An Evaluation of Model-Based Approaches to Sensor Data Compression

Nguyen Quoc Viet Hung, Hoyoung Jeung, Karl Aberer

Abstract—As the volumes of sensor data being accumulated are likely to soar, data compression has become essential in a wide range of sensor-data applications. This has led to a plethora of data compression techniques for sensor data, in particular model-based approaches have been spotlighted due to their significant compression performance. These methods, however, have never been compared and analyzed under the same setting, rendering a 'right' choice of compression technique for a particular application very difficult. Addressing this problem, this paper presents a benchmark that offers a comprehensive empirical study on the performance comparison of the model-based compression techniques. Specifically, we re-implemented several state-of-the-art methods in a comparable manner, and measured various performance factors with our benchmark, including compression ratio, computation time, model maintenance cost, approximation quality, and robustness to noisy data. We then provide in-depth analysis of the benchmark results, obtained by using 11 different real datasets consisting of 346 heterogeneous sensor data signals. We believe that the findings from the benchmark will be able to serve as a practical guideline for applications that need to compress sensor data.

Index Terms—lossy compression, sensor data, benchmark

1 Introduction

Today, sensors are ubiquitous. They yield data from almost every field of human activity, often generating high-rate readings continuously. As a result, data compression has become essential in a wide spectrum of applications that store and manage continuously streaming sensor data.

The benefits of archiving sensor data are in fact not exclusively limited to reducing storage at local databases. For example, battery lifetime of sensors, which is a critical issue in wireless sensor networks, can be substantially extended by reducing the amount of transmitted data [10, 13, 26], while also maximizing the utilization of limited communication bandwidth [8, 39, 41]. Moreover, compression enables fast data processing by evaluating queries directly over compressed data [12, 24, 30, 33, 42], incurring much less I/O, since compressed data are generally stored using lower numbers of disk pages [40].

To fully harvest these benefits, a rich body of research work has proposed data compression techniques for sensor data [3, 9, 10, 13, 18, 24]. In particular, model-based compression approaches—which represent sensor data using well-established approximation models whose parameters are then stored as compressed data—have gained significant popularity. These model-based techniques exploit the correlation structure of sensor data for compression, meaning that a stronger correlation leads to a higher compression. For example, if temperature remains nearly constant for a while, the temperature readings can yield a very low volume of compressed data. As a result, model-based compression techniques generally outperform standard compression techniques [2, 34], including BWT [4], LZ77 [44], LZ78 [45], and PPM [22], by exploiting domain-specific knowledge on which errors can be tolerated [11].

Moreover, the models used to approximate raw sensor data offer various benefits to data processing. For instance, they are often used not only for compression, but also for inferring precise values of uncertain raw sensor reading [38], as well as for outlier detection [43] and sensor data cleaning [14, 15, 21], which are important issues in sensor data management. Furthermore, the model approximations can be directly used for indexing [5, 28], data analysis [27, 36], similarity search [24], and so on. These approaches are broadly classified into four categories according to the model type:

- *Constant models* approximate sensor signals using simple constant functions, serving as the basis in PAA[1] [23], PCA [28], APCA [24], Cache filter [31], and PCH [3].
- *Linear models* find the best fitting linear functions to represent a sensor signal, used in PLA [7], PWLH [3], SWAB [25], LTC [35], Slide/Swing Filters (SF) [13].
- *Nonlinear models*, such as Polynomial or Chebyshev approximations (CHEB) [1, 5, 16], capture raw sensor-data signals using complex functions.
- *Correlation models* collectively approximate multiple sensor signals while reducing redundant information, as in SBR [10], RIDA [9], and GAMPS [18].

- *Q.V.H Nguyen and K. Aberer are with École Polytechnique Fédérale de Lausanne (EPFL), Switzerland.*
  E-mail: quocviethung.nguyen and karl.aberer@epfl.ch
- *H. Jeung is with SAP Research Brisbane, Australia.*
  E-mail: hoyoung.jeung@sap.com

1. Table 1 offers full names for all abbreviations.

Each of these techniques has distinct performance characteristics. One, for example, may achieve very high compression over certain types of sensor data, while another incurs low CPU usage for model maintenance. In contrast, some techniques are designed to facilitate efficient query processing without taking into account CPU usage. Moreover, these methods have never been compared all together, and each work often reported its superior performance generally using a limited variety of data sets or evaluation methodologies. As a result, understanding the performance implications of these techniques, for a given type of application, is a challenging problem.

The primary goal of this study is to compare all these methods within a common framework. To this end, we present a benchmark that offers an overview of comprehensive performance comparison among the compression techniques, describes in-depth analysis on the performance behavior of each method, and provides guidance on the selection of appropriate compression schemes. Specifically, the salient features of the benchmark are highlighted as follows:

- We fairly re-implemented one or two most representative state-of-the-art method(s) in each category of the model-based approaches, including PCA, APCA, PWLH, SF, CHEB, and GAMPS. We then compared them with regard to compression ratio, computation time, packet ratio, power ratio, approximation error (RMSE), and robustness to noise. As a result, the benchmark revealed the performance characteristics of a particular class of model-based approach, as well as of individual methods.
- We established a generic, extensible benchmarking framework to assist in the evaluation of different model-based compression techniques, so that subsequent studies are able to easily compare their proposals with the state-of-the-art techniques. It is straightforward to plug in a new compression method as well as a new dataset into our framework. The source code of the framework is publicly available on our website[2] for further research, comparison or repeatability.
- We integrated the TOSSIM [29] simulator into our benchmarking framework for simulating data transmission over two well-known network topologies: broadcast and multi-hop. In addition, our benchmark also allows users to customize the network topology by modifying included scripts. On top of the communication simulator, we furthermore support estimating energy consumption using PowerTOSSIM-Z [32].
- We used various real sensor datasets in the benchmark—11 large-scale datasets obtained from 346 heterogeneous sensors used in an environmental monitoring project, the Swiss Experiment[3]. In particular, we carefully selected sensor data signals that exhibit very different data distributions and dynamicity (Figure 5). This diversity of data helped clearly to demonstrate the characteristics of each method under various data distributions found in practice.
- We offer extensive as well as intensive performance analyses. Whenever we had an unclear outcome, we ran the same test using different settings and datasets, and finally drew meaningful and reliable conclusions. We believe that the analyses can serve as a practical guideline for how to select a well-suited compression method on particular application scenarios.

2. http://lsirwww.epfl.ch/benchmark
3. http://www.swiss-experiment.ch/

The remainder of the paper is organized as follows. Section 2 reviews state-of-the-art model-based compression techniques. We then describe the methodology used in the benchmark in Section 3. Section 4 offers in-depth discussions on the benchmark results. Section 5 finally summarizes and concludes this study, where we provide important suggestions for the applications that consider employing a model-based compression method.

PAA	Piecewise Aggregate Approximation [23]
PCA	Piecewise Constant Approximation [28]
APCA	Adaptive Piecewise Constant Approximation [24]
PCH	Piecewise Constant Histogram [3]
PLA	Piecewise Linear Approximation [7]
PWLH	PieceWise Linear Histogram [3]
LTC	Lightweight Temporal Compression [35]
SWAB	Sliding Window and Bottom-up [25]
SF	Slide Filters [13]
CHEB	Chebyshev Approximation (CHEB) [1, 5, 16]
SBR	Self-Based Regression [10]
RIDA	Robust Information-Driven Data Compression Architecture [9]
GAMPS	Grouping and AMPlitude Scaling [18]

TABLE 1: List of abbreviations.

2 Model-Based Compression Techniques

As sensor data are the results of physical processes, it is generally possible to observe the following properties: *continuity* of the sampling processes and *correlations* of different sampling processes. In principle, model-based compression methods exploit either or both of the properties. Specifically, they first compute the dependency from one variable (e.g., time) to another (e.g., sensor reading), and then reduce the redundant or repeated information. Various models are used to discover such dependency in sensor signals, resulting in different compression performance.

Model-based approaches can outperform standard compression techniques, as they exploit the dependency information of data on which errors are being tolerated within a maximum error threshold ϵ. In other words, the compression techniques are built upon approximations of one- or multi-dimensional sensor data signals under the L_∞ norm.

In the following, we offer the details of model-based compression techniques commonly used for sensor data processing. We summarize them according to the model classification described in the previous section.

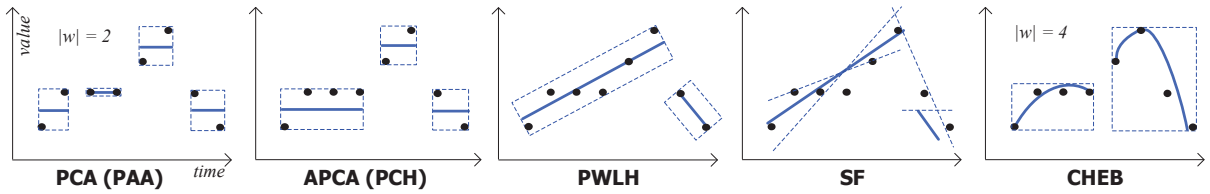


Fig. 1: A comparison of sensor data representations using different compression algorithms.

2.1 Constant-Model Compression

Constant-model compression methods approximate sensor signals using simple constant functions, which require small numbers of model parameters. They also facilitate intuitive indexing. The literature suggests various algorithms that fall into this category, including Piecewise Aggregate Approximation (PAA) [23], Piecewise Constant Approximation (PCA) [28], Adaptive Piecewise Constant Approximation (APCA) [24], Piecewise Constant Histogram (PCH) [3] [17], and Cache filter [31]. This section presents the details for two of these models, PCA and APCA, which cover the characteristics of the others.

