RAKE: a Simple and Efficient Lossless Compression Algorithm for the Internet of Things

Giuseppe Campobello[1], Antonino Segreto[1], Sarah Zanafi[2], Salvatore Serrano[1]
[1]Department of Engineering - University of Messina (Italy)
[2]Faculty of Science Tetouan, University Abdelmalek Essaadi (Morocco)

***Abstract*—In this paper we propose a new lossless compression algorithm suitable for Internet of Things (IoT). The proposed algorithm, named RAKE, is based only on elementary counting operations and has low memory requirements, and therefore it can be easily implemented in low-cost and low-speed micro-controllers as those used in IoT devices. Despite its simplicity, simulation results show that, in the case of sparse sequences, the proposed algorithm outperforms well-known lossless compression algorithms such as rar, gzip and bzip2. Moreover, in the case of real-world data, RAKE achieves higher compression ratios as even compared to IoT-specific lossless compression algorithms.**

I. Introduction

Compression techniques are widely used in Internet of Things (IoT) because they allow to reduce storage and bandwidth resources [1], [2] and, moreover, to increase lifetime of battery powered devices [3] [4].

In general, despite lossy compression algorithms allow to achieve much higher compression ratios, lossless compression algorithms are more widespread in several IoT scenarios [5]. For instance this is the case of IoT devices developed for biomedical and health-related signals where it is necessary to ensure that medically important details are not lost causing errors in medical diagnosis [6].

Although several lossless compression algorithms exist, for instance the well-known Lempel-Ziv algorithm [7], most of them are not suitable when only limited storage and computational resources are available [8]. This is the case of wireless sensor networks (WSN), one of the key enablers of the IoT paradigm, where nodes are based on low-speed microcontrollers with just a few kilobytes of memory [9].

In this paper we present a simple and effective lossless compression algorithm, henceforward named RAKE, that is able to outperform conventional lossless compression algorithms. Considering its inherent low complexity and memory requirements, RAKE is well suited for low-cost micro-controller and embedded devices as those used in IoT.

Simulation results show that in the case of sparse binary sequences (i.e. sequences with many fewer ones than zeros or vice versa) the proposed compression algorithm well approaches the Shannon's entropy bound [10] achieving higher compression ratios as compared to well-known (and more complex) lossless compression algorithms such as gzip [11], rar [12] and bzip2 [13].

Moreover, also in the case of real-world data, RAKE achieves higher compression ratios as even compared to IoT-specific lossless compression algorithms.

The proposed algorithm exhibits several other advantages:

- *Simplicity:* its implementation requires only elementary counting operations;
- *Limited Overhead:* only few additional bits are needed for encoding all parameters needed for decompression;
- *Restrained Storage:* RAKE does not rely on pre-stored codewords: actually, only the original sequence and the encoded sequence must be stored;
- *Generality:* Despite RAKE has been thought primarily for compression of sparse binary sequence, we show that it can be easily extended to sequence of integers.

The rest of this paper is organized as follows: in Sec. II related works are discussed; in Sec. III the basic idea of RAKE is introduced; in Sec. IV performance of RAKE and other lossless compression schemes are compared. Finally, conclusions and future works are drawn in Sec. V.

II. Related works

When lossless algorithms are considered, the common approach is to exploit temporal correlation and a simple method is to use differences between two consecutive samples (commonly called residues). As shown in [14] and references therein, residues of different real-world data (temperature, humidity, solar radiation, etc.) fit well with zero-mean Gaussian or Laplace distributions. Consequently, the basic idea behind several compression algorithms is to encode residues by using a dictionary optimized on the basis of preliminary information about data distributions.

Examples of dictionary-based approaches are:

- *S-LZW* [15] where the authors simplify the well-known algorithm of Lempel-Ziv-Welch by taking into account limited resources of sensor nodes. Basically, the algorithm divides data into blocks of fixed size, and then separately compresses each block by using a dictionary.
- *SHuffman* [16], where for each new sampled value, x_i, the residue $r_i = x_i - x_{i-1}$ is calculated and encoded on the basis of a variable length code. The SHuffman dictionary for $r_i \in [-31, 31]$ is reported in Tab. I together with the number of bits l_i needed for representing the related codeword c_i.
- *ND-Encoding* [17] uses a dictionary specifically designed to achieve high compression ratios in the case of data with Normal Distribution and small variance, i.e. $\sigma^2 \leq 7$ (see Tab. I).

