Compressing Historical Information in Sensor Networks

Antonios Deligiannakis[*]
University of Maryland
adeli@cs.umd.edu

Yannis Kotidis
AT&T Labs-Research
kotidis@research.att.com

Nick Roussopoulos
University of Maryland
nick@cs.umd.edu

ABSTRACT

We are inevitably moving into a realm where small and inexpensive wireless devices would be seamlessly embedded in the physical world and form a wireless sensor network in order to perform complex monitoring and computational tasks. Such networks pose new challenges in data processing and dissemination because of the limited resources (processing, bandwidth, energy) that such devices possess. In this paper we propose a new technique for compressing multiple streams containing historical data from each sensor. Our method exploits correlation and redundancy among multiple measurements on the same sensor and achieves high degree of data reduction while managing to capture even the smallest details of the recorded measurements. The key to our technique is the *base signal*, a series of values extracted from the real measurements, used for encoding piece-wise linear correlations among the collected data values. We provide efficient algorithms for extracting the base signal features from the data and for encoding the measurements using these features. Our experiments demonstrate that our method by far outperforms standard approximation techniques like Wavelets, Histograms and the Discrete Cosine Transform, on a variety of error metrics and for real datasets from different domains.

1. INTRODUCTION

Technological advances in the development of low-power embedded communication devices have made possible scenarios in which thousands of sensor nodes are seamlessly embedded in the physical world and form a wireless sensor network. These sensors monitor various quantities such as temperature, pressure, humidity, movement, noise levels, chemicals, etc. that are periodically transmitted to a *base-station* for further processing and analysis. A base-station may represent any node of the network with increased storage, battery and processing capabilities. Applications of such networks span a large variety of domains from collaborative environments to military command and control systems.

[*]Work partially done while author was visiting AT&T Labs-Research. Author also supported by the DoD-Army Research Office under Awards DAAD19-01-2-0011 and DAAD19-01-1-0494.

Large-scale sensor networks require tight data handling and data dissemination techniques. Transmitting a *full-resolution* data feed from each sensor back to the base-station is often prohibitive due to (i) limited bandwidth that may not be sufficient to sustain a continuous feed from all sensors and (ii) increased power consumption due to wireless multi-hop communication. In order to minimize the volume of the transmitted data, we can apply two well known ideas: *aggregation* and *approximation*. Aggregation works by summarizing the measurements in the form of simple statistics like average, maximum, minimum etc. that are then transmitted to the base-station over regular intervals. Aggregation is an effective mean to reduce the volume of data, but is rather crude for applications that need detailed historical information (like military surveillance). When data feeds exhibit a large degree of redundancy, approximation is a less intrusive form of data reduction in which the underlying data feed is replaced by an approximate signal tailored to the application needs. The trade-off is then between the size of the approximate signal and its precision compared to the real-time information monitored by the sensor.

In this paper we present a data reduction algorithm for the dissemination of historical measurements in constraint environments, such as sensor networks. Our techniques build on the observation that the values of the collected measurements exhibit similar patterns over time, or that different measurements are naturally correlated, as is the case between pressure and humidity in weather monitoring applications. At the core of our approximation lies the notion of a *base signal*, a set of values from the collected measurements that capture prominent features of the data. Following the construction of the *base signal*, the collected data is partitioned into intervals that can be efficiently approximated as linear projections of some part of the *base signal*. As we will show in this paper, our techniques provide:

- *Increased accuracy and robustness when compared to other approximation techniques:* Our algorithm feeds from intrinsic redundancy among multiple values (like many data reduction techniques), but has full control over the data model used to exploit these redundancies (which values to insert into the base signal), the amount of space allocated for this data model and the number of coefficients that describe the projections of the data values on this data model. Due to this fact, our algorithm produced up to 27 and 1000 times smaller errors than the next best competitive method for the sum squared and the sum squared relative error metrics, respectively, Moreover, due to its fall-back mechanism to linear regression, as we will later explain, it performs at least as good as linear regression, but in practice is significantly more accurate, for the same data reduction factor.
- *Adaptability to different error metrics:* Our algorithms can

be adapted with only minor modifications, *which do not alter their time complexity*, to minimize different error metrics, such as the sum squared error, sum squared relative error, and maximum error of the approximation. We further discuss extensions when the application requires strict error bounds or a combination of error and space bounds.

Our contributions are summarized as follows:

1. We introduce a new approximation scheme that encodes piecewise correlations among the values of multiple data streams. Such correlations are often linear in nature and can be easily captured by standard techniques like linear regression. We exploit correlations both within the values of a single measurement (ex: periodicity, self-similarity) as well as among values of different quantities (ex: pressure and humidity).

2. We introduce the concept of the *base signal* that is analogous to a *carrier-wave* in radio-frequency transmissions and is used for encoding the measurements. We explore the technical challenges of (i) constructing the base signal, (ii) approximating the recorded measurements by exploring piecewise correlations amongst them and the base signal, and (iii) dynamically updating the base signal to capture new data trends in subsequent transmissions.

3. We provide an efficient algorithm (Self-Based Regression or *SBR*) that answers all questions above, while balancing the cost of transmitting new (or updated) base signal values with the gains of using them for approximating the data values. For a dataset containing n measurements to approximate, SBR takes $O(n^{1.5})$ time and requires linear space, while its running time scales linearly to the size of both the transmitted data and the base signal. The SBR algorithm is invoked periodically, when enough historical data has being collected on the sensor. Thus, its processing cost is amortized over the input size. In experiments on a low-end 300MHz CPU, SBR was processing up to 1,000 data items per second. More importantly, as our results demonstrate, few initial invocations of the SBR algorithm are crucial for the population of the base signal, after which the base signal is either rarely updated, or in very small parts, since it is already of good quality. This permits us to shortcut some parts of the algorithm in constrained environments, as is the case in sensor network applications, and perform their execution only periodically (i.e., when we notice a degradation in the quality of the approximation). These shortcuts significantly reduce the complexity of the algorithm, resulting in a linear dependency to the size of the processed data. Thus, our framework is not only suitable for sensor networks (where the data sampling rates are significantly slower than the rates that SBR processes data) but may have applications in other areas where historical information is being collected in a distributed fashion, like network measurements.

4. We provide an extensive experimental study of our framework using real datasets from different domains and make direct comparisons against previously studied approximation techniques like the Wavelet and Discrete Cosine Transforms and Histograms. In all datasets our method achieves substantially lower approximation errors for the same data reduction factor. Even in the case of datasets containing values from entirely different applications with no apparent correlations amongst them, the SBR algorithm managed to identify piecewise correlations between parts of these signals and significantly outperform all the other competitive methods.