2.1.1 Piecewise Constant Approximation (PCA)

PCA is a straightforward method that divides an entire sensor data signal into piecewise fixed-length segments, and then represents the data values in each segment using model parameter values in turn. More precisely, given a sensor signal $S = \langle v_1, v_2, \cdots, v_n \rangle$, a window w with size of $|w|$, and a maximum error threshold ϵ, PCA first takes $|w|$ consecutive data points of S (i.e., $w = \langle v_1, v_2, \cdots, v_{|w|} \rangle$), and then calculates the maximum value $v_{max} \in w$ and the minimum value $v_{min} \in w$. If $v_{max} - v_{min} < 2\epsilon$, all the data points $v_i \in w$ are represented by a constant $\hat{v} = (v_{max} - v_{min})/2$. Otherwise, the original data values are not approximated, and preserved in the compression output. This process is sequentially repeated for the remaining segments in the signal. Piecewise Aggregate Approximation (PAA) works the same; the terms PCA and PAA are used interchangeably. **PCA (PAA)** in Figure 1 illustrates this process, where constant line segments represent the approximated signal corresponding to the raw data points shown as dark dots.

Model parameters: Applying the above approximation process, the original signal is represented by a sequence of model parameters c_i. c_i is set by either the constant value $\hat{v}$ (when $v_{max} - v_{min} < 2\epsilon$) or the original data values (when the maximum error constraint is violated). Therefore, in addition to c_i, PCA needs a boolean value f_i (1 bit) to indicate whether c_i is a constant or a raw data point. As the window size is identical for all segments, the time information of each compressed data point can be inferred by segment indexes, when decompression of the compressed data is required. Equation 1 is the formal definition of PCA's model parameters.

$$PCA(S) = \langle (c_1, f_1), (c_2, f_2), ...(c_k, f_k) \rangle \qquad (1)$$

2.1.2 Adaptive PCA (APCA)

In principle, APCA also operates as PCA, yet the window size $|w|$ varies accordingly. Given a signal S and an error tolerance ϵ, APCA greedily scans S in order, inserting data values into w until $v_{max} - v_{min} > 2\epsilon$ (excluding the last point that causes the invalidation). The data points in w are then represented by the median value $\hat{v} = (v_{max} - v_{min})/2$. APCA repeats the same process from the next value using a new empty window, i.e., $|w| = 0$, until all the values in S are approximated. The same process of data approximation applies to Piecewise Constant Histogram (PCH) [3]. **APCA (PCH)** in Figure 1 depicts an example of this approximation mechanism.

Model parameters: The approximation process of APCA results in a sequence of disjoint data segments, each of which contains a different number of data points. Each segment is then represented by a pair $(\hat{v}_i, t_i)$, where $\hat{v}_i$ is the median value of the data points, and t_i is the timestamp of the last data point in each segment. Formally, APCA's model parameters are defined as:

$$APCA(S) = \langle (\hat{v}_1, t_1), (\hat{v}_2, t_2), ...(\hat{v}_k, t_k) \rangle \qquad (2)$$

2.2 Linear-Model Compression

Typically, the compression methods built upon linear models extend the constant-model compression techniques, by using linear functions for approximating a raw sensor data signal. Here, we review two most representative linear-model compression techniques, PWLH [3] and SF [13]:

2.2.1 PieceWise Linear Histogram (PWLH)

PWLH is an extension of the APCA algorithm. In this technique, a sensor data signal is first divided into a sequence of variable-length data segments, as the process in APCA. All data points in each segment are then modeled by a line which minimizes the maximum distance from these points to this line. A key difference between APCA and PWLH is the shape of data segment. In APCA, data segments are horizontal, whilst those in PWLH can rotate while centering to the approximation lines, shown as **PWLH** in Figure 1. Note that the maximum error guarantee also holds for the approximated data in each segment.

Model parameters: As the begin-value $\hat{v}_i^b$ and the end-value $\hat{v}_i^e$ of the bisecting line of each data segment are different, both values must be kept when compressing a sensor data signal using PWLH. Concerning time information, PWLH also stores the last timestamp t_i of each segment, as APCA does. Formally, the model parameters

of PWLH are:

$$PWLH(S) = \langle(\hat{v}_1^b, \hat{v}_1^e, t_1), ..., (\hat{v}_k^b, \hat{v}_k^e, t_k)\rangle \quad (3)$$

2.2.2 Slide Filter (SF)

SF approximates a sensor signal in a similar manner as PWLH; however, it finds a linear approximation that can cover the largest number of data points. This filter is mainly designed for online processing; at any point in time, the filter maintains a set of possible approximation lines, all obeying the maximum error constraint. As each new data point is added to the data segment, the filter prunes the lines in the set that do not tolerate the maximum error, until the last one remains. **SF** in Figure 1 illustrates this process. SF accepts the mixture of connected and disconnected line segments. The major difference between connected and disconnected line segments is the number of data points to be stored in the compressed file. For representing two disconnected line segments, SF needs four points; whilst only three points are required to capture two connected segments. Thus, whenever possible, connected line segments are used to increase the compression ratio.

Model parameters: Due to the mixture of connected and disconnected line segments, SF takes a one-bit boolean operator f_i to indicate whether the end point of an approximation line segment is connected to the beginning point of the next approximation line segment. Equation (4) is the formal definition of SF's model parameters.

$$SF(S) = \langle(\{\hat{v}_1^b\}, \hat{v}_1^e, t_1, f_1), ..., (\{\hat{v}_k^b\}, \hat{v}_k^e, t_k, f_k)\rangle \quad (4)$$

Note that $\{\hat{v}_i^b\} = \emptyset$ when either $\hat{v}_i^b = v_0$ (the first data point in S) or $\hat{v}_i^b = \hat{v}_{i-1}^e$ (connected line segments). This is indicated by f_i.

2.3 Nonlinear-Model Compression

Nonlinear models capture sensor signals using more complex functions, resulting in theoretically more accurate approximation of data. However, compression techniques built on nonlinear models generally require more coefficients to approximate a given sensor data signal, rendering the size of compressed data relatively larger. Both polynomial and Chebyshev Approximations (CHEB)[1, 5, 16] are widely used techniques from this compression category. In particular, CHEB is often preferred for data processing, as it can quickly compute a near-optimal approximation for a given signal.

2.3.1 Chebyshev Approximation (CHEB)

Chebyshev compression (CHEB) operates like PCA in terms of how to divide a given signal—a fixed-length window is used. For a segment (window) $w = \langle v_1, v_2, \cdots, v_{|w|}\rangle$ and a degree d, following Chebyshev transformation [16], we can represent w as a linear combination of Chebyshev polynomials:

$$\hat{v}_t = \sum_{k=0}^{d} \beta_k \cdot T_k(t')$$

where

- $t' = (t - \frac{|w|+1}{2}) \cdot \frac{2}{|w|-1}$ is the normalized value of t within the range $[-1, 1]$.
- $T_k(x) = \cos(k \cdot \cos^{-1}(x))$, $x \in [-1, 1]$ is the Chebyshev polynomial of degree k.
- β_k is the Chebyshev polynomial coefficient at degree k.

CHEB in Figure 1 shows an example of the Chebyshev approximation.

Model parameters: When the above approximation process is completed, we have a set of Chebyshev coefficient sets $\{\beta_i\} = \beta_0, \beta_1, ..., \beta_d$ for each data segment in a raw signal. In the same way as PCA stores its parameters to the compressed file, CHEB also stores $\{\beta_i\}$ only when the maximum error constraint is satisfied. Otherwise, the original data points are preserved in the compressed data. Given a degree d, the model parameters of CHEB stored in compressed data are formally defined as:

$$CHEB(S) = \langle(\{\beta_i\}_1, f_1), ..., (\{\beta_i\}_k, f_k)\rangle \quad (5)$$

where f_i is a d-bit bitmap mask that represents which order of the coefficients in $\{\beta_i\}$ are used in the compressed data. More details regarding this mechanism are described in [16].

2.4 Correlation-Model Compression

Correlation models collectively approximate multiple sensor signals while reducing redundant information in correlated signals. Robust Information-Driven Data Compression Architecture (RIDA) [9], Self-Based Regression (SBR) [10], and Grouping and AMPlitude Scaling (GAMPS) [18] are well-known methods among those established on such correlation models. While GAMPS is based on L_∞, RIDA and SBR compress data streams under L_2. L_2 is not of interest in our benchmark since it does not support query-processing directly on compressed data. Moreover, the modification of RIDA and SBR to support L_∞ requires probabilistic guarantee and expensive computation [18]. For these reasons, we implement GAMPS only.

2.4.1 Grouping and AMPlitude Scaling (GAMPS)

GAMPS is an offline method that compresses multiple sensor data signals simultaneously. It first approximates each signal using APCA, then groups similar ones among the approximated signals. One signal in each group is then chosen as a *base signal*. For each remaining signal,

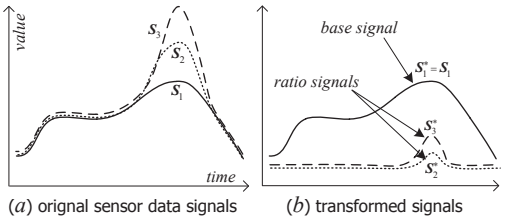


Fig. 2: Signal transformation in GAMPS.

GAMPS computes the *ratio signal* with respect to the base signal. Since signals in the same group tend to share similar trends in data values, collaboratively approximating the transformed (APCA-applied) signals generally requires less segments to store in the compressed file.

For example in Figure 2, S_1 is chosen as base signal, thus $S_1^* = S_1$. Meanwhile, S_2 and S_3 in Figure 2(a) are transformed respectively as S_2^* and S_3^* in Figure 2(b), by considering the ratio to the base signal S_1. As a base signal, S_1^* is compressed by APCA with ϵ_b; while as ratio signals, S_2^* and S_3^* are compressed by APCA with ϵ_r. Regarding the decompression, the original signals are re-constructed from base signal and ratio signals. Therefore, to guarantee the maximum error threshold ϵ, we must have $\epsilon_b + \epsilon_r = \epsilon$.