ISBN 978-0-9928626-7-1

	SHuffman		ND-Encoding		MinDiff*	
r_i (or d_i)	c_i	l_i	c_i	l_i	c_i	l_i
0	00	2	00	2	00	2
-1,+1	010x	4	01x	3	01x	3
-3,-2,+2,+3	011xx	5	10xx	4	10xx	4
-5,-4,+4,+5	1000xx	6	110xx	5	11...	2+k
-7,-6,+6,+7	1001xx	6	1110xx	6	11...	2+k
-15,...,-8,+8,...+15	101xxxx	7	1111...	4+w	11...	2+k
-31,...,-16,+16,...+31	110xxxxx	8	1111...	4+w	11...	2+k

TABLE I
DICTIONARY OF SOME LOSSLESS COMPRESSION ALGORITHMS

	RAKE	
r_i	c_i	l_i
-1	1	1
+1	01	2
-2	001	3
+2	0001	4
...	...	...
-R	0...01	$2R-1$
+R	0....01	$2R$
0	all zeros	$2R$

TABLE II
DICTIONARY OF THE RAKE ALGORITHM

A second class of algorithms is based on a predictive coding approach. Basically, algorithms in this class are based on the fact that in most cases it is sufficient to encode only those residues, resulting from the difference between the predicted value and the actual value, which falls inside a relatively small range $[-R, R]$ and to transmit the values outside this range (i.e. outliers) as the original raw data. This approach is commonly known as Two-Modal (TM) transmission [18].

The main problem of this approach is the "huge" complexity of predictive algorithms in comparison of the limited storage and computational resources available in microcontroller-based IoT devices. Despite it is possible to perform a prediction within a node that is energy capable (i.e. the sink in WSN), in the case of multi-hop networks, the energy consumptions (and delay) due to the necessity of forwarding update messages with prediction parameters is a serious drawback.

Recently, a simple lossless compression technique that combines the above approaches has been proposed in [9]. The algorithm, named MinDiff*, exploits the fact that a simple prediction can be done locally, directly on sensor nodes, using the range of the actual set of data.

More precisely, the MinDiff* algorithm represents a set of N residues, $X = \{r_1, ..., r_N\}$, originally encoded with w bits each, with the set $C = \{\mu, d_1, ..., d_N\}$ where $\mu = \min\{r_i\}$ and $d_i = r_i - \mu$ with $i \in [1, ..., N]$.

The differences d_i are firstly encoded with the minimum number of bits k needed for their binary representation, i.e. $k = \lceil \log_2(\max\{d_i\}+1) \rceil$. In a second step, the number of bits needed to represent C is further reduced using the dictionary shown in Tab. I.

In this paper we present a new lossless compression algorithm named RAKE showing that it is able to outperform all the above algorithms.

III. RAKE: BASIC IDEA

Many compression algorithms operate in two phases: in the first phase, pre-processing techniques [19], [20] or transformations [21] are used to obtain sequences that are sparse in some domains; in the second phase, sparsity is exploited in order to achieve compression.

The proposed RAKE algorithm follows the same basic idea by taking into account the limited computational resources of IoT devices on both phases.

In particular, we show in the next Section that the RAKE algorithm is a very efficient compression scheme for binary sparse sequences. Therefore the RAKE algorithm can be used in combination of every transformation or pre-processing technique able to obtain a sparse binary sequence from the original data set.

A possible solution is given by the dictionary shown in Tab. II that is based on a variable length code where every codeword has at most one set bit.

The reader might observe that the above dictionary can lead to a very long binary sequence that could be not compatible with the low memory resources available in IoT devices, however we will show in the next Section how to avoid this problem by using codeword lengths (i.e. l_i) instead of actual codewords c_i.

Formally, a n-bits sequence is said to be k-sparse if there are only $k << n$ non-zero values. Several algorithms already exist able to effectively compress a binary sparse sequence. For instance, the Positional Encoding (PE) algorithm [1] (which encodes a k-sparse sequence of n bits with $k \cdot \lceil \log_2(n) \rceil$ bits by simply indexing positions of the set bits) and the Run-Length Encoding (RLE) algorithm [2].