5. We have adapted ideas from the Singular Value Decomposition and the Discrete Cosine Transform for constructing alternative base signals. Our experiments demonstrate that the base signal features selected by SBR outperform these techniques, by a factor of ten in some datasets. Furthermore, we show that SBR makes near-optimal choices when balancing the number of features to include in the base signal over the number of coefficients used to compress the data values using these features.

While our techniques are motivated by applications of sensor networks, the same algorithms can also be used in other domains for the lossy compression of multiple time series. In addition to weather data that is common in sensor network applications, we also provide experimental evidence using phone-call and stock datasets. In these domains where significantly more powerful processors can be used than in sensor network applications, some of the optimizations proposed in this paper may be relaxed in order to achieve even better approximation; for example, a larger number of candidate intervals for inclusion in the base signal may be considered.

The rest of the paper is organized as follows. Section 2 presents related work. In Section 3 we state our problem and sketch the basics of our techniques, while in Section 4 we describe our framework in more detail. Section 5 contains our experiments. Section 6 presents concluding remarks and future directions.

2. RELATED WORK

In recent years there has been a flurry of research in the area of sensor networks. Some of the most important issues addressed include network self-configuration [2] and data discovery [10, 13]. In-network data aggregation is investigated in [8, 11, 15, 18, 30]. The main idea is to build an aggregation tree, which partial results will follow. Non-leaf nodes of the tree aggregate the values of their children before transmitting the aggregate result to their parents, thus reducing substantially the number of messages in the network.

Sensor nodes are small devices that "measure" their environment and communicate streams of low-level values to a base station for further processing and archiving. These streams are then used to construct a higher-level model of the environment. This process makes historical data equally important to current values [7]. In this paper we propose approximation as a less intrusive data reduction method that is more suited for applications in which a long-term historical record of measurements from each sensor is required.

Recently, there has been increasing interest in studying the general principles over continuous queries in data streams [4, 14, 21, 28, 31]. The work of [23] studies the trade-off between precision and performance when querying replicated, cached data. In [22] the users register continuous queries with strict precision constraints at a central *stream processor*, which, in turn installs filters at the remote data sources. These filters adapt to changes in the streams to minimize update rates. The work in [8] investigates the optimal assignment of error filters in order to minimize the bandwidth consumption of continuous aggregate queries in sensor networks. An online algorithm for minimizing the update cost while the query can be answered within an error bound is presented in [16]. The authors of [6] study a probabilistic query evaluation method that places appropriate confidence in the query answer to quantify the *uncertainty* of the recorded data values.

Approximate processing techniques have been widely studied. Histograms ([24, 26]) have been extensively used by query optimizers to estimate the selectivity of queries, and recently in tools for providing fast approximate answers to queries. Wavelets are a mathematical tool for the hierarchical decomposition of functions, with applications in image and signal processing. More recently, Wavelets have been applied successfully in answering range-sum aggregate queries over data cubes [29], in selectivity estimation [20] and in approximate query processing [3]. The Discrete Cosine Transform (DCT) [1] constitutes the basis of the *mpeg* encoding algorithm and has also been used to construct compressed multi-dimensional histograms [17]. Linear regression has been recently used in [5] for on-line multidimensional analysis of data streams.

The use of a dictionary is a standard technique in data compression algorithms (for example in gzip). As an abstraction, the base signal is a dynamic data dictionary that is used to compress present and future values. A fundamental difference with compression techniques such as gzip is that our compression is lossy, which allows us to achieve significantly higher compression ratios. Furthermore, the details of our approximation (construction of the base signal, approximation using regression etc.) differ at a fundamental level from standard compression techniques. Many popular transforms also use some form of basis that is either fixed (ex: Wavelets, DCT) or data dependent (ex: SVD). We also explore the use of such bases in our framework and present the necessary modifications. However, the base signal constructed by our algorithms seems to always outperform these bases, for the specific encoding we use in our approximation.

3. PRELIMINARIES

In this section we first present a description of the characteristics of sensor networks and their applications. We then describe the operation of sensor nodes in our data reduction framework and present the optimization problems that we tackle in this paper.

3.1 Characteristics of Sensor Networks

Recent technological advances have made possible the development of low-cost sensor nodes with heavily integrated sensing, processing and communication capabilities. Information about the environment is gathered using a series of sensing elements connected to an analog-to-digital converter. Examples include microphones for acoustic sensing, accelerometers and temperature sensors. Once enough data is collected, it is processed locally and periodically forwarded to a base station, using a multi-hop routing protocol [27].

The processing subsystem on the nodes depends on the nature of the application. Applications such as military reconnaissance that require significant processing to be performed at the nodes use sensor nodes with significant processing power. As an example, an improved model of the commonly used StrongARM 1100 processor (μAMPS [27] and HiDRA nodes) reaches a frequency of 400 MHz and can support up to 64 MB of memory.

As the processing and storage capabilities of sensor nodes tend to follow Moore's Law their communication and power subsystems become the major bottleneck of their design. For example, over the last years, the energy capacity of the batteries used in such nodes has exhibited a mere 2-3% annual growth.[1] The main source of energy consumption in a node is the data transmission process. There are several reasons for this:

1. The energy drain during transmission is much larger than the consumption during processing [10]. As an example, on a Berkeley MICA Mote sending one bit of data costs as much energy as 1,000 CPU instructions [19]. Faster processors typically achieve lower power consumptions per CPU instruction, making the above ratio even larger.

[1]http://nesl.ee.ucla.edu/courses/ee202a/2002f/lectures/L07.ppt

2. Transmission ranges between nodes are fairly short. The transmitted data may thus require to traverse multiple hops to reach the base station. This retransmission process at each intermediate node is very costly. Furthermore, because nodes often use broadcast protocols over radio frequencies [18], transmitted messages are not only received by the intended node, but by all nodes in the vicinity of the sender, thus increasing the overall power consumption.

Even on applications where battery lifetime is not a concern (ex: military surveillance sensing nodes attached to moving vehicles with practically infinite power supply), the available bandwidth may not sustain a continuous feed of measurements for all sensors deployed in the terrain. The design of data reduction protocols that effectively reduce the amount of data transmitted in the network is thus essential when the goal is to meet the application's bandwidth constraints or to increase the network's lifetime.