Intuitively, compression on the transformed signals in Figure 2(b) would be more effective. However, this approach can become ineffective when given signals are uncorrelated since the maximum error threshold ϵ is divided among base signals and ratio signals.

Model parameters: GAMPS needs to store two types of information in compressed data: (1) a mapping table that contains the information of base signals, and (2) a set of model parameters of APCA for all the ratio as well as base signals. The mapping table for n signals consists of a set of n tuples (i, m_i), where $1 \leq i, m_i \leq n$, forming $M = \langle (1, m_1), (2, m_2) ... (n, m_n) \rangle$. If $i = m_i$, signal i indicates the base signal. Otherwise, it is compressed with respect to signal m_i.

In GAMPS, each signal S_i is transformed into S_i^* that is either a ratio signal or base signal. Let $APCA_{\hat{\epsilon}}(S_i^*)$ be the set of model parameters of APCA for signal S_i^* under maximum error threshold $\hat{\epsilon}$. Equation 6 defines the model parameters for GAMPS:

$$GAMPS(S_1, ..., S_n) = M \cup \left[\bigcup_{i=1}^{n} APCA_{\hat{\epsilon}}(S_i^*) \right] \quad (6)$$

where

$$APCA_{\hat{\epsilon}}(S_i^*) = \begin{cases} APCA_{\epsilon_b}(S_i^*) & \text{if } S_i^* \text{ is base signal} \\ APCA_{\epsilon_r}(S_i^*) & \text{if } S_i^* \text{ is ratio signal} \end{cases}$$

To sum up this section, we already implemented six compression techniques—PCA, APCA, PWLH, SF, CHEB and GAMPS—which compress data under L_∞ error tolerance and support query-processing directly on compressed data. In addition, these techniques use online computing model except GAMPS. Table 2 features each implemented technique with following key characteristics:

- **compression model :** the function type to capture the behavior of data stream such as constant, linear, non-linear, etc.
- **computing model :** the ability to perform (online or offline) in response to the new arrival of data points.
- **segment length :** the adaptivity to various lengths (fixed vs. arbitrary) of data segment.
- **#input stream:** the ability to process a single stream or multiple streams.

technique	compression model	computing model	segment length	#input stream
PCA (PAA)	constant	online	fixed *	single
APCA (PCH)	constant	online	variant	single
PWLH	linear	online	variant	single
SF	linear	online	variant	single
CHEB	non-linear	online	fixed *	single
GAMPS	correlation	offline	variant	multiple

* user must predefine segment length

TABLE 2: Characteristics of compression techniques.

One interesting point to note is that all of the above techniques support queries on compressed data with two window operators [6]: sliding window [37] and tumbling window (disjoint window) [19]. This is because when decompression of the compressed data is required, the approximated (decompressed) value and error of one data point are independent of the approximated value and error of other data points since these techniques do not require historical data and the error tolerance is L_∞ norm. Therefore, given a timestamp and compression model parameters, we can estimate the value and the range of original data without decompressing the entire data stream. GAMPS is specific among the chosen methods as in order to support queries on compressed data, it requires additional index structures [18].

3 Benchmark Setup

This section describes the setup used in our benchmark. We first present the details for our benchmarking framework as well as the implementation of each compression algorithm. We then offer an overview of the datasets used in the experiments, followed by descriptions of the measures used to assess the compression performance of each method.

3.1 Framework

A primary goal of this study is to provide a flexible and powerful tool to support the comparison and modification of the model-based compression methods. To this end, we have developed a framework that facilitates the performance study of various (model-based) compression methods. The framework is very flexible and extensible, since a new compression method as well as a new dataset can be easily plugged in. The framework is available for download from our website[4].

Figure 3 illustrates the architecture of the framework. It is built upon a component-based architecture having three layers: *data access layer, computing layer* and *application layer*. The data access layer abstracts the underlying sensor data sources, and loads the data to the upper layer. The computing layer then runs the compression algorithms implemented and plugged into the framework, while reading the input data provided by the data access layer. The computing layer consists of four components: algorithm component, output component, simulator component and evaluation component. The algorithm component takes a configuration file provided by the user, specifying the

4. http://lsirwww.epfl.ch/benchmark

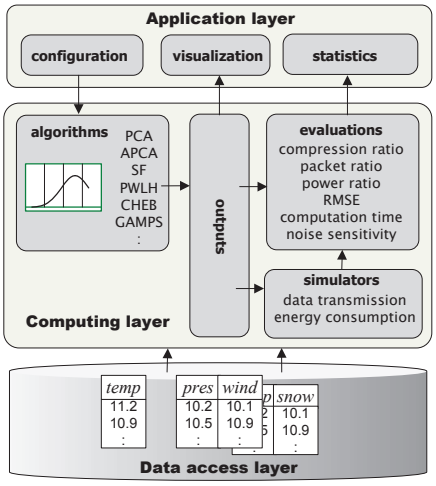


Fig. 3: Benchmarking framework.

parameters for each compression algorithm. The output component provides pre-defined abstractions that feed inputs to the visualization component in the application layer. This component helps understand results from each test of the compression algorithms. The simulator component models the process of transmitting data on sensor networks in order to provide information about transmission cost and energy consumption. The evaluation component accepts inputs from the output component as well as the simulator component, and delivers summarized information to the application layer, facilitating test analysis.

As presented, it is straightforward to plug in a new compression method into the framework. We believe that subsequent studies are able to easily compare their proposals with the state-of-the-art techniques by using our framework.

3.2 Implementation

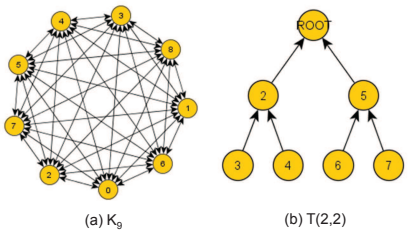
In this section, we discuss the key issues to develop this framework. Firstly, we describe the details of the simulator component. After that, we explain the implementation assumptions and methodologies for comparing compression algorithms in a fair manner.

3.2.1 Transmission Simulator

In order to implement the simulator component, we use TOSSIM [29] to simulate transmitting packets and use PowerTOSSIM-Z [32] (an extension to TOSSIM) to estimate energy consumption of TinyOS applications. The energy model for PowerTOSSIM-Z is micaZ that can be modified by potential users. Note that the TOSSIM simulator limits the packet size up to 28 bytes. Thus if the size exceeds 28 bytes, the packet is split into smaller packets.

The simulator requires a predefined topology of the sensor-network where each node is a sensor mote. So far, we already implemented two well-known network topologies: broadcast and multi-hop data gathering tree.

- **Broadcast:** The network can be visualized as a complete graph K_n with n sensor motes as the vertices. In terms of direction of data transmission, each mote in the broadcast topology is both a data producer (send AM messages to all other motes) and a data consumer (receive AM messages from them). Figure 4(a) illustrates a broadcast network with 9 sensor motes.
- **Multi-hop:** A k-ary tree $T(h, k)$ of height h and internal degree k is used to graphically present the network. Motes being the leaves of the k-ary tree transmit AM messages to their corresponding parents. Upon receiving the messages, those parents immediately transfer them to the next higher level. This process is continued until the message arrives at the root node. Thus, the leaves and the root act solely as the producers and the consumer respectively, while the internal nodes play both. Figure 4(b) depicts a 2-hop network.



(a) K_9 (b) T(2,2)

Fig. 4: Broadcast and Multi-hop sensor networks

Moreover, our benchmark also provides a script for users to define their own topology.

3.2.2 Compression Algorithms

In order to compare the performance of state-of-the-art model-based compression methods, we have made the best effort to fairly re-implement the most representative algorithms in each category of the model-based compressions. We used C++ language for the implementation without using any programming libraries, so that we could fairly control the performance of each method. Here, we offer the implementation details for each compression method used in the benchmark:

- **PCA**: In addition to the error tolerance ϵ, which is a common system parameter for all the model-based compression methods, PCA requires an extra parameter, window size $|w|$ (Section 2.1.1 offers the details). We set window size $|w| = 16$ and kept this setting for all tests related to PCA.
- **APCA**: The implementation of APCA is relatively simple. As the window w works adaptively, APCA does not need to specify an appropriate value for $|w|$. Both data approximation and storing the model parameters (Equation 2) to compressed data are processed on-the-fly while reading input data.
- **PWLH**: It requires computing a linear rectangle (window) for representing a segment of raw data points, illustrated as **PWLH** in Figure 1. We computed the rectangle using the convex hull of the data points, by applying the Graham algorithm [20]. Fortunately, sensor data are already ordered in time, thus the sorting step in the Graham algorithm was eliminated.

Therefore, the time complexity to build and update such a convex hull is $O(n)$, instead of $O(n \cdot \log n)$.

- **SF**: Our implementation for SF did not use a convex hull optimization, since the results shown in [13] indicate that the optimization works out only when the error bound ϵ is greater than 10% of data range. In real applications, ϵ is set by a small value, typically much lower than $\epsilon = 10\%$. In addition, the cost of computing convex hull is expensive; thus it is reasonable to avoid the convex hull optimization process of SF.
- **CHEB**: Similar to PCA, CHEB also requires an extra parameter, window size $|w|$. We set $|w| = 16$ as we did for PCA. In addition, the degree of Chebycheb polynomials is another system parameter in CHEB (see Section 2.3.1). We set $d = 15$ following the recommendation in [16]. These settings were applied for all the tests regarding CHEB in the benchmark.
- **GAMPS**: Following the authors' suggestion in [18], the splitting fraction in terms of a given ϵ was set to $0.4 \times \epsilon$ for base signals as well as ratio signals. In addition, we implemented a static group, in which the ratio signals were computed for the whole signal, instead of dividing that into sub-signals.