However, the RAKE algorithm uses a new (and more efficient) encoding scheme. In particular, the proposed algorithm can be explained by imagining a rake able to catch the set bits one bit at the time (the name RAKE derives exactly from the homonymous tool used in agriculture).

In particular, let us consider a T-teeth rake able to fork T bits at a time and let us suppose that a codeword is generated on the basis that a set bit is catched or not. More precisely,

- A single zero bit codeword is used to state that no set bit is found under the rake (i.e. all forked bits are zeros). Otherwise, if at least one set bit has been forked, a codeword of $L = 1 + \lceil \log_2 T \rceil$ bits is generated. In particular, the first bit of the codeword is set to 1 to state that one set bit has been catched and the other $\lceil \log_2 T \rceil$ bits are used to encode its position (here identified as p_{first} and counted starting from 0 to $T-1$).
- After that a codeword has been generated the rake moves forward to catch other possible set bits. More precisely, the position of the rake is shifted by $p_{first}+1$ positions when a set bit has been found or by T positions when all bits forked were zeros.

The above operations are repeated until the rake reaches the end of the sequence to be compressed. Finally, the compressed sequence is obtained by concatenating all the codewords.

ISBN 978-0-9928626-7-1 © EURASIP 2017

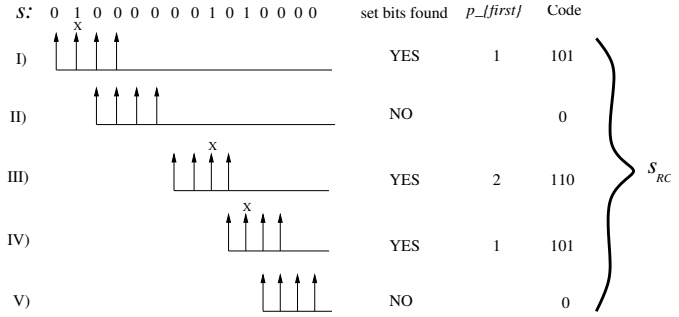


Fig. 1. Example of RAKE compression algorithm.

In Figure 1 is reported a simple example showing how a sparse sequence of $n = 15$ bits, i.e. $S = [010000001010000]$ is compressed by the RAKE algorithm (with $T = 4$) to produce a compressed sequence of 11 bits, i.e. $S_{RC} = [10101101010]$,

Let us illustrate in detail how the algorithm works.

In the first iteration, the initial $T = 4$ bits of the sequence S (i.e. 0100) are considered. The second of these bits is set (so that $p_{first} = 1$) therefore the sequence 0100 is encoded as 101, that is one bit used as prefix (i.e. 1) followed by the binary representation of p_{first} = 1 (i.e. 01, considering that $\lceil \log_2(T) \rceil = 2$ bits must be used for its binary representation). At this point the initial position of the rake is shifted forward by $p_{first} + 1 = 2$ positions.

In the second iteration, four zero bits are forked; thus, on the basis of the algorithm, a single 0 bit is used to encode them and the rake is shifted forward by $T = 4$ positions.

In the third iteration the subsequence 0010 is found under the rake. The third of these bits is 1 (i.e. $p_{first} = 2$) therefore the subsequence is encoded as 110 and the rake is shifted forward by $p_{first} + 1 = 3$ positions.

Similarly, in the forth iteration the subsequence 0100 is encoded as 101 and the rake moves forward by 2 positions.

Finally, in the last iteration the subsequence 0000 is encoded as a single bit 0. At the end of the execution, the compressed sequence is obtained by concatenating all the codewords.

The RAKE algorithm can be effectively combined with the precoding phase (i.e. the dictionary in Tab. II) so that only the compressed and the incoming sequences have to be stored. This is shown in Fig. 2 where a Matlab code able to implement both phases is reported.

As regards decompression, the original binary sequence can be recovered from a compressed sequence as follows:

- Every time a bit 0 is read it is decoded as a sequence of T zeros;
- Every time a bit 1 is read, the next $L - 1$ bits are considered. In particular a number x is obtained as the decimal representation of the binary number encoded in these $L - 1$ bits. Then, a sequence of x bits set to 0 is decoded followed by a bit set to 1.