3.2 Data Model and Processing

In order not to deplete their power supply (and to conserve bandwidth), the sensors do not continuously transmit every new measurement they obtain but rather wait till enough data is collected and then forward it to the base station [27]. This form of batch processing allows them to power-down their radio transmitter and prolong their lifetime in a way analogous to [18].

Within a sensor, the recorded data is depicted in a two dimensional array where each row i stores sampled values of a distinct quantity. Informally, each row i is a time series $\vec{Y_i}$ of samples from quantity i collected by the sensor. The array has N rows, N being the number of recorded quantities and M columns, where M depends on the available memory.[2]

As more measurements are obtained, the sensor's memory buffers become full. At this point the latest NxM values are processed and each row i (of length M) is approximated by a much smaller set of B_i values, i.e. $B_i \ll M$. The resulting "compressed" representation, of total size equal to $B = \sum_{i=1}^{N} B_i$, is then transmitted to the base station. The base station maintains the data in this compact representation by appending the latest "chunk" to a log file. A separate file exists for each sensor that is in contact with the base station. The entire process is illustrated in Figure 1.

Each sensor allocates a small amount of memory of size M_{base} for what we call the *base signal*. This is a compact ordered collection of values of prominent features that we extract from the recorded values and are used as a base reference in the approximate representation that is transmitted to the base station (details will be given in the next section). The data values that the sensor transmits to the base station are encoded using the in-memory values of the base signal at the time of the transmission. The base signal may be updated at each transmission to ensure that it will be able to capture newly observed data features and that the obtained approximation will be of good quality. When such updates occur, they are transmitted along with the data values and appended in a special log file that is unique for each sensor. This allows the base station to reconstruct (approximately) the series $\vec{Y_i}$ at any given point in the past.

3.3 Our Optimization Problem

We can think of the base signal as a dictionary of features used to describe the data values. The richer the pool of features we store in the base signal the better the approximation. On the other hand,

[2]We here assume that all quantities are sampled with the same frequency. This simplifies notation, however, our framework also applies when each quantity is recorded on a different schedule.

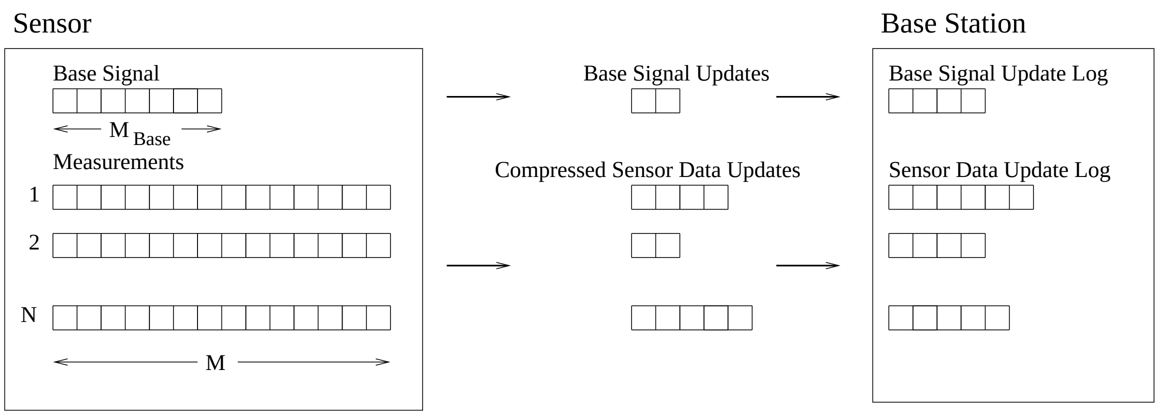

Figure 1: Transfer of approximate data values and of the base signal from each sensor to the base station

Configuration Parameters	
N	Number of input signals
M	Measurements per input signal
Input Parameters	
$TotalBand$	Total bandwidth per transmission
M_{base}	Buffer size for base signal values
Derived/Calculated Parameters	
$n = N \times M$	Size of in-memory data
$W = \sqrt{n}$	Size of each base interval
B	Compressed Data Size
$maxIns$	Maximum number of base intervals inserted in current transmission
Ins	Number of base intervals actually inserted in the current transmission

Table 1: Configuration, input and derived parameters of our algorithms

these features have to be (i) kept in the memory of the sensor to be used as a reference by the reduction algorithm and (ii) sent to the base station in order for it to be able to reconstruct the values. Thus, for a target bandwidth constraint (number of values that can be transmitted) the more insert and update operations on the base signal that we perform, the less bandwidth is left available for approximating the data values. Moreover, the time to perform the data approximation increases, in our algorithms, linearly with the size of the base signal.

In the next section we present an efficient algorithm that decides (i) how large the base signal needs to be at each transmission (ii) what new features to be included in it (iii) which older features are not relevant any more and (iv) how to best approximate the data measurements using these features. The only user input needed by the algorithm is the target bandwidth constraint and the maximum buffer size of the base signal values.

4. THE SBR FRAMEWORK

We now describe our framework in more detail. We start with a motivational example that demonstrates the intuition behind our techniques. Subsection 4.2 presents the primitive operations required by our framework while the *SBR* algorithm is presented in subsection 4.3. Table 1 contains a brief description of the parameters used in our algorithms.

4.1 Motivational Example

Many real signals are correlated. We expect this to be particularly true for measurements taken by a sensor, especially if they are physical quantities like temperature, dew-point, pressure etc. The same is often true in other domains. For example, in Figure 2 we plot the average Industrial and Insurance indexes from the New York stock market for 128 consecutive days.[3] Both signals show similar trends, i.e. they go up and down together. Figure 3 depicts a XY scatter plot of the same values. This is created by pairing values of the Industrial (X-coordinate) and Insurance (Y-coordinate) indexes of the same day and plotting these points in a two-dimensional plane. The strong correlation among these values makes most points lie on a straight line. This observation motivates our work. Assuming that the Industrial index (call it $\vec{X}$) is given to us in a time-series of 128 values, we can approximate the other time-series (Insurance: $\vec{Y}$) as:

$$\vec{Y'} = a * \vec{X} + b$$

The coefficients a and b are determined by the condition that the sum of the square residuals, or equivalently the L_2 error norm $||\vec{Y'} - \vec{Y}||_2$, is minimized. This is nothing more than standard linear regression. However, unlike previous methods, we will not attempt to approximate each time-series independently using regression. In Figure 2 we see that the series themselves are not linear, i.e. they would be poorly approximated with a linear model. Instead, we will use regression to approximate piece-wise correlations of each series to a base signal that we will choose accordingly. In the example of Figure 3 the base signal can be the Industrial index ($\vec{X}$) and the approximation of the Insurance index will be just two values (a, b). In practice the base signal may be much smaller than the complete time series, since it only needs to contain the "important" trends of the target signal $\vec{Y}$. For instance, in case $\vec{Y}$ is periodic, a sample of the period would suffice. Our algorithm breaks the latest measurements obtained by the sensor into small intervals (of varying sizes) and looks for intervals of the same length in the base signal that are linearly correlated. At the same time, the base signal values are evaluated and may get updated with features from the newly collected measurements when necessary.