Note that searching for parameter settings is a critical issue to evaluate the above methods. In addition to streaming data and maximum error threshold, some of these methods need other tunable parameters (e.g. window size) which might significantly affect the performance of compression. However, finding the best parameters for each method would be difficult; and even if we could find them, it would be unfair to the methods with fewer parameters. Therefore, for fair comparison, all the parameter settings are decided based on the original authors' recommendation and fixed for all runs. In case of no precise recommendation, we use a test-set consisting of 10 streams for each data category. For each parameter setting, we define all possible values and then use brute-force search to find the best value with respect to compression ratio. The compression ratio is used since it is considered as the most important metric.

In addition, by the design of our framework, potential users are still able to change these parameter settings by simply altering the configuration file in the benchmark. This is actually another motivation of our benchmark, which aims to offer a reference for potential users to find not only best-suited method but also appropriate parameters for their specific application.

3.3 Datasets

In order to evaluate the model-based compression methods under realistic settings, we carefully chose sensor datasets while considering heterogeneity of sensors, volatility of data values, diversity of sampling rates, and regions where the data were collected. The datasets were obtained from a wide variety of sensors deployed for an environmental monitoring project[5]. These sensors measured 11 physical variables: moisture, pressure, humidity, voltage, lysimeter, snow-height, temperature, CO_2, radiation, wind-speed and wind-direction. Each dataset contains a different number of sensor data signals from another. Table 3 offers key statistics of each dataset, and Figure 5 demonstrates a subsequence of each dataset. As mentioned above, the sensor signals show very diverse trends and volatilities of data, which can effectively impact the performance characteristics of each model-based compression approach. Note that the mean and min values of snow-height have negative values, which are inaccurate, due to the sensors' mechanical problems for some periods.

5. http://www.swiss-experiment.ch

3.4 Evaluation Measures

We characterize the compression methods compared in the benchmark using five measures: *compression ratio, computation time, transmission cost, approximation error*, and *sensitivity to outliers*. Regarding transmission cost, we consider two important metrics: packet ratio and power ratio. We describe the details for each of the measures in the sequel.

3.4.1 Compression Ratio

Obviously, the most important aspect of a compression method is its compression power. It is straightforward how to measure that—compression ratio is defined as the fraction of storage cost between compressed data and the corresponding original data:

$$compression\_ratio = \frac{size\_of\_compressed\_data}{size\_of\_original\_data} \quad (7)$$

The lower compression ratio, the higher power of compression. We set increasing values for the error tolerance ϵ, while measuring the compression ratios of each method in the experiments.

3.4.2 Transmission Cost

In a wide range of sensor network applications, especially multi-hop sensor networks, data transmission is a costly operation for sensor nodes. In particular, the energy consumption for data propagation on networks is much higher than the energy consumption for compression [9]. Therefore, transmission cost —i.e., the cost of a data transmission from one node to another—is an important metric for evaluating compression methods.

Specifically, this measure concerns a sensor-network scenario where one of the compression techniques is shared by two nodes: data producer and consumer. The producer node keeps approximating an incoming sensor data signal, and then subsequently updates the model parameters of the compression method to the consumer node, only when the maximum error guarantee is violated. This scenario is commonly used in wireless sensor network applications [1, 13, 31].

To reflect transmission cost, we consider two important metrics: (1) packet ratio and (2) power ratio:

Packet Ratio: Packet ratio is measured as the fraction of the number of packets between using original data and

dataset (abbreviation)	number of signals	total number of readings	sampling rate (avg.)	min	max	mean	standard deviation	short-term fluctuation
moisture (mois)	20	442,313	300 sec.	16.18	30.50	18.20	1.09	medium
pressure (pres)	14	161,004	2 sec.	18.20	796.90	268.43	222.96	low
humidity (humi)	34	2,031,732	60 sec.	19.88	93.02	72.89	14.27	meium
voltage (volt)	16	523,877	600 sec.	3.29	3.49	3.41	0.05	medium
lysimeter (lysi)	24	232,746	300 sec.	0	3.81	0.11	0.35	medium
snow-height (snow)	29	349,557	600 sec.	-662.10	370.00	-18.17	155.00	low
temperature (temp)	78	4,846,162	60 sec.	-29.76	49.05	-4.57	9.78	medium
CO_2 (CO_2)	16	3,088,233	16 sec.	0	2000	558.00	219.4	high
radiation (radi)	34	3,955,268	600 sec.	137.5	798.40	261.02	46.86	medium
wind-speed (wspd)	46	2,376,156	60 sec.	0	14.37	2.84	2.31	high
wind-direction (wdir)	35	2,084,943	60 sec.	1.79	357.47	232.65	84.40	high

TABLE 3: Real datasets used in the benchmark.

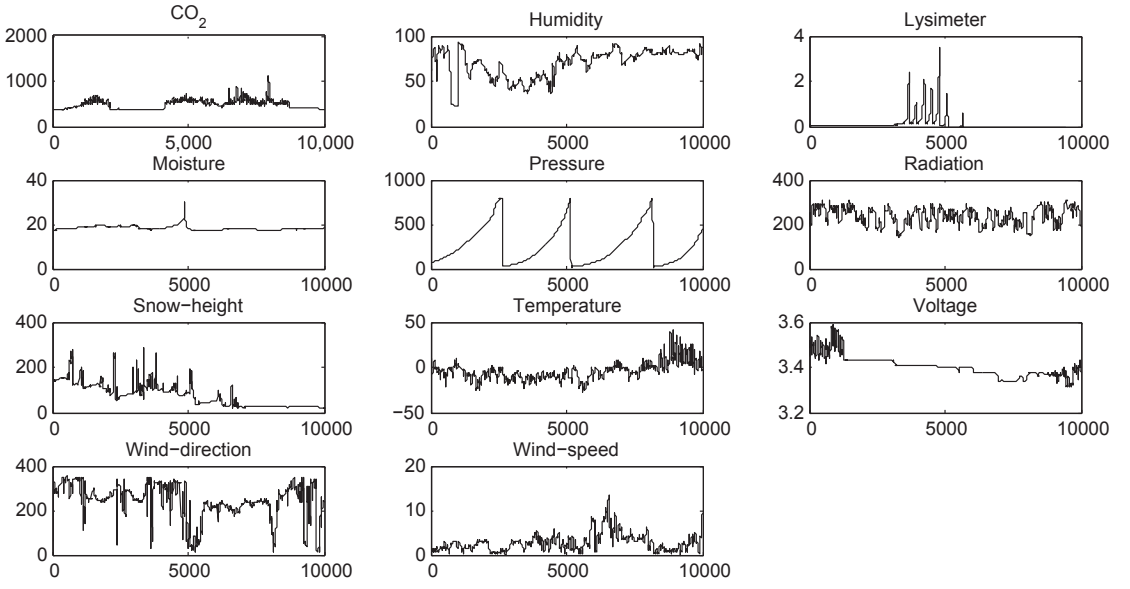


Fig. 5: Visualizations of 10,000 data points in each sensor dataset.

using the corresponding compressed data. Specifically, we measure as follows:

$$packet\_ratio = \frac{\#packet\_of\_original\_data}{\#packet\_of\_compressed\_data} \quad (8)$$

Power Ratio: Power ratio is defined as the fraction of energy consumption between sending compressed data and sending raw data. Formally, it is calculated as:

$$power\_ratio = \frac{power\_of\_compressed\_data}{power\_of\_original\_data} \quad (9)$$

In addition to these normalized metrics, our benchmark also provides absolute values such as number of transmitted packets and amount of power consumption (in joule J). Obviously, compression techniques with fewer transmitted packets incur a low communication overhead, leading to energy-efficient sensor network applications.

3.4.3 Computation Time

Another metric for evaluating compression techniques is running time for compression. Various sensor-network applications often have constraints on computing speed, or limitations in using CPU for energy saving. As a result, the running time for compression becomes an important aspect, when we characterize a sensor data compression method.

In our benchmark, all algorithms are implemented and tested on the same standard. Specifically, we randomly chose 100 sensor signals of length 1,000, 10,000, and 100,000 values. We then measured the average processing time per data point, while varying the value for the error tolerance ϵ.

3.4.4 Approximation Error

Although the benchmark concerns the compression methods under the L_∞ norm, applications often need to know the quality of data compression, i.e., (overall) difference between actual data values and the corresponding decompressed (approximated) data values. In particular, when multiple compression methods show similar performance, approximation error can become a subordinate metric to consider. To reflect this aspect, we represent the approximation errors of data compression using the root mean square error (RMSE), which is widely used to evaluate the quality of approximation. Formally,

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(v_i - \hat{v}_i)^2} \quad (10)$$

where v_i is a raw data value in a given sensor data signal, and $\hat{v}_i$ is the corresponding approximated value in the compressed signal. The lower RMSE, the better approximation (compression).

As each sensor data signal has a particular range where data values are bounded, we normalized the RMSE values (from 0 to a given error tolerance ϵ) at each experiment set, in order to have a common view to analyze the experimental results.

3.4.5 Sensitivity to Outliers

One typical characteristic of sensor data is their uncertain and erroneous nature, originating from various sources, such as discharged batteries, network failures, accumulation of dirt on sensors, and imprecise readings from low-cost sensors. As a result, it is important to know how each compression method performs over noisy data, which is generally characterized by the portion of significant outliers.

In the benchmark, we studied the sensitivity to outliers by recording the compression ratios, while varying the degree of outliers. To this end, we artificially included outliers to the tested datasets, while applying different appearance probabilities of outliers p = 0%, 1%, 2%, 3%, 4%.

4 Experimental Evaluation

We proceed to report on the results of applying the benchmark to the six model-based data compression algorithms. The main goal of the experiments is not only to compare the compression performances, but also to analyze the effects of data characteristics on the performance behavior. All the experiments ran on an Intel Core i7 processor 2.8 GHz system with 4 GB of main memory.

Throughout the benchmark, we computed the evaluation measures with different applications of compression methods and error threshold ϵ. Note that the absolute error varies over a widely different range since each dataset has different domain values. Thus it is impossible to compare all compression methods in a same manner. For the purpose of giving the general picture, we used relative error (in percentage) in the experiments.