A. Length of the compressed sequence

It is worth observing that the length of the compressed sequence, henceforward indicated as b_{RAKE}, is maximum

```
function [Sout] = RAKE(Xin)
  r = [0,-1,diff(Xin)]; l_bits = abs(r)*2 - (r < 0);
  z_bits = max(l_bits); K = sum(l_bits ~= 0);
  N = sum(l_bits) + sum(l_bits == 0)*z_bits;
  L = ceil(log2((N/K - 1)*log(2)))+1; T = 2^(L-1);
  Sout = [de2bi(Xin(1), 16), de2bi(L,4)]; % Header
  cnt_z = 0;
  for i=1:length(r)
      if (r(i) == 0) cnt_z += z_bits; one_found = 0;
      else cnt_z += l_bits(i) - 1; one_found = 1;
      end
      while (cnt_z >= T)
        Sout = [Sout, 0]; cnt_z = cnt_z - T;
      end
      if (one_found)
        Sout = [Sout,1,de2bi(cnt_z, L-1)]; cnt_z = 0;
      end
  end
  if(cnt_z > 0) Sout = [Sout, 0]; end;
end;
```

Fig. 2. Matlab-like code for joint econding and RAKE compression

when the overall number of rake operations needed to catch all the bits 1 is maximum. For a fixed value of k this worst case occurs when the uncompressed sequence S starts with a single burst of k consecutive ones, i.e. $S = [1, 1, 1, ..., 1, 0, 0, 0, ...0]$. In fact, in this case, the RAKE algorithm encodes the first k bits with L bits each, whilst the remaining $n - k$ consecutive zeros are catched with the maximum number of rake operations, i.e. $\lceil \frac{n-k}{T} \rceil$, so that $b_{RAKE} = k \cdot L + \lceil \frac{n-k}{T} \rceil$.

Obvioulsy for all the other sequences will be

$$b_{RAKE} \leq k \cdot L + \lceil \frac{n-k}{T} \rceil \tag{1}$$

If we restrict the possibile values of T only to the powers of two so that $T = 2^{L-1}$, the previous relation can be rewritten as $b_{RAKE} < f(T)$ where $f(T) = k \cdot (1 + \log_2(T)) + \frac{n-k}{T} + 1$.

In order to choose a convenient value of T, we consider the one which minimizes the function $f(T)$. Accordingly, by solving the equation $\frac{df(T)}{dT} = 0$ with respect to T, we obtain

$$T^* = (\frac{n}{k} - 1) \cdot \ln(2) \tag{2}$$

So, throughout the rest of the paper we will consider $L = \lceil \log_2(T^*) \rceil + 1$ and $T = 2^{L-1}$.

B. RAKE complexity and compression overhead

As regards the complexity of the RAKE algorithm, it is possible to observe that only counting operations are required for its implementation and that the number of iterations needed for its execution is in the order of $\mathcal{O}(\frac{n}{T}) = \mathcal{O}(k)$. Moreover, as regards the storage requirements, only the compressed and the incoming sequences have to be stored. Accordingly, requirements in terms of storage are restrained.

In order to perform a successful reconstruction of the original sequence S, the value of T must be known at the receiver; thus this value have to be encoded and sent together with the compressed sequence, leading to an overhead of h bits. Considering that T is constrained to be a power of two, i.e. $T = 2^{L-1}$, it is possible to obtain the value of T from L. In this case we have $h = \lceil \log_2(L) \rceil = \mathcal{O}(\log_2(\log_2(\frac{n}{k})))$ bits that is a really small overhead.

ISBN 978-0-9928626-7-1 © EURASIP 2017

IV. Comparison results

In this section we compare the performance achieved by the proposed RAKE compression algorithm with those of other lossless compression algorithms.

To this purpose, the following metrics are considered:

- *Compression Ratio (CR)* defined as the ratio between the number of bits before and after compression i.e.,

$$CR = \frac{\text{Number of bits BEFORE compression}}{\text{Number of bits AFTER compression}} \quad (3)$$

  In particular we indicate with CR_{opt} the maximum lossless compression ratio that can be achieved with an ideal lossless algorithm able to reach the Shannon's entropy.
- *Compression Efficiency (CE%)* defined as

$$CE\% = 100 \cdot (1 - \frac{1}{CR}) \quad (4)$$

A. RAKE for sparse sequences

In this subsection we compare compression ratios achieved with the RAKE algorithm and other lossless compression algorithms in the case of sparse sequences.