4.2 Primitives of our Implementation

[3]Data at http://www.marketdata.nasdaq.com/mr4b.html

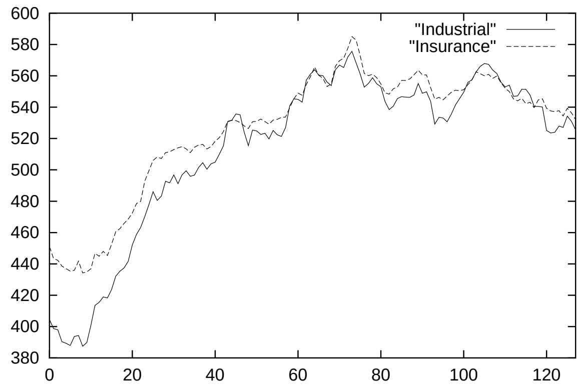

Figure 2: Example of two correlated signals (Stock Market)

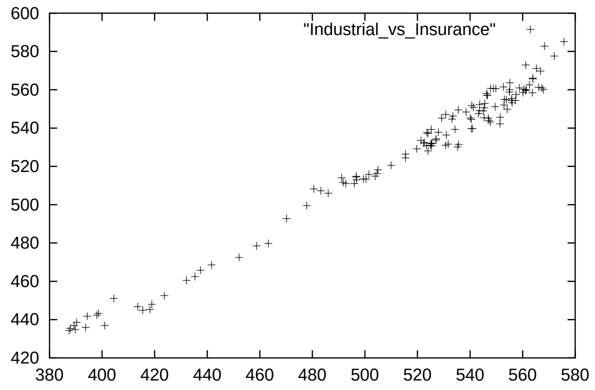

Figure 3: XY scatter plot of Industrial (X axis) vs Insurance (Y axis)

Piece-wise Approximation of Measurements

We here assume that the base signal $\vec{X}$ is given to us. We will approximate the latest $N \times M$ measurements in $\vec{Y}_1, \ldots, \vec{Y}_N$ using $B \geq 4 \times N$ values (i.e. using at least four values per time series). We later describe how to construct the base signal.

To simplify notation, we model the collected data as a single series $\vec{Y}$ that is simply the concatenation of the N series $\vec{Y}_i$. Our technique relies on breaking $\vec{Y}$ into $B/4$ intervals and "mapping" each one to an interval of the base signal of equal length.[4] The algorithm works recursively. It starts with a single interval for each row of the collected data. In each iteration, the interval with the largest error in the approximation is selected and divided in two halves, until the "budget" of $B/4$ intervals is exhausted. An interval I is a data structure with six entries:

- $start$, $length$: these define the scope of the interval; i.e. I represents values of $Y[i]$, with i in $[start, start + length)$.
- $shift$: it defines the part of the base signal that is used to approximate the values of I; the interval I is mapped to segment $[shift, shift + length)$ in $\vec{X}$.
- a, b, err: the first two are the regression parameters, while err is the sum squared error (*sse*) of the approximation.

Subroutine `Regression()` shown in Algorithm 1 is at the core of our method. This function pairs a segment of the base signal between values $[start\_x, start\_x + length)$ with values of $\vec{Y}$ between $[start\_y, start\_y + length)$, as in Figure 3, and computes the regression parameters a, b as well as the (sse) error of the approximation $\vec{Y'} = a\vec{X} + b$ in this range. Each value $Y[i]$ with index i in $[start\_y, start\_y + length)$ is approximated as $aX[start\_x + i - start\_y] + b$.

It should be noted that the `Regression()` subroutine calculates the optimal a, b values that minimize the sum squared error of the approximation. If the desired error metric is different, then the formulas need to be appropriately modified. In [9] we present the necessary modifications for two interesting optimization problems: minimizing the sum squared relative error, and minimizing the maximum absolute error of the approximation. The

[4]This mapping requires four values per interval, thus the division by 4.

Algorithm 1 `Regression` Subroutine

```
Require: $\vec{X}$, $\vec{Y}$, $start\_x$, $start\_y$, $length$
1: {Compute Regression Parameters}
2: $sum\_x = \sum_{0 \leq i < length} X[i + start\_x]$
3: $sum\_y = \sum_{0 \leq i < length} Y[i + start\_y]$
4: $sum\_xy = \sum_{0 \leq i < length} X[i + start\_x]Y[i + start\_y]$
5: $sum\_x2 = \sum_{0 \leq i < length} X[i + start\_x]^2$
6: $a = \frac{length \times sum\_x\_y - sum\_x \times sum\_y}{length \times sum\_x2 - sum\_x \times sum\_x}$
7: $b = \frac{sum\_y - a \times sum\_x}{length}$
   {Compute sse of approximate signal $\vec{Y'} = a\vec{X} + b$ in range $[start\_y, start\_y + length)$}
8: $err = \sum_{i=0}^{length-1} (Y[i + start\_y] - (aX[i + start\_x] + b))^2$
9: return ($a$,$b$,$err$)
```

modified algorithms run in $O(length)$ time and require $O(1)$ and $O(length)$ space, respectively.

Subroutine `BestMap()` of Algorithm 2 looks for the best way to approximate an interval I. It shifts I over $\vec{X}$ and calculates the regression parameters and the approximation error for the $shift$ parameter that produces the smallest error. This algorithm contains two deviations from our previous discussion. First, it also considers approximating each interval I using standard linear regression, and uses a negative value for the $I.shift$ parameter to denote this. Second, it performs the shifting process over the base signal only for intervals with a maximum length of $2 \times W$, where W is a parameter that denotes the length of the intervals that constitute the base signal.[5] The last modification is performed both to reduce the time complexity of the algorithm to O($I.length + W \times M_{base}$), and because of the reduced likelihood that large intervals will be accurately mapped to multiple consecutive intervals of the base signal.

