW, recently proposed in the literature [3]. To the best of our knowledge, this is the only example of lossless compression algorithm specifically designed for sensor nodes. S-LZW is a version of the famous Lempel-Ziv-Welch (LZW [12]) compression algorithm. S-LZW splits the uncompressed input bitstream into fixed size blocks and then compresses separately each block. For each new block, the dictionary used in the compression is re-initialised 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 is 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. Finally, to take specific advantage of the sensor data repetitiveness, 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 is used to pre-process the raw data so as to build structured datasets, which allow the S-LZW algorithm to perform better. In our experiments, we adopted the parameter values suggested in [3]: a block size of 528 bytes, a dictionary of 512 entries which is frozen as-is when it gets full, a mini-cache of 32 entries and structured datasets. The first row of Table III summarises the results obtained by S-LZW in compressing the temperature and relative humidity datasets. By comparing these results with Table II, we observe that our algorithm considerably outperforms S-LZW in terms of compression ratio. In the assumption of packets containing at most 25 bytes of payload, the use of S-LZW would allow reducing the number of messages from 116 to 84 (against 38 of our algorithm) for temperature data and from 116 to 67 (against 38 of our algorithm) for relative humidity data. With the aim to compare the two algorithms also in terms of computational complexity, we adopted the SimIt StrongArm simulator [13], since there already existed a free available version of S-LZW implemented for this simulator. The average number of instructions required by our algorithm to compress temperature and relative humidity data was 42% of the average number of instructions required by S-LZW. Thus, we can conclude that our algorithm overcomes S-LZW in terms of both compression ratio and computational complexity.

Finally, to further highlight the good characteristics of our algorithm in terms of compression ratios, in Table III we also show the results obtained by applying two well-known compression algorithms, namely gzip [14] and bzip2 [15], to the same datasets. By comparing these results with Table II, we can observe how our algorithm outperforms both gzip and bzip2. On the other hand, as already proved in [3] [6], both the algorithms are not portable to sensor nodes because they have memory and processing requirements that far exceed what processing unit onboard nodes can provide.

IV. CONCLUSIONS

We have introduced a simple lossless compression algorithm particularly suited to the reduced storage and computational resources of a WSN node. We have evaluated the algorithm by compressing temperature and relative humidity data collected by a real WSN. We have obtained compression ratios of 66.99% and 67.33% for temperature and relative humidity datasets, respectively. Then, we have compared this algorithm with S-LZW, a lossless compression algorithm previously introduced in the literature for compressing data in WSNs. We have shown that our algorithm can achieve higher compression ratios than S-LZW, despite a lower memory occupation and a less computational effort. Finally, we have also shown that our algorithm outperforms gzip and bzip2.

REFERENCES

[1] G. Anastasi, M. Conti, M. Di Francesco, and A. Passarella, "How to prolong the lifetime of wireless sensor networks," in *Mobile Ad Hoc and Pervasive Communications*, M. Denko and L. Yang, eds. Valencia, CA: American Scientific Publishers, to be published.
[2] E. Fasolo, M. Rossi, J. Widmer, and M. Zorzi, "In-network aggregation techniques for wireless sensor networks: a survey," *IEEE Trans. Wireless Commun.*, vol. 14, no. 2, pp. 70–87, Apr. 2007.
[3] C. M. Sadler and M. Martonosi, "Data compression algorithms for energy-constrained devices in delay tolerant networks," in *Proc. SenSys '06: 4th Int. Conference on Embedded networked sensor systems*, 2006, pp. 265–278.
[4] S. Croce, F. Marcelloni, and M. Vecchio, "Reducing power consumption in wireless sensor networks using a novel approach to data aggregation," *The Computer J.*, vol. 51, no. 2, pp. 227–239, 2008.
[5] A. A. Somasundara, A. Kansal, D. D. Jea, D. Estrin, and M. B. Srivastava, "Controllably mobile infrastructure for low energy embedded networks," *IEEE Trans. Mobile Comput.*, vol. 5, no. 8, pp. 958–973, Aug. 2006.
[6] K. C. Barr and K. Asanović, "Energy-aware lossless data compression," *ACM Trans. Comput. Syst.*, vol. 24, no. 3, pp. 250–291, Aug. 2006.
[7] W. B. Pennebaker and J. L. Mitchell, *JPEG Still Image Data Compression Standard*, 2nd ed. Norwell, MA: Kluwer Academic Publishers, 1992.
[8] Avrora. [Online]. Available: http://compilers.cs.ucla.edu/avrora/
[9] Sentilla Corporation. [Online]. Available: http://www.sentilla.com
[10] Sensirion. [Online]. Available: http://www.sensirion.com
[11] A. Mainwaring, D. Culler, J. Polastre, R. Szewczyk, and J. Anderson, "Wireless sensor networks for habitat monitoring," in *Proc. WSNA '02: 1st ACM Int. Workshop on Wireless Sensor Networks and Applications*, 2002, pp. 88–97.
[12] T. A. Welch, "A technique for high-performance data compression," *Computer*, vol. 17, no. 6, pp. 8–19, June 1984.
[13] SimIt-ARM. [Online]. Available: http://simit-arm.sourceforge.net/
[14] Gzip Home Page. [Online]. Available: http://www.gzip.org/
[15] Bzip2 Home Page. [Online]. Available: http://www.bzip.org/