In particular, in Tab.III we report the mean compression ratios of several lossless algorithms obtained considering 100 random files of 10KB each (i.e. $n = 80000$ bits) with different sparsity ratios $p = k/n$ (note that files of different lengths have been used for tests by obtaining similar results not reported here for sake of space).

From Tab.III it is possible to observe that the compression ratio achieved by RAKE is better than what can be obtained with other compression algorithms.

It is worth noting that the value of CR_{RAKE} approaches the maximum compression ratio that could be obtained with an ideal entropy encoder, i.e. $CR_{opt} = \frac{1}{H}$ where $H = p \cdot \log_2 \frac{1}{p} + (1-p) \cdot \log_2 \frac{1}{1-p}$ is the Shannon's entropy for binary memoryless sources.

In particular, for all p up to 0.2, CR_{RAKE} differs from CR_{opt} by only a few percent ($< 4\%$). Instead, for all the other algorithms the difference is at least $[15-25]\%$. Accordingly, we can state that in case of sparse sequences the RAKE algorithm is the one which achieves the best performance.

Finally, it is worth noting that there are not appreciable differences among the compression ratios achieved by RAKE and more complex algorithms when $p \geq 0.25$ (note that CRs for $p > 0.25$ are not shown for sake of space).

TABLE III
Compression ratio of different compression algorithms for different values of the sparsity ratio $p = k/n$.

	p							
Algorithm	0.002	0.005	0.01	0.05	0.1	0.15	0.2	0.25
CR_{opt}	48.0	22.0	12.4	3.5	2.1	1.6	1.4	1.2
CR_{RAKE}	47.4	21.5	12.0	3.5	2.1	1.6	1.4	1.2
CR_{rar}	22.1	12.1	7.4	2.6	1.7	1.4	1.2	1.2
CR_{gzip}	21.4	11.8	7.4	2.6	1.8	1.4	1.3	1.2
CR_{bzip2}	26.0	14.2	8.7	2.6	1.7	1.3	1.2	1.1
CR_{PE}	29.5	11.9	5.9	1.2	0.6	0.4	0.3	0.2
CR_{RLE}	35.7	15.7	8.4	2.1	1.2	0.9	0.8	0.6

B. RAKE for real-world data

In this subsection we consider real-world data obtained by the Intel Berkeley Research laboratory [22], regarding indoor weather conditions, and the outdoor micro-meteorology data of the Willow Creek Tower [23].

The Berkeley database have timestamped topology information, along with humidity, temperature, light and voltage values collected from 54 sensors once every 31 seconds. As in [17], three different sensor nodes (node 3, 8 and 19) and two physical variables (temperature and relative humidity) have been considered for evaluation. In the case of the Willow Creek Tower database, outdoor temperature and humidity data have been collected every 30 minutes, so large variations exist among consecutive samples. More detailed statistical information about the above data sets are reported in [9].

As done in [17] and [9], compression efficiencies have been evaluated considering, for each physical variable, a set of 10000 data divided into 100 blocks of 100 words each and represented with $w = 16$ bits. The achieved compression efficiencies are reported in Fig. 3 and in Fig. 4. Note that bold horizontal lines represent the ideal maximum efficiencies imposed by the Shannon's entropy, i.e. $CE_{opt} = 100 \cdot (1 - H/w)$, where the entropy $H = \sum_i p_i \log_2(\frac{1}{p_i})$ has been calculated accordingly to the frequencies of the residues r_i.

As it is possible to observe in all cases RAKE has better mean and maximum compression efficiencies.

The reader might observe that in the case of indoor data the improvement of RAKE with respect to the MinDiff* algorithm is modest (i.e. $\sim 2\%$). However it should be considered that performance of both algorithms are quite near the maximum efficiency CE_{opt}, so further improvements are difficult to be achieved due to the lossless constraint.

Instead, in the case of outdoor data, the compression efficiency obtained with the RAKE algorithm is considerably greater (i.e. $20 - 25\%$) than all the other compression algorithms (even in comparison to more complex algorithms such as gzip and rar, as shown in Fig. 4(b)).