To set the relative error ϵ in the experiments, we first computed the difference between the maximum and minimum values in a sensor data signal. We then used a certain proportion to the difference for the value of the error tolerance. For instance, suppose that a sensor signal has 1000 and 0 as the maximum and minimum values among its all readings, respectively. Error tolerance $= 1\%$ then indicates that ϵ equals to $(1000 - 0) \times 0.01 = 10$. In real scenarios, the benchmark also allows users to define absolute error tolerance since we do not know the max/min values in advance if the data are compressed on the fly.

Instead of using the exact max/min values of a sensor data signal for deriving an error tolerance, we first computed the range of a 95% confidence interval (for the mean) of the values in the signal. The values fall far outside the range of standard deviation are considered statistically significant in a wide range of sciences, implying outliers. We then selected the border values of the range as the max/min values to derive the error tolerance. In this way, normal random error or variation in the measurements is distinguished from potential errors.

4.1 Compression Ratio

In the first set of experiments, we compare the compression power of the methods. Moreover, we also discuss the results from three difference perspectives: *overall performance, effect of data*, and *effect of error tolerance*.

4.1.1 Overall Performance

Table 4 presents the ranking of the compression methods with regard to compression ratio. Table 3 shows the full names of the abbreviations used for dataset names in the table. Although there is no absolute winner who outperforms all the others across all the datasets and all different parameter settings, APCA and SF show better performance in many tests. Specifically, APCA shows the highest compression power on 10 datasets, and SF reports the second best performance on 8 datasets. In principle, SF uses linear regression models that are more expressive to capture dynamic data trends than constant models; however, the compression power of APCA is higher in most test cases. The key reason is the representation of data approximation. APCA requires only two values in the compressed data to represent a constant segment (Equation 2); while SF requires three or four values to capture a linear segment in the compressed data (Equation 4).

Compression methods using fixed-size widow, i.e., PCA and CHEB, show low performances in Table 4. Their unsatisfactory results are due to the sensitivity to the maximum error constraint, which is often violated using a fixed-size window. Whenever such a violation occurs, those methods store the raw data as they stand into the compression file. We described this process in Section 2.1.1 and Section 2.3.1. This is the main reason why PCA and CHEB reported low compression performances.

GAMPS also shows low compression power; it is ranked as the worst on three datasets. The key reason is that GAMPS is not designed for compressing uncorrelated data, e.g., the datasets used in this benchmark. Therefore, GAMPS may perform better using some different datasets with strong correlations.

4.1.2 Effect of Data Heterogeneity

Figure 6 demonstrates the results of compression ratios using the settings of ϵ= 1%, 5% for each dataset. The longest bar in each test (data) shows the lowest compression power, and the number on top of the bars indicates the exact

dataset	PCA	APCA	PWLH	SF	CHEB	GAMPS
mois	5	1	3	2	4	6
pres	5	3	2	1	6	4
humi	5	1	3	2	4	6
volt	6	1	3	2	4	5
lysi	5	1	3	2	4	6
snow	6	1	2	3	5	4
temp	6	1	2	3	4	5
CO_2	6	1	3	2	4	5
radi	6	1	3	2	4	5
wspd	6	1	3	2	5	4
wdir	6	1	3	2	4	5

TABLE 4: Ranking of the compression methods w.r.t. compression ratio for each dataset.

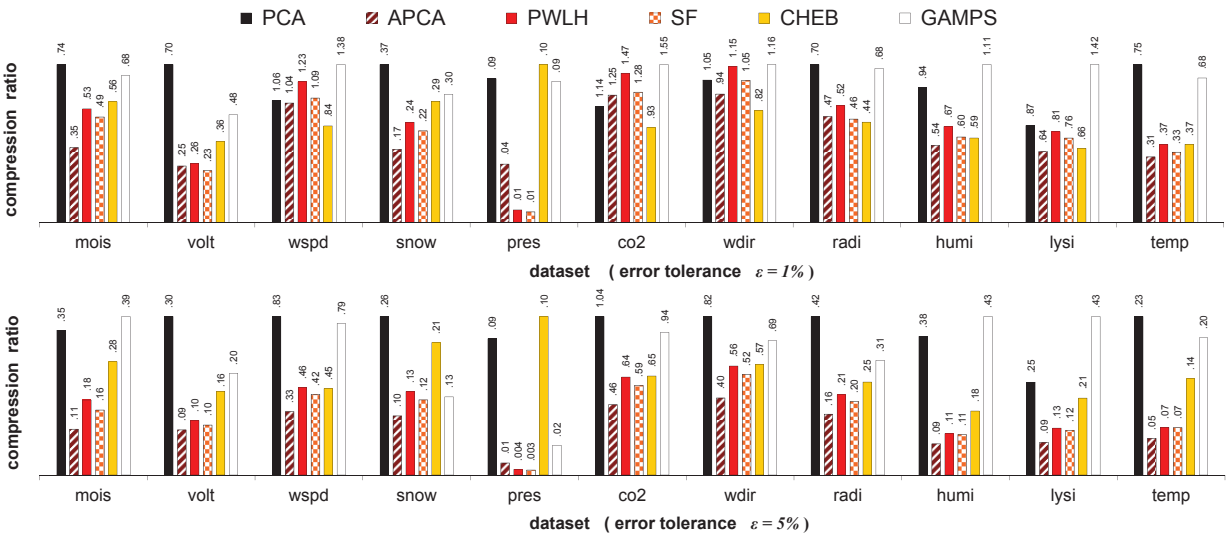


Fig. 6: Comparison of compression ratios (the lower, the better).

compression ratio. The others' results (bars) in each dataset are normalized according to the longest bar. For example, PCA shows the worst compression ratio 0.74 in mois when $\epsilon = 1\%$, and the APCA's compression ratio in this data is about the half of PCA's compression ratio.

In the results, we find many interesting facts. First, the compression power of the methods becomes prominent over the datasets showing relatively smooth changes of value (indicated by 'short-term fluctuation' in Table 3). For example, the worst compression ratios specified by the numbers on top of the longest bars in pres and snow are 0.1 and 0.37 respectively while the numbers in other datasets vary from 0.74 to 1.42. In particular, all the methods achieve very high compressions over pres, even the worst result from CHEB still reaches a 10% compression (i.e., 0.10 compression ratio) at $\epsilon = 1\%$. This is indeed a remarkable result, showing the potential of model-based data compression. The main reason for this outstanding result is that pres shows a high-rate sampling rate, i.e., 2 seconds (see Table 3), but the changes of data values over time are very slight.

In contrast, the datasets containing sharp fluctuation trends, such as wspd, CO_2, and wdir, seem to pose difficulties to the compression algorithms to achieve high compression. This is natural, since all the compression methods are built upon approximation models that cannot precisely capture "randomly" varying data.

Another important finding in Figure 6 is that the methods using variable-length window, in particular APCA, SF, and PWLH, perform between $1/1.59$ to $1/14.3$ better compared to the others over low-volatility datasets. In addition, this observation becomes clearer for medium-fluctuation datasets, when a high value for ϵ is used. In the results of humi and lysi when $\epsilon = 5\%$, for instance, the differences between the adaptive-window and fixed-window compressions are larger than (between 2.13 to 2.52 times) those when $\epsilon = 1\%$ on the same datasets. These results reflect that the adaptability effect of window becomes stronger when the error tolerance grows.

In pres, both PWLH and SF outperform the others, including APCA. Recalling Figure 5, the data values in pres gradually grow with a few sharp drops. Obviously, linear lines are more effective than constant lines to approximate larger numbers of data points in such constantly increasing data. This leads to the higher compression power of the methods built on linear models.

4.1.3 *Effect of Error Tolerance* ϵ

Obviously, the compression power of model-based compression techniques increases, as the value for ϵ increases; however, the degree of this correlation does not follow any particular rule in our results. This was actually another motivation of our benchmark study, which aims to offer a reference for potential users to find an appropriate value for ϵ in their applications.

Table 5 shows the changes of compression ratios of each method, while varying values for ϵ. As the error tolerance gets bigger, from $\epsilon = 1\%$ to 10%, the performance differences among the methods become smaller. For example, in the results from the tests using $\epsilon = 1\%$, the compression ratios of each method are very distinct from one another, yet the differences become smaller in the results obtained from using $\epsilon = 10\%$.

The improvement of compression power with increasing ϵ is clearer when small values are set for ϵ. For instance, the difference in compression ratio of GAMPS between $\epsilon = 1\%$ and $\epsilon = 5\%$ is $(0.76 - 0.45) = 0.31$, while that between $\epsilon = 5\%$ and $\epsilon = 10\%$ is only $(0.45 - 0.29) = 0.16$. Therefore, using a small value for ϵ is more effective in the tradeoff between compression power and error tolerance.

model	ϵ=1%	ϵ=5%	ϵ=10%
PCA	0.76 ± 0.18	0.45 ± 0.17	0.29 ± 0.12
APCA	0.55 ± 0.22	0.17 ± 0.09	0.08 ± 0.04
PWLH	0.66 ± 0.26	0.24 ± 0.12	0.12 ± 0.06
SF	0.59 ± 0.23	0.22 ± 0.11	0.11 ± 0.06
CHEB	0.54 ± 0.15	0.29 ± 0.10	0.20 ± 0.06
GAMPS	0.87 ± 0.27	0.41 ± 0.16	0.24 ± 0.10

TABLE 5: Average compression ratios with 95% confidence interval (all data).