The core approximation algorithm `GetIntervals()` is given in Algorithm 3. The approximation obtained is returned as a list of $B/4$ intervals in i_list. This list is maintained sorted (priority queue) based on the sse of each interval. $\vec{X}$ is the current base signal. The complete algorithm runs in $O(NMlog(\frac{B}{N}) + B \times M_{base} \times W)$ time. The logarithmic factor in the above formula is produced because the size of the intervals in the algorithm is repeatedly halved.

[5]This will become more clear later in our discussion.

Algorithm 2 `BestMap` Subroutine

```
Require: $\vec{X}$, $\vec{Y}$, Interval $I$, $W$
 1: $I.shift = -1$
 2: Perform standard linear regression on $I$ and set the values of
    $I.a$, $I.b$ and $I.err$
 3: if $I.length \leq 2 \times W$ then
 4:     {Shift $I$ over $\vec{X}$ and find the segment for which the regression
        error is minimized}
 5:     for $shift$ in $0..length(\vec{X}) - I.length - 1$ do
 6:         (a,b,err) = Regression($\vec{X}$, $\vec{Y}$, $shift$, $I.start$, $I.length$)
 7:         if err is minimum error so far then
 8:             Update values of $I.a$, $I.b$, $I.err$ and $I.shift$
 9:         end if
10:     end for
11: end if
```

Algorithm 3 `GetIntervals` Algorithm

```
Require: $\vec{X}$, $\vec{Y_1}$,..., $\vec{Y_N}$, $B$, $W$
 1: i_list = ()
 2: $\vec{Y} = concat(\vec{Y_1}, \ldots, \vec{Y_N})$ {Virtual assignment}
 3: {Create an interval for each row $\vec{Y_i}$ ($M$ values each)}
 4: for $i$ in 1..$N$ do
 5:     ($I.start$, $I.length$) = ((i-1) $\times$ $M$, $M$)
 6:     BestMap($\vec{X}$, $\vec{Y}$, $I$, $W$)
 7:     $i\_list$.push($I$);
 8: end for
 9: $num\_intervals$ = $N$
10: while $num\_intervals$++ $< B$ / 4 do
11:     {$i\_list$ is sorted on decreasing order of $I.err$}
12:     $I$ = $i\_list$.pop()
13:     {Break I in 2 pieces}
14:     ($I_{left}.start$, $I_{left}.length$) = ($I.start$, $I.length/2$)
15:     BestMap($\vec{X}$, $\vec{Y}$, $I_{left}$, $W$)
16:     ($I_{right}.start$, $I_{right}.length$) =
            ($I.start$+$I.length/2$, $I.length/2$)
17:     BestMap($\vec{X}$, $\vec{Y}$, $I_{right}$, $W$)
18:     $i\_list$.push($I_{left}$)
19:     $i\_list$.push($I_{right}$)
20: end while
21: return $i\_list$
```

For each interval in i_list a record with four values ($I.start$, $I.shift$, $I.a$, $I.b$) is transmitted to the base station. The base station will sort the intervals based on $I.start$ and, thus, there is no need to transmit their length. It is interesting to note that the `GetIntervals()` algorithm decides dynamically how many intervals it will use to approximate each of the N rows of the collected data, allocating more intervals to signals that are harder to approximate accurately.

Selecting Data Features for Inclusion in the Base Signal

We focus on the time when the sensor's memory is filled with NxM values, as depicted in Figure 1. We assume that the buffer allocated to the base signal is of size M_{base}. This buffer is organized as a list of intervals (called *base intervals*) of the same length W. For simplicity, we assume that both M and M_{base} are multiples of W. We note here that in Algorithm 3 the base signal is presented as a series of M_{base} values, which is simply the concatenation of the base intervals in the buffers.

The `GetBase()` algorithm (Algorithm 4) is called during the initialization and update procedure of the base signal. The algorithm receives as inputs the N signals, each of size M, the size W of each base interval, and the maximum number of intervals $maxIns$ that can be inserted in our base signal, where

$$maxIns = \frac{\min\{M_{base}, TotalBand\}}{W}$$

Each input signal $\vec{Y_i}$ is broken into $\frac{M}{W}$ non-overlapping intervals of size W. This provides a "dictionary" of $\frac{N*M}{W}$ *candidate base intervals* (CBIs). The algorithm will choose $maxIns$ CBIs out of this dictionary to be inserted into a *candidate update base signal*. We will describe in subsection 4.3 how to determine how many of these CBIs will ultimately be inserted into the base signal.

Algorithm 4 `GetBase()` Algorithm

```
Require: $\vec{Y_1}, \ldots, \vec{Y_N}$, $W$, $M$, $maxIns$
 1: Create $K = \frac{N \times M}{W}$ CBIs of width $W$
 2: For each CBI $Cand_i$, set its benefit to 0
 3: Maintain unsorted list $Q$ with CBIs
 4: Maintain list base_list with selected stored intervals
 5: $LinearErr(Cand_j)$ is the error of approximating $Cand_j$ using standard linear regression
 6: for $i$ in 1..$K$ do
 7:     for $j$ in 1..$K$ do
 8:         {Calculate error of approximating the j-th CBI by using
            as base the i-th CBI}
 9:         error=Regression($Cand_i$,$Cand_j$,0,0,$W$)
10:         if $error \leq LinearErr(Cand_j)$ then
11:             $Cand_i.benefit$+=$LinearErr(Cand_j)$-$error$
12:         end if
13:     end for
14:     Q.insert($Cand_i$)
15: end for
16: for i in 1..$maxIns$ do
17:     $C$ = Q.popBestInterval()
18:     base_list.insert($C$)
19:     for j in 1..$|Q|$ do
20:         adjust(Q[j].benefit, $C$)
21:     end for
22: end for
23: return base_list
```

Each CBI $Cand_i$ can be used to approximate any other CBI $Cand_j$, which is in-fact part of some $\vec{Y_k}$, using regression. We consider such an approximation to be beneficial, only if the error of the approximation is smaller than the error of approximating $Cand_j$ using standard linear regression. In Algorithm 4 we denote the latter error as $LinearErr(Cand_j)$. The benefit of using $Cand_i$ to approximate $Cand_j$ is simply the reduction in error that we get compared to $LinearErr(Cand_j)$.

The CBIs are stored in an unordered list Q. At each step of the algorithm, the CBI in Q with the largest benefit is selected for inclusion in the candidate update base signal stored in base_list. After each selection, the benefits of the remaining CBIs in Q have to be properly updated. As we mentioned, the benefit of using $Cand_i$ to approximate $Cand_j$ is originally equal to the reduction in error that we get compared to $LinearErr(Cand_j)$. However, at an intermediate step of the algorithm, some CBIs have already been selected for inclusion in the candidate update base signal. By using these stored CBIs, many of the remaining CBIs can now be better approximated than by using standard linear regression. Thus, the benefit of using $Cand_i$ to approximate $Cand_j$ has to be adjusted to depict the reduction in error that we get when compared to the best approximation for $Cand_j$ that we have so far, by using the current candidate update base signal. Intuitively, this adjustment prohibits the inclusion in the base signal of CBIs that help approximate well similar parts of the data.