V. Conclusion

In this paper we have presented a simple and effective lossless compression algorithm, named RAKE. Using only a simple dictionary and counting operations, RAKE is able to outperform existing solutions even when different data sets and different physical parameters are considered. Moreover, considering its inherent low complexity and memory requirement, it is well suited for IoT devices. As future works we will apply RAKE to other types of IoT-related signals (i.e. images and biomedical signals) and we will derive further theoretical results about its complexity and performance.

References

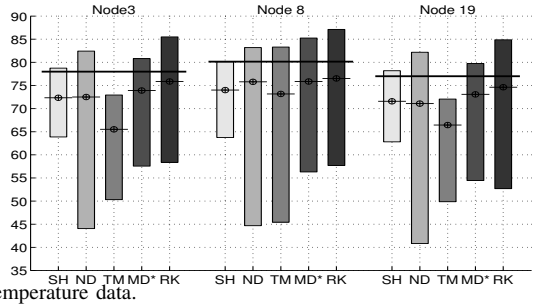
[1] D. Salomon and G. Motta, *Handbook of Data Compression (5.ed.)*. Springer, 2010.

[2] K. Sayood, *Introduction to data compression (4.ed.)*. The Morgan Kaufmann series in multimedia information and systems, 2012.

[3] G. Campobello, S. Serrano, L. Galluccio, S. Palazzo, "Applying the chinese remainder theorem to data aggregation in wireless sensor networks," *IEEE Communications Letters*, vol. 17, pp. 1000–1003, 2013.

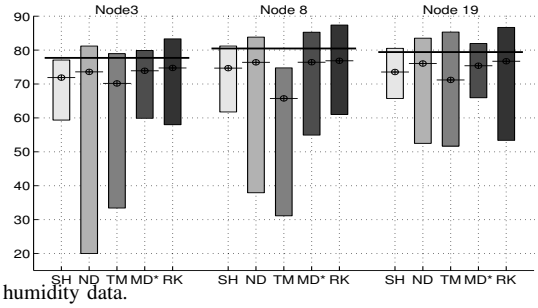
ISBN 978-0-9928626-7-1 © EURASIP 2017

Node #	SH max/**mean**/min [%]	ND max/**mean**/min [%]	TM max/**mean**/min [%]	MinDiff* max/**mean**/min [%]	RAKE max/**mean**/min [%]	Ideal **mean** [%]
3	79/**72**/64	82/**73**/44	73/**66**/50	81/**74**/58	86/**76**/58	**78**
8	80/**74**/64	83/**76**/45	83/**73**/45	85/**76**/56	87/**77**/58	**80**
19	78/**72**/63	82/**71**/41	72/**66**/50	80/**73**/54	85/**75**/53	**77**



(a) Compression efficiencies for indoor temperature data.

Node #	SH max/**mean**/min [%]	ND max/**mean**/min [%]	TM max/**mean**/min [%]	MinDiff* max/**mean**/min [%]	RAKE max/**mean**/min [%]	Ideal **mean** [%]
3	77/**72**/59	81/**74**/20	79/**70**/33	80/**74**/60	83/**75**/60	**78**
8	81/**75**/62	84/**76**/38	75/**66**/31	85/**76**/55	87/**77**/62	**80**
19	81/**74**/66	84/**76**/53	85/**71**/52	82/**75**/66	87/**77**/55	**79**



(b) Compression efficiencies for indoor humidity data.

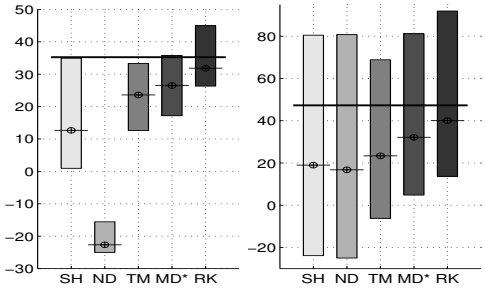
Fig. 3. Compression efficiencies of WSN-specific compression algorithms in the case of real-world indoor data (Intel Berkeley labs' data set).

Data set	SH max/**mean**/min [%]	ND max/**mean**/min [%]	TM max/**mean**/min [%]	MinDiff* max/**mean**/min [%]	RAKE max/**mean**/min [%]	Ideal **mean** [%]
Temp.	35/**13**/1	-16/**-23**/-25	33/**24**/13	36/**27**/17	45/**32**/27	**35**
Hum.	81/**19**/-24	81/**17**/-25	69/**23**/-6	81/**32**/5	92/**40**/14	**47**

(a) Compression efficiencies of WSN-specific compression algorithms.