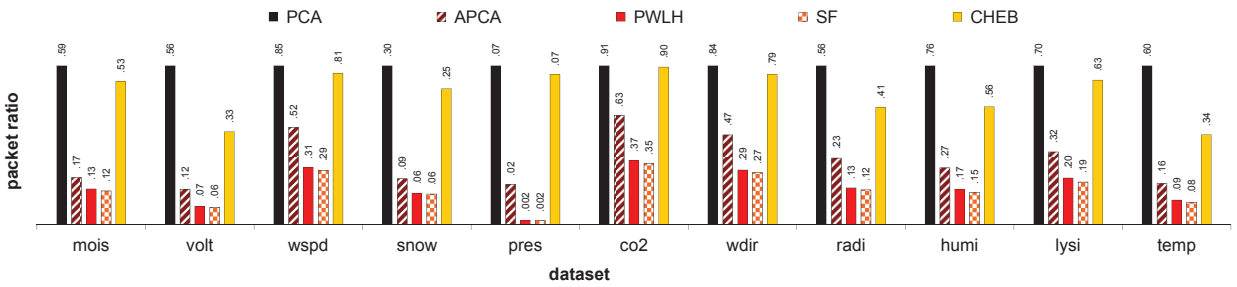


Fig. 7: Comparison of packet ratios when $\epsilon = 1\%$ (the lower, the better).

This fact should be taken into consideration when an application searches an appropriate value for ϵ.

4.2 Transmission Cost

As described in Section 3.4.2, the benchmark simulates data transmission as well as measures transmission cost under two metrics—packet ratio and power ratio. Through a wide range of experiments, we found that the packet ratio is equivalent to the power ratio. Therefore, instead of comparing compression methods in both metrics, we focus on packet ratio only. But note that both metrics are already supported in our benchmark.

In this section, we firstly present experimental results to validate the equivalence between packet ratio and power ratio. After that, we evaluate compression methods in respect of packet ratio.

4.2.1 Equivalence between packet ratio and power ratio

In order to validate the equivalence between packet ratio and power ratio, we design a variety of experiments with broadcast and multi-hop network topology mentioned in Section 3.2.1. In case of broadcast topology, we carried out simulations using different complete graphs K_3, K_6, K_9. For multi-hop topology, simulations were carried out using perfect k-ary tree $T(h,k)$ while varying height h from 2 to 4 and internal degree k from 1 to 3. For each topology setting, we select randomly 300 data streams and apply every compression method.

Figure 8 shows the distribution of all the points whose x and y values are packet ratio and power ratio, respectively.

$$\alpha = \frac{power\_ratio}{packet\_ratio}$$

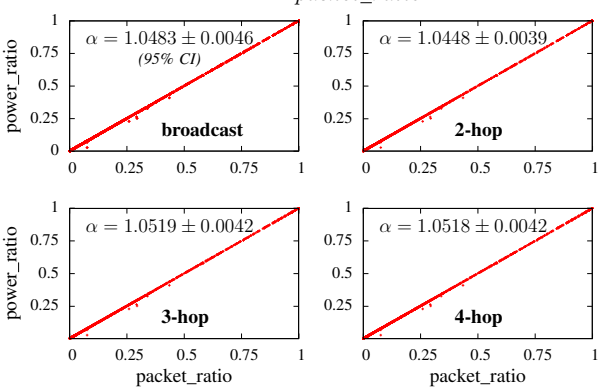


Fig. 8: Equivalence between power ratio and packet ratio

Each point represents a run that is a combination of data stream, compression method and topology setting. We categorize them into four plots: broadcast, 2-hop, 3-hop and 4-hop.

It is clearly seen that all values lie on the angle bisector of the upper right quadrant of the coordinate plane. In all plots, the average value of α—fraction of power ratio over packet ratio—is approximately 1.05 and the 95% confidence interval is lower than 0.005. As a result, we confidently conclude that the packet ratio is equivalent to power ratio (using the PowerTOSSIM-Z simulator with micaZ energy model).

4.2.2 Comparison between compression methods

Figure 7 shows the packet ratio from each compression method across different datasets using error threshold $\epsilon = 1\%$ (we omit the results using $\epsilon = 5\%$ due to the similarity) The results do not contain GAMPS, since GAMPS is an offline compression method, which cannot be applied for the wireless-communication scenario considered in this benchmark.

Clearly, linear-model compressions are winners on this experiment set, indicating that SF and PWLH require fewer data transmissions to update the model parameters from a producer node to a consumer node. Interestingly, PWLH and SF appear to perform between $\frac{1}{1.36}$ to $\frac{1}{33.3}$ times better compared to the other techniques, when a sensor signal exhibits not only smooth fluctuations (e.g., snow and pres), but also high fluctuations (e.g., wspd, CO_2, and wdir). In fact, the performance superiority of PWLH and SF—especially over APCA that was the winner in compression ratio—becomes more prominent, when data values in a signal show high-degree fluctuations. This result implies that the linear approximation models used in SF and PWLH are effective to capture varying sensor signals.

Between PWLH and SF, SF demonstrates slightly better results than those obtained from PWLH, across all tests. Their performance differences are, however, not remarkable (≤ 0.02).

To sum up, our results draw an important suggestion—SF could be a more suitable choice than APCA (the overall winner in the compression ratio experiments) for wireless sensor-network applications.

4.3 Computation Time

Sensor data are often delivered at a high rate; for example, an accelerometer in a smart phone generally produces more

dataset	mois	vol	wspd	snow	pres	co_2	wdir	radi	humi	lysi	temp
compression ratio											
PCA	0.74 ± 0.19	0.70 ± 0.20	1.06 ± 0.08	0.37 ± 0.09	0.09 ± 0.00	1.14 ± 0.07	1.05 ± 0.08	0.70 ± 0.11	0.94 ± 0.07	0.87 ± 0.20	0.75 ± 0.07
APCA	0.35 ± 0.12	0.25 ± 0.08	1.04 ± 0.11	0.17 ± 0.08	0.04 ± 0.00	1.25 ± 0.11	0.94 ± 0.12	0.47 ± 0.14	0.54 ± 0.07	0.64 ± 0.20	0.31 ± 0.05
PWLH	0.53 ± 0.19	0.26 ± 0.08	1.23 ± 0.12	0.24 ± 0.10	0.01 ± 0.00	1.47 ± 0.13	1.15 ± 0.13	0.52 ± 0.15	0.67 ± 0.09	0.81 ± 0.24	0.37 ± 0.07
SF	0.49 ± 0.18	0.23 ± 0.07	1.09 ± 0.11	0.22 ± 0.09	0.01 ± 0.00	1.28 ± 0.12	1.05 ± 0.12	0.46 ± 0.14	0.60 ± 0.08	0.76 ± 0.22	0.33 ± 0.06
CHEB	0.56 ± 0.15	0.36 ± 0.09	0.84 ± 0.07	0.29 ± 0.07	0.10 ± 0.00	0.93 ± 0.06	0.82 ± 0.07	0.44 ± 0.09	0.59 ± 0.06	0.66 ± 0.16	0.37 ± 0.05
GAMPS	0.68 ± 0.08	0.48 ± 0.11	1.38 ± 0.08	0.30 ± 0.07	0.09 ± 0.00	1.55 ± 0.07	1.16 ± 0.08	0.68 ± 0.07	1.11 ± 0.18	1.42 ± 0.06	0.68 ± 0.08
compution time (milisec.)											
PCA	0.09 ± 0.02	0.08 ± 0.02	0.09 ± 0.01	0.05 ± 0.01	0.02 ± 0.00	0.09 ± 0.01	0.09 ± 0.00	0.08 ± 0.01	0.09 ± 0.01	0.08 ± 0.01	0.08 ± 0.01
APCA	0.04 ± 0.01	0.03 ± 0.01	0.07 ± 0.01	0.03 ± 0.00	0.02 ± 0.00	0.08 ± 0.01	0.07 ± 0.01	0.05 ± 0.01	0.05 ± 0.00	0.05 ± 0.01	0.04 ± 0.00
PWLH	48.58 ± 24.28	38.81 ± 29.96	29.44 ± 20.75	33.82 ± 11.02	58.10 ± 5.02	47.00 ± 2.27	39.33 ± 6.28	42.52 ± 16.54	25.89 ± 1.86	38.34 ± 7.62	16.86 ± 3.26
SF	10.52 ± 8.26	15.85 ± 15.44	10.22 ± 7.13	2.01 ± 0.74	15.48 ± 1.28	0.86 ± 0.49	1.03 ± 0.49	12.80 ± 12.94	1.16 ± 0.54	0.50 ± 0.13	1.32 ± 0.53
CHEB	11.76 ± 1.41	9.33 ± 1.44	14.37 ± 0.46	10.32 ± 0.79	10.44 ± 0.15	12.23 ± 0.27	11.93 ± 0.44	13.29 ± 0.47	11.41 ± 0.48	10.10 ± 1.05	11.79 ± 0.20
GAMPS	0.27 ± 0.05	0.05 ± 0.09	1.05 ± 0.16	0.43 ± 0.11	0.40 ± 0.09	0.52 ± 0.10	0.59 ± 0.28	0.98 ± 0.17	0.30 ± 0.22	0.46 ± 0.07	1.50 ± 0.37
packet ratio											
PCA	0.59 ± 0.15	0.56 ± 0.16	0.85 ± 0.07	0.30 ± 0.07	0.07 ± 0.00	0.91 ± 0.06	0.84 ± 0.06	0.56 ± 0.09	0.76 ± 0.06	0.70 ± 0.16	0.60 ± 0.06
APCA	0.17 ± 0.06	0.12 ± 0.04	0.52 ± 0.05	0.09 ± 0.04	0.02 ± 0.00	0.63 ± 0.06	0.47 ± 0.06	0.23 ± 0.07	0.27 ± 0.04	0.32 ± 0.10	0.16 ± 0.03
PWLH	0.13 ± 0.05	0.07 ± 0.02	0.31 ± 0.03	0.06 ± 0.03	0.00 ± 0.00	0.37 ± 0.03	0.29 ± 0.03	0.13 ± 0.04	0.17 ± 0.02	0.20 ± 0.06	0.09 ± 0.02
SF	0.12 ± 0.05	0.06 ± 0.02	0.29 ± 0.03	0.06 ± 0.02	0.00 ± 0.00	0.35 ± 0.03	0.27 ± 0.03	0.12 ± 0.04	0.15 ± 0.02	0.19 ± 0.06	0.08 ± 0.02
CHEB	0.53 ± 0.15	0.33 ± 0.09	0.81 ± 0.07	0.25 ± 0.07	0.07 ± 0.00	0.90 ± 0.06	0.79 ± 0.07	0.41 ± 0.09	0.56 ± 0.06	0.63 ± 0.16	0.34 ± 0.05
RMSE $\times 1000$											
PCA	1.58 ± 0.75	1.61 ± 0.70	0.50 ± 0.14	1.89 ± 0.26	2.10 ± 0.11	0.44 ± 0.20	0.76 ± 0.17	1.22 ± 0.22	1.42 ± 0.19	1.13 ± 0.43	1.99 ± 0.21
APCA	3.58 ± 1.29	4.27 ± 1.24	4.35 ± 0.27	4.30 ± 0.34	5.81 ± 0.01	4.24 ± 0.20	4.21 ± 0.26	5.77 ± 0.57	5.44 ± 0.08	4.65 ± 0.65	5.13 ± 0.31
PWLH	3.53 ± 1.21	3.98 ± 1.09	3.84 ± 0.26	4.06 ± 0.31	6.35 ± 0.09	3.66 ± 0.24	3.60 ± 0.24	4.57 ± 0.41	5.18 ± 0.08	4.63 ± 0.33	5.05 ± 0.21
SF	3.72 ± 1.20	4.11 ± 1.07	4.63 ± 0.28	4.19 ± 0.28	6.45 ± 0.08	4.59 ± 0.23	4.30 ± 0.24	4.94 ± 0.42	5.68 ± 0.08	5.15 ± 0.39	5.31 ± 0.18
CHEB	1.91 ± 0.59	2.18 ± 0.56	1.11 ± 0.19	1.90 ± 0.28	2.08 ± 0.11	0.68 ± 0.21	1.28 ± 0.22	1.93 ± 0.28	2.71 ± 0.15	1.96 ± 0.35	2.99 ± 0.19
GAMPS	0.65 ± 0.22	0.24 ± 1.81	1.26 ± 0.04	2.25 ± 0.62	2.36 ± 0.00	1.31 ± 0.05	1.40 ± 0.06	2.07 ± 0.18	1.68 ± 0.17	1.41 ± 0.21	1.89 ± 0.17