An example is presented in Figure 4. In this small example we consider just 3 CBIs, out of which we need to pick which two to select. In the left part of the figure, we present the benefits of each of the 3 CBIs. The first CBI has the largest total benefit, and is thus selected. In the right part of the figure, the adjusted benefits of the remaining CBIs are presented. Notice that now, the third CBI will be selected, even though initially it had a lower benefit than the second CBI.

In the `GetBase()` algorithm, for each of the $K = \frac{N \times M}{W}$ CBIs, we first estimate its benefit for approximating all the other CBIs. Each such approximation requires $O(W)$ time, thus resulting in a total complexity of $O(\frac{N^2M^2}{W})$. Then, for each of the $maxIns$ selected CBIs, detecting the one with the largest benefit requires $O(K)$ time (we do not sort the CBIs). After each selection, adjusting the benefits of the remaining CBIs requires time $O(K^2)$. Thus, the overall running time complexity of the algorithm is $O(\frac{N^2M^2}{W} + maxIns \times \frac{N^2M^2}{W^2})$, while its space requirements is $O(\frac{N^2M^2}{W^2})$.

For $n = N \times M$ being the size of the data, a value of $W = \sqrt{n}$ used by the SBR algorithm (described in the next subsection) results in a running time of O($n^{1.5}$) for GetBase() and space of O(n), since $maxIns \times W \le TotalBand \le n$. In case of severe memory constraints, we can easily modify the `GetBase()` algorithm to only store for each CBI the smallest error of approximating it using at each step the current base signal. The only modification will be to replace Lines 19-21 of the `GetBase()` algorithm with a double for-loop similar to the one of Lines 6-15, and alter the calculation of each CBI's benefit to take into account the error of the best approximation that we have for each CBI so far. This modified algorithm requires $O(\sqrt{n})$ space and has a running time of $O(maxIns \times n^{1.5})$.

4.3 The SBR Algorithm

We now present the *Self-Based Regression* (SBR) algorithm that performs the approximation of the data values. The algorithm receives as input the latest $n = N \times M$ data values, a bandwidth constraint $TotalBand$ (number of values to transmit, *including any base signal values*), the maximum size of the base signal M_{base} and the current base signal $\vec{X}$ of size $|\vec{X}| \le M_{base}$.[6] From these parameters the user/application has to provide only $TotalBand$ and M_{base}. The SBR algorithm must then make the following decisions:

1. Decide how many, and which base intervals to insert into the base signal. Recall that any such base interval has to be transmitted to the base station.

2. If the above procedure causes the size of the base signal to exceed M_{base}, then some base intervals need to be evicted from the base signal, in order to keep its maximum size at M_{base}.

3. Decide how to best approximate the data values given the updated base signal.

We here have to emphasize that it is not always desirable to insert a large number of base intervals into the base signal. Since any inserted base interval needs to be communicated to the base station, the larger the number of such intervals, the smaller the number of intervals that can be used to approximate the N signals by the `GetIntervals()` algorithm, since the overall bandwidth consumption is upper-bounded by the $TotalBand$ parameter.

[6]At the first transmission the current base signal will be empty.

Algorithm 5 SBR Algorithm

```
Require: $\vec{X}, \vec{Y_1}, \ldots, \vec{Y_N}$, $M$, $TotalBand$, $M_{base}$
 1: $maxIns = \frac{\min\{M_{base}, TotalBand\}}{W}$
 2: $W = \sqrt{N \times M}$
 3: $base\_list = GetBase(\vec{Y_1}, \ldots, \vec{Y_N}, W, M, maxIns)$
 4: {$Errors[i]$ is the approximation error after inserting the first $i$
    CBIs of base_list in the base signal}
 5: Initialize $Errors[i]$ = UNDEFINED $\forall i \in [0..maxIns)$
 6: $Ins$ = Search($\vec{X}, \vec{Y_1}, \ldots, \vec{Y_N}, W, M, TotalBand, base\_list,$
    $Errors, 0, maxIns$)
 7: Form $\vec{X}_{new}$ by appending the $Ins$ first intervals of the base_list
    to $\vec{X}$
 8: $B = TotalBand - Ins \times (W+1)$
 9: $GetIntervals(\vec{X}_{new}, \vec{Y_1}, \ldots, \vec{Y_N}, B, W)$
10: if $|\vec{X}_{new}| > M_{base}$ then
11:     Evict $Repl = \frac{|\vec{X}_{new}| - M_{base}}{W}$ intervals of $\vec{X}_{new}$ that also
        belonged to $\vec{X}$ using a LFU replacement policy
12:     Replace evicted intervals with the last $Repl$ intervals of
        $\vec{X}_{new}$
13: end if
14: $\vec{X} = \vec{X}_{new}$
15: Transmit the inserted base intervals, their offsets in the base
    signal and the regression intervals
```

Algorithm 6 `CalculateError` SubRoutine

```
Require: $\vec{X}, \vec{Y_1}, \ldots, \vec{Y_N}, B, W, Errors, pos$
 1: if Errors[pos] == UNDEFINED then
 2:     list' = $GetIntervals(\vec{X}, \vec{Y_1}, \ldots, \vec{Y_N}, B - pos \times W, W)$
 3:     Errors[pos] = sum of errors in list'
 4: end if
```

The SBR algorithm is presented in Algorithm 5. It initially calls the `GetBase()` subroutine to select a set of $maxIns$ CBIs, where $maxIns = \frac{\min\{M_{base}, TotalBand\}}{W}$. It then performs a binary search on this list, to determine the number of CBIs that will ultimately be inserted into the base signal. This search terminates when the algorithm determines a number of intervals Ins, such that the error of the approximation when inserting the first Ins intervals of the aforementioned list in the base signal is lower than inserting either the first $Ins - 1$ intervals, or the first $Ins + 1$ intervals into the base signal. This is achieved through the call to function `Search()` at Line 7, which is presented in Algorithm 7. The approximation of the N signals is then performed by using the concatenation of the previous base signal with these Ins intervals. After this step, if the size of the base signal now exceeds M_{base}, then enough base intervals of the old base signal are evicted from the base signal using a Least Frequently Used (LFU) replacement policy. Any newly inserted base interval will thus either occupy an empty position of the base signal, or replace another base interval. Each transmission includes exactly $TotalBand$ values:

1. The Ins newly inserted base intervals, and their position in the base signal in which they were ultimately inserted ($Ins \times (W+1)$ values in total).