Data set	$CE_{gzip}[\%]$	$CE_{bzip2}[\%]$	$CE_{rar}[\%]$	$CE_{RAKE}[\%]$
Temp.	3.8	4.3	28.0	32.1
Hum.	36.7	35.4	36.4	40.2

(b) Mean compression efficiencies of general purpose compression algorithms vs RAKE.



(c) Temperature (d) Humidity

Fig. 4. Compression efficiencies in the case of real-world outdoor data (Willow Creek Tower's data set).

[4] G. Campobello, A. Segreto, and S. Serrano, "Data gathering techniques for wireless sensor networks," *Int. J. Distributed Sensor Networks*, vol. 2016, pp. 1–17, Mar. 2016.

[5] M. Vecchio, R. Giaffreda, and F. Marcelloni, "Adaptive lossless entropy compressors for tiny iot devices," *IEEE Transactions on Wireless Communications*, vol. 13, no. 2, pp. 1088–1100, February 2014.

[6] E. Chua and W. C. Fang, "Mixed bio-signal lossless data compressor for portable brain-heart monitoring systems," *IEEE Transactions on Consumer Electronics*, vol. 57, no. 1, pp. 267–273, February 2011.

[7] "LZWC: Lempel-Ziv-Welch Codec." [Online]. Available: http://sourceforge.net/projects/lzw/ (rev 1.4)

[8] K. C. Barr and K. Asanovic', "Energy-aware lossless data compression," *ACM Trans. Comput. Syst.*, vol. 24, no. 3, pp. 250–291, 2006.

[9] G. Campobello, O. Giordano, A. Segreto, and S. Serrano, "Comparison of local lossless compression algorithms for wireless sensor networks," *J. Network and Computer Appl.*, vol. 47, no. C, pp. 23–31, Jan. 2015.

[10] T. M. Cover and J. A. Thomas, *Elements of Information Theory*. Wiley-Interscience, 1991.

[11] "Gzip Home Page." [Online]. Available: http://www.gzip.org (ver 1.6)

[12] "RAR Home Page (www.win-rar.com)." [Online]. Available: http://www.rarlab.com (ver 5.3)

[13] "Bzip2 Home Page." [Online]. Available: http://www.bzip.org (ver 1.0.6)

[14] T. Srisooksai et al., "Practical data compression in wireless sensor networks: A survey," *Journal of Network and Computer Applications*, vol. 35, pp. 37–59, 2012.

[15] C. M. Sadler and M. Martonosi, "Data compression algorithms for energy-constrained devices in delay tolerant networks," *Proc. SenSys '06: 4th Int. Conf. on Embedded networked sensor systems*, pp. 265–278, 2006.

[16] F. Marcelloni and M. Vecchio, "A simple algorithm for data compression in wireless sensor networks," *IEEE Communications Letters*, vol. 12, no. 6, pp. 411–413, 2008.

[17] R. Xuejun and F. Dingyi, "A normal distribution encoding algorithm for slowly-varying data compression in wireless sensor networks," *6th Int. Conf. on Wireless Communications, Networking and Mobile Computing (WiCom2)*, pp. 1–4, 2010.

[18] Y. Liang and W. Peng, "Minimizing energy consumptions in wireless sensor networks via two-modal transmission," *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 1, pp. 13–18, 2010.

[19] J. Pinho, "An online preprocessing technique for improving the lossless compression of images with sparse histograms," *IEEE Signal Processing Letters*, vol. 9, no. 1, pp. 5–7, 2002.

[20] M. Iwahashi, H. Kobayashi and H. Kiya, "Lossy compression of sparse histogram image," in *ICASSP'12*, 2012, pp. 1361–1364.

[21] R. J. Cintra and F. M. Bayer, "A DCT approximation for image compress," *IEEE Signal Processing Letters*, vol. 18, pp. 579–582, 2011.

[22] "Intel Berkeley lab Sensor's data." [Online]. Available: http://db.csail.mit.edu/labdata/labdata.html

[23] K. J. Davis et al., "ChEAS: Chequamegon Ecosystem Atmosphere Study." [Online]. Available: http://cheas.psu.edu/data/flux/wcreek/wcreek2000_met.txt