TABLE 6: Average value and 95% confidence interval of all evaluation measures of each method over all datasets ($\epsilon = 1\%$).

than 1,000 readings within a minute. As a result, quickly compressing a given sensor signal is a key factor.

APCA and PCA are clear winners on this concern. In Table 6, APCA and PCA's computation time is less than 0.1 ms across all data sets. GAMPS and SF are slower as their values varies within the range $[0.27, 1.5]$ and $[0.5, 15.85]$ respectively. Finally, the remaining techniques—CHEB and PWLH—exhibit expensive computation as their running time is over 9 ms in their best case. This is because CHEB transformation takes relatively sophisticated computation, and PWLH needs to compute convex hulls to maintain the linear window every time when a new data point is added to the window.

Overall, we conclude that PCA, APCA, GAMPS, SF are affordable to use at low-speed devices, whereas CHEB and PWLH may be inappropriate for many sensor network applications, in particular those who deal with a large number of sensor data streams.

4.4 Approximation Quality

In order to reflect the quality of compression, in which the intuition behind this measure was explained in Section 3.4.4, the benchmark computed the RMSEs of data compression when each test ran. The results are presented in Figure 9 and Table 6.

A noticeable observation in the results is that the RMSEs obtained from PCA and CHEB are significantly lower than (2.78 times in average) those from the others. This observation may be interpreted as "the higher compression power, the lower quality of data compression". This is, however, not what we believe. Turning to the processes of how PCA and CHEB produce compressed data (described in Sections 2.1.1 and 2.3.1), those two methods store raw data points as they stand into the compressed file, when the data approximation in a window (segment) violates a given maximum error guarantee ϵ. Clearly, an RMSE for those data points in the compressed file is 0. As both methods generally do not achieve high compression ratios, such raw data points preserved in the compressed data are likely to present a lot. This fact results in the low RMSE values from PCA and CHEB.

Among the results reported from the adaptive-window methods, which are APCA, SF, and PWLH, the differences of RMSE are negligible.

4.5 Sensitivity to Outliers

A variety of reasons cause noises or outliers in sensor data, such as discharged batteries, network failures, accumulation of dirt, and errors from low-cost sensors. As a result, it is important for sensor-network applications to know how each method performs when data are not clean.

In this experiment set, we compared the robustness to noisy data of the compression methods. As we discussed in Section 3.4, we put artificial outliers to the datasets, having different appearance probabilities, from $p = 0\%$ to $p = 4\%$. More precisely, given a data stream, we first compute the relative error rate (in percentage) and then add outliers randomly into this stream. The reason for this setting is to avoid the side-effects of adding outliers on the relative error rate as the max/min values could be changed by these outliers. Figure 10 demonstrates the compression ratios of each technique along with varying p.

Interestingly, GAMPS shows low sensitivity to noisy data; the compression ratios of GAMPS do not vary substantially as the value for p increases. This trend looks similar to the results from APCA. This is reasonable. As we described in Section 2.4.1, GAMPS uses APCA to approximate individual signals, and then groups the APCA-applied signals if they are similar. Therefore, the performance behaviors in terms of the sensitivity to outliers between GAMPS and APCA are essentially similar to each other.

Another key finding in Figure 10 is that fixed-window methods (i.e., PCA and CHEB) demonstrate high sensitivity to outliers. The main reason can be found by understanding how outliers affect the process of data approximation. Outliers generally produce more segments during data approximation in the compression methods, since they are likely to incur more frequent violations for the error tolerance. Once such a violation occurs, PCA and CHEB store the original data in the compressed data; while adaptive-window methods still store approximated data as compression, and

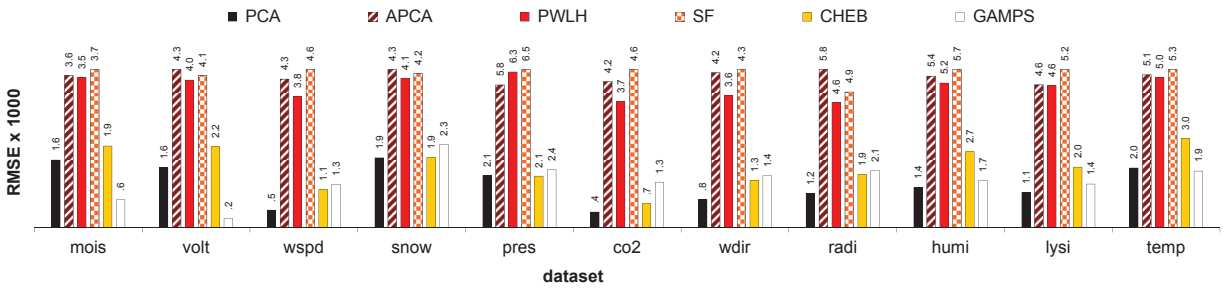


Fig. 9: Comparison of compression qualities (RMSE) when $\epsilon = 1\%$ (the lower, the better).

simply start a new approximation. Therefore, even the same amount of outliers can decrease the compression power of the fixed-window methods more.

Between PWLH and SF, SF shows slightly better results than PWLH; their differences are less than 0.02. This is because SF uses multiple lines to select the best one while approximating data, rendering the approximation process of SF more flexible to noisy data, compared to PWLH.

5 Summary and Conclusions

This paper presented a thorough evaluation and comparison of model-based compression techniques widely used in sensor data processing. We offered an overview of four major classes of model-based approaches, while discussing about the characteristics of the underlying models. We then introduced the flexible and extensible benchmark framework, in which a new compression method as well as a new dataset can be easily plugged. During the framework development, we made the best effort to re-implement the most representative model-based compression techniques, and evaluated them in a fair manner. We also analyzed various performance factors for each compression method, including compression ratio, computation time, packet ratio, power ratio, approximation error, and robustness to noise, using real data signals collected from 346 different sensors.

We here summarize our principal findings as a set of recommendations for how to select a well-suited compression method on particular application scenarios:

- Overall, APCA and SF performed best. In particular, they by far outperformed the others, when data values in a sensor signal exhibit low fluctuations.
- The adaptability of window (i.e., varying-size data segment) substantially increases the compression power. This effect becomes even more prominent when the volatility of data is high. Therefore, subsequent studies should take into account this fact when they design a new compression method for sensor data.
- Compression quality in terms of RMSE is also greatly affected by the adaptability of window, but in an opposite way; adaptive-window methods, i.e., APCA, SF, and PWLH, show high RMSEs, compared with fixed-window methods—PCA and CHEB.
- The idea of GAMPS is interesting, but it turns out to be ineffective when sensor signals show a diversity in sampling rate as well as less correlations.
- For efficient data communication, linear-model compression techniques are promising. Especially, SF can be a good choice to facilitate infrequent data communication in wireless or in-network sensing environments.
- If sensor data are noisy (i.e., containing many outliers), consider using GAMPS, especially when a high degree of data correlation is expected. APCA can also work robustly against noisy data, particularly useful when data do not show correlations.