CBI	Approximated CBI 1	2	3	Total Benefit
1	1	0.95	0.50	2.45
2	0.8	1	0.55	2.35
3	0.6	0.65	1	2.25

Initial Benefits of CBIs

CBI	Approximated CBI 2	3	Total Benefit
2	0.05	0.05	0.10
3	0	0.5	0.50

Adjusted Benefits of Non-Stored CBIs

Figure 4: Example of the GetBase() Algorithm

Algorithm 7 `Search` SubRoutine

```
Require: $\vec{X}, \vec{Y_1}, \ldots, \vec{Y_N}, W, B, base\_list, Errors, start, end$
 1: if end == start then
 2:     return start
 3: end if
 4: middle = (start + end) / 2
 5: $CalculateError(\vec{X}, \vec{Y_1}, \ldots, \vec{Y_N}, B, W, middle)$
 6: $CalculateError(\vec{X}, \vec{Y_1}, \ldots, \vec{Y_N}, B, W, start)$
 7: if $Errors[middle] > Errors[start]$ then
 8:     $CalculateError(\vec{X}, \vec{Y_1}, \ldots, \vec{Y_N}, B, W, end)$
 9:     if $Errors[end] > Errors[start]$ then
10:         return Search($\vec{X}, \vec{Y_1}, \ldots, \vec{Y_N}, W, M, B, base\_list,$
            $Errors, start, middle$)
11:     else
12:         return Search($\vec{X}, \vec{Y_1}, \ldots, \vec{Y_N}, W, M, B, base\_list,$
            $Errors, middle, end$)
13:     end if
14: else
15:     $CalculateError(\vec{X}, \vec{Y_1}, \ldots, \vec{Y_N}, B, W, middle + 1)$
16:     if $Errors[middle + 1] < Errors[middle]$ then
17:         return Search($\vec{X}, \vec{Y_1}, \ldots, \vec{Y_N}, W, M, B, base\_list,$
            $Errors, middle + 1, end$)
18:     else
19:         return Search($\vec{X}, \vec{Y_1}, \ldots, \vec{Y_N}, W, M, B, base\_list,$
            $Errors, start, middle$)
20:     end if
21: end if
```

2. $\frac{TotalBand - Ins \times (W+1)}{4}$ intervals of four values each (start, shift plus the two regression parameters).

The running time complexity of the SBR algorithm is $O(n^{1.5} + (nlog(\frac{TotalBand}{N}) + TotalBand \times \sqrt{n} \times M_{base}) \times \log(maxIns))$, where $maxIns = \frac{\min\{M_{base}, TotalBand\}}{\sqrt{n}}$. Thus, the entire algorithm has a modest $O(n^{1.5})$ dependency on the data size, while its running time scales linearly with the size of the transmitted data $TotalBand$ and the (maximum) size of the base signal M_{base}.

4.4 Understanding the Complexity of SBR

We note that the SBR algorithm is only executed periodically, thus, its running-time complexity has to be evaluated with respect to the size of the data and the frequency that the algorithm is being executed in a real application. Using our implementation of the algorithm on a 300MHz processor, it takes about 30 seconds to process n=20,480 data values (10 time series of 2048 values each) for an 10% compression ratio (see Section 5). Even if one measurement is being taken every second, the above running time corresponds to measurements collected over 34 minutes. This means that the time required by the SBR algorithm for approximating the data is just the 1/68 of the time it took the sensor to collect it, thus making it easy for the SBR algorithm to run in parallel with the collection process. If a shorter running time of SBR is desired, one can simply either execute the algorithm with a smaller value of n,[7] or decide not to update the base signal, which is by far the most expensive part of the SBR algorithm, in each transmission. The latter method is not expected to affect the quality of the approximation significantly, since our experiments have demonstrated that after the first transmissions few base intervals are inserted in the base signal, because the current base signal is already of good quality at that point. Notice that if the base signal is not updated, then only the `GetIntervals()` algorithm is invoked, resulting in an overall running time complexity of: $O(nlog(\frac{TotalBand}{N}) + TotalBand \times \sqrt{n} \times M_{base})$. Notice that in this case the algorithm exhibits a linear dependency on the size of the processed data n.

4.5 Providing Strict Error Bounds

The SBR algorithm, as presented above, seeks to minimize a user-defined error metric (ex: sum squared error) given a target bandwidth constraint. An interesting extension is when the application requires strict error bounds. The typical goal in such cases is to minimize the maximum error of the approximation and provide this maximum error along with the approximate signal. In this case, a `Regression()` subroutine for minimizing the maximum error of the approximation (see [9]) should be used.

Another interesting case occurs when the application provides a target size $TargetBand$ and an error target with which it will be satisfied. In this case, the application will be satisfied with any approximation of size less or equal to $TargetBand$ that satisfies the error target (if such an approximation exists). In this case the recursive procedure of the `GetIntervals()` algorithm may be stopped if the error target is achieved before the size of the transmitted data reaches $TargetBand$.

5. EXPERIMENTS

In this section, we provide a thorough analysis of our techniques. In subsection 5.1 we compare the SBR algorithm against standard approximation techniques (Wavelets, DCT, Histograms). In subsection 5.2 we compare the `GetBase()` algorithm against alternative base-signal constructions, while in subsection 5.3 we present an analysis of the SBR algorithm. For these experiments we used the following real datasets:

1. `Phone Call Data:` Includes the number of long distance calls originating from 15 states (AZ, CA, CO, CT, FL, GA, IL, IN, MD, MN, MO, NJ, NY, TX, WA). For each state we provide the number of calls per minute for a period of 19 days (data provided by AT&T Labs).

2. `Weather Data:` Includes the air temperature, dewpoint temperature, wind speed, wind peak, solar irradiance and relative humidity weather measurements for the station in the

[7]For example, when reducing the value of n to 10,240 data values, the corresponding running time of SBR is just 14.4 seconds.

university of Washington, and for year 2002 (http://www-k12.atmos.washington.edu/k12/grayskies).

3. `Stock Data:` Includes information on all trades performed in a minute basis over April 3 and April 4 of year 2000. The approximated measure in our experiments is the trade value of the stock.

5.1 Comparison to Alternative Techniques

For this experiment we used all three datasets described above. From the *Stock* data, we extracted the trade values of the following ten (N=10) stocks: Microsoft, Oracle, Intel, Dell, Yahoo, Nokia, Cisco, WorldCom, Ariba and Legato Systems. For each stock we created a random sample of 20,480 of its trade values, and then split each sample in ten files of 2,048 values each. The first of these ten files of each stock was used for the initial creation of our base signal, while the remaining files were used to simulate nine update operations. For the *Weather* dataset, we selected the first 40,960 records and then split the data measurements of each signal into ten files of 4,096 values each. For the *Phone Call* dataset, the aggregates for each state (N=15) were broken into ten files of 2,560 values each.

In our experiments we compared the accuracy of SBR against the approximations obtained by using the *Wavelet* decomposition [3, 29], equi-depth Histograms [25] and the DCT. The Fourier transform was also considered, but produced consistently larger errors than DCT and is thus omitted. For a fair comparison we set the space used by all methods to the exact same amount.

For all methods we considered both treating each bunch of updates as a group of N series $\vec{Y_i}$ each of length M and, alternatively, concatenating the signals into a single series Y of length $N \times M$. For Wavelets, we found out that this produced in most cases significantly more accurate results than by dividing the space equally among the N signals (by a factor of 5 in many cases) because some signals needed more wavelet coefficients than others to be approximated well. For Wavelets, we also considered a 2-dimensional decomposition of the $N \times M$ values, which produced worse results than the 1-dimensional decomposition. We here present the best results achieved by each method.

5.1.1 Varying the Compression Ratio

We varied the compression ratio (size of the transmitted data $TotalBand$ over the data size n) from 5% to 30%. In this experiment we set M_{base} to 2,048 values for the *Phone Call* and the *Stocks* datasets and to 3,456 values for the *Weather* dataset. In Tables 2 and 3 we present the results.

In all datasets SBR produces significantly more accurate results than the other approximations. The difference is larger for the *Phone Call* dataset which contained the largest values. As the size of transmitted data increases, the error in our method decreases more sharply, and is up to 4.4 times smaller than the error of Wavelets. The DCT and the Histogram approximations produced much larger errors is most cases.

We repeated the experiment for the *Phone Call* dataset, computing this time the sum-squared relative error. The results are also shown in Table 3. The modified `Regression()` algorithm is presented in [9]. Depending on the compression ratio, our method was up to 49 times better than Wavelets, 9.8 times better than DCT and 258 times better than Histograms. We note here that for this comparison we used Haar Wavelets that are optimal only under the sum-squared-error. The work of [12] describes algorithms for minimizing, among other metrics, the relative error of a Wavelet-based approximation. Except for cases of very skewed datasets, these algorithms reduce the mean relative error up to 3 times over regular Wavelets. These improvements were seen for very coarse approximations (i.e. for a compression ratio of 5% or less) where our method already has an advantage of 42-1 over regular Wavelets. For more space, these techniques are a lot closer to regular Wavelets.

5.1.2 Mixing The Datasets

The SBR algorithms exploits intrinsic correlations between the signals. We now explore its behavior when these correlations are reduced. At this experiment we tried mixing data from the three datasets. We created a dataset that contains phone call data from three states (AZ, CA and FL), three types of meteorological measurements (air temperature, pressure and solar irradiance), and data from three stocks (Microsoft, Intel and Oracle). For each of these data series we created ten files of 2,048 values each. We then varied the compression ratio of all algorithms from 5% to 30% and set M_{base} to 2,048 values. In Table 4 we present the average sum squared and total sum squared relative errors for all methods. The improvements of the SBR algorithm were even larger in this case. The SBR algorithm produced up to 27 times smaller average sum squared errors than the closest competitor, while the improvement reached up to 1,034 times for the total sum squared relative error.

While the results may seem surprising because the correlation between the datasets was decreased, they are not counter-intuitive. All the approximation methods exploit some form of correlation or redundancy to reduce the footprint of the data. Table 4 simply shows that SBR is more robust, than Wavelets and Histograms for example, when such correlations are reduced. The design of the algorithm allows it to find correlations even in such cases, between intervals from different signals and different time periods. The algorithm also has a fall-back plan of using plain regression when such correlations are not strong. In such cases, fewer space is allocated for the base-signal and most of the transmitted values are used for approximating more, smaller intervals. In the next set of experiments we will demonstrate that SBR, because of the judicious allocation of space between the base signal and the approximated intervals, outperforms linear regression even when we do not use its fall-back plan to (potentially use) linear regression for the approximation of some intervals.

5.2 Alternative Base Signal Constructions

In the Appendix we present two alternative algorithms to `GetBase()`. The first, denoted as `GetBaseSVD()`, is based on the Singular Value Decomposition. The second algorithm, denoted as `GetBaseDCT()`, uses the basis of the Discrete Cosine Transform (DCT), which is a collection of cosine functions. Finally, a third alternative for SBR is to do standard linear regression without using a specially constructed base signal. For the later case, no bandwidth is lost for sending base signal values and we do not need the $I.shift$ pointer. Thus, we can send exactly $TotalBand$/3 intervals for a bandwidth limit $TotalBand$. Similarly, the DCT base consists of cosine functions and its values are constructed on the fly and are thus neither stored in memory, nor are they transmitted to the base station.

In Table 5 we compare the approximations obtained by using the base signals computed in algorithm `GetBase()` with the base signal from the alternative constructions. We need to emphasize here that for this experiment we modified the `BestMap()` function not to use linear regression as an alternative to using the base signal (so that the differences among `GetBase()`, `GetBaseSVD()`, `GetBaseDCT()` and linear regression are not diffused). Using the BestMap() function as presented in Section 4.2 would further

Compression Ratio	Weather Data				Stock Data			
	SBR	Wavelets	DCT	Histograms	SBR	Wavelets	DCT	Histograms
5%	1.160	2.187	35.835	27.692	0.089	0.123	0.232	0.283
10%	0.403	0.824	20.169	11.294	0.033	0.056	0.208	0.233
15%	0.209	0.514	14.328	5.432	0.017	0.034	0.192	0.214
20%	0.118	0.356	10.774	3.009	0.009	0.022	0.179	0.199
25%	0.069	0.258	8.975	1.507	0.006	0.015	0.166	0.182
30%	0.043	0.191	6.526	0.995	0.003	0.011	0.153	0.169

Table 2: Average SSE Error Varying the Compression Ratio for Weather and Stock Datasets

Compression Ratio	Average SSE Error				Total Sum