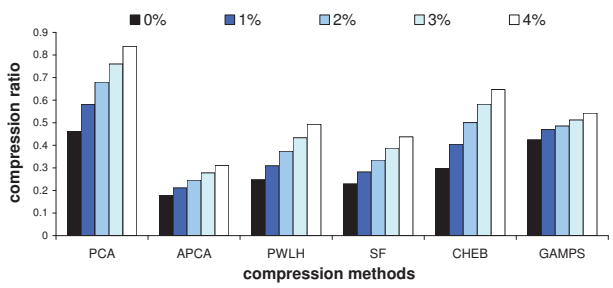


Fig. 10: Sensitivities to noise, $outlier$= 0%, 1%, 2%, 3%, 4% of original data.

category	winner	2nd best	worst
compression ratio	**APCA**	SF	PCA
computation time	**APCA**	PCA	PWLH
packet ratio	**SF**	PWLH	PCA
power ratio	**SF**	PWLH	PCA
RMSE of compression	**PCA**	CHEB	APCA
robustness to noise	**GAMPS**	APCA	PCA

As a concluding remark, we recommend a potential application to use our benchmarking framework as a tool to find out the best-suited compression technique accordingly, since there is no absolute winner that outperforms the others in every case. As the benchmark source code as well as the datasets used in the benchmark are publicly available, we expect that the experimental results presented in this paper will be refined and improved by the research community, in particular when more data become available, more experiments are performed, and more techniques are integrated into the framework in the future.

References

[1] A. Arion, H. Jeung, and K. Aberer. "Efficiently Maintaining Distributed Model-Based Views on Real-Time Data Streams". In: *GLOBECOM*. 2011, pp. 1–6.

[2] K. C. Barr and K. Asanovic. "Energy aware lossless data compression". In: *MobiSys*. 2003, pp. 231–244.

[3] C. Buragohain, N. Shrivastava, and S. Suri. "Space Efficient Streaming Algorithms for the Maximum Error Histogram". In: *ICDE*. 2007, pp. 1026–1035.

[4] M. Burrows and D. J. Wheeler. "A Block-sorting Lossless Data Compression Algorithm". In: (1994).

[5] Y. Cai and R. Ng. "Indexing spatio-temporal trajectories with Chebyshev polynomials". In: *SIGMOD*. 2004, pp. 599–610.

[6] D. Carney et al. "Monitoring streams: a new class of data management applications". In: *VLDB*. 2002, pp. 215–226.

[7] Q. Chen et al. "Indexable PLA for efficient similarity search". In: *VLDB*. 2007, pp. 435–446.

[8] D. Chu et al. "Approximate Data Collection in Sensor Networks using Probabilistic Models". In: *ICDE*. 2006, p. 48.

[9] X. T. Dang, N. Bulusu, and W. chi Feng. "RIDA: A robust information-driven data compression architecture for irregular wireless sensor networks". In: *EWSN*. 2007, pp. 133–149.

[10] A. Deligiannakis, Y. Kotidis, and N. Roussopoulos. "Compressing Historical Information in Sensor Networks". In: *SIGMOD*. 2004, pp. 527–538.

[11] Y. Diao et al. "Rethinking Data Management for Storage-centric Sensor Networks". In: *CIDR*. 2007, pp. 22–31.

[12] H. Ding et al. "Querying and mining of time series data: experimental comparison of representations and distance measures". In: *PVLDB* 1.2 (2008), pp. 1542–1552.

[13] H. Elmeleegy et al. "Online piece-wise linear approximation of numerical streams with precision guarantees". In: *PVLDB* 2.1 (2009), pp. 145–156.

[14] E. Elnahrawy and B. Nath. "Cleaning and querying noisy sensors". In: *WSNA*. 2003, pp. 78–87.

[15] E. Elnahrawy and B. Nath. "Online data cleaning in wireless sensor networks". In: *SenSys*. 2003, pp. 294–295.

[16] C. A. F. et al. "Data compression using Chebyshev transform". Patent US 7,249,153 B2. 2007.

[17] S. Gandhi, L. Foschini, and S. Suri. "Space-efficient online approximation of time series data: Streams, amnesia, and out-of-order". In: *ICDE*. 2010, pp. 924–935.

[18] S. Gandhi et al. "GAMPS: compressing multi sensor data by grouping and amplitude scaling". In: *SIGMOD*. 2009, pp. 771–784.

[19] N. Giatrakos et al. "TACO: tunable approximate computation of outliers in wireless sensor networks". In: *SIGMOD*. 2010, pp. 279–290.

[20] R. Graham. "An Efficient Algorithm for Determining the Convex Hull of a Finite Planar Set". In: *Information Processing Letters*. 1972, pp. 132–133.

[21] S. R. Jeffery, M. Garofalakis, and M. J. Franklin. "Adaptive cleaning for RFID data streams". In: *VLDB*. 2006, pp. 163–174.

[22] C. John and W. Ian. "Data Compression Using Adaptive Coding and Partial String Matching". In: *TCOM* 32.4 (1984), pp. 396–402.

[23] E. Keogh et al. "Dimensionality Reduction for Fast Similarity Search in Large Time Series Databases". In: *Knowl. Inf. Syst.* 3.3 (2000), pp. 263–286.

[24] E. Keogh et al. "Locally adaptive dimensionality reduction for indexing large time series databases". In: *SIGMOD*. 2001, pp. 151–162.

[25] E. J. Keogh et al. "An Online Algorithm for Segmenting Time Series". In: *ICDM*. 2001, pp. 289–296.

[26] N. Kimura and S. Latifi. "A Survey on Data Compression in Wireless Sensor Networks". In: *ITCC*. 2005, pp. 8–13.

[27] A. Klein. "Incorporating quality aspects in sensor data streams". In: *PIKM*. 2007, pp. 77–84.

[28] I. Lazaridis and S. Mehrotra. "Capturing Sensor-Generated Time Series with Quality Guarantees". In: *ICDE*. 2003, pp. 429–440.

[29] P. Levis et al. "TOSSIM: accurate and scalable simulation of entire TinyOS applications". In: *SenSys*. 2003, pp. 126–137.

[30] X. Lian and L. Chen. "Efficient processing of probabilistic reverse nearest neighbor queries over uncertain data". In: *PVLDB* 18.3 (2009), pp. 787–808.

[31] C. Olston, J. Jiang, and J. Widom. "Adaptive filters for continuous queries over distributed data streams". In: *SIGMOD*. 2003, pp. 563–574.

[32] E. Perla et al. "PowerTOSSIM z: realistic energy modelling for wireless sensor network environments". In: *PM2HW2N*. 2008, pp. 35–42.

[33] G. Reeves et al. "Managing Massive Time Series Streams with Multi-Scale Compressed Trickles". In: *VLDB*. Vol. 2. 1. 2009, pp. 97–108.

[34] K. Sayood. *Introduction to Data Compression*. 2000.

[35] T. Schoellhammer et al. "Lightweight Temporal Compression of Microclimate Datasets". In: *LCN*. 2004, pp. 516–524.

[36] R. Shumway and D. Stoffer. *Time Series Analysis and Its Applications*. Springer-Verlag, 2005.

[37] S. Subramaniam et al. "Online outlier detection in sensor data using non-parametric models". In: *VLDB*. 2006, pp. 187–198.

[38] Y. L. Tan, V. Sehgal, and H. H. Shahri. *SensoClean: Handling Noisy and Incomplete Data in Sensor Networks using Modeling*. Tech. rep. University of Maryland, 2005.

[39] D. Tulone and S. Madden. "PAQ: Time Series Forecasting for Approximate Query Answering in Sensor Networks." In: *EWSN*. 2006, pp. 21–37.

[40] T. Westmann et al. "The implementation and performance of compressed databases". In: *SIGMOD Record* 29.3 (2000), pp. 55–67.

[41] J. Yick, B. Mukherjee, and D. Ghosal. "Wireless sensor network survey". In: *Computer Networks* 52.12 (2008), pp. 2292–2330.

[42] L. Yu et al. "Enabling ϵ-approximate querying in sensor networks". In: (2009), pp. 563–574.

[43] Y. Zhang, N. Meratnia, and P. Havinga. "Outlier Detection Techniques For Wireless Sensor Networks: A Survey". In: *Journal of IEEE Communications Survey & Tutorials* 12.2 (2010).

[44] J. Ziv and A. Lempel. "A Universal Algorithm for Sequential Data Compression". In: *TIT* 23.3 (1977), pp. 337–343.

[45] J. Ziv and A. Lempel. "Compression of Individual Sequences via Variable-Rate Coding". In: *TIT* 24.5 (1978), pp. 530–536.

Nguyen Quoc Viet Hung is PhD student at Ecole Polytechnique Federale de Lausanne (EPFL), Switzerland. He obtained a BSc and a MSc in Computer Science from Ho Chi Minh City University of Technology (HCMUT), and a MSc in Computer Science from EPFL. His research interests are Spatio-Temporal Data Management and Data Integration.

Hoyoung Jeung is a senior researcher at SAP Research Brisbane, Australia. Prior to SAP, he worked as a postdoctoral researcher in the Swiss Federal Institute of Technology (EPFL), conducting research in a wide spectrum of data- and computing-intensive systems in computer science: databases, sensor networks, data mining, cloud computing, distributed systems, and in-memory computing. Many of his research works have been published at top-tier conferences and journals including VLDB, ICDE and VLDBJ. He earned his PhD in computer science at the University of Queensland Australia, while assisting various research projects carried out by NICTA QLD/NSW as well as Aalborg university in Denmark.

Karl Aberer is a full professor for Distributed Information Systems at EPFL Lausanne, Switzerland, since 2000. His research interests are on decentralization and self-organization in information systems with applications in peer-to-peer search, semantic web, trust management and mobile and sensor networks. Karl Aberer received his Ph.D. in mathematics in 1991 from the ETH Zurich. From 1991 to 1992 he was postdoctoral fellow at the International Computer Science Institute (ICSI) at the University of California, Berkeley. In 1992 he joined the Integrated Publication and Information Systems institute (IPSI) of GMD in Germany, where he was leading the research division Open Adaptive Information Management Systems. Since 2005 he is the director of the Swiss National Research Center for Mobile Information and Communication Systems (NCCR-MICS, www.mics.ch). He is member of the editorial boards of VLDB Journal, ACM Transaction on Autonomous and Adaptive Systems and World Wide Web Journal. He has also been consulting for the Swiss government in research and science policy as a member of the Swiss Research and Technology Council (SWTR) from 2004 to 2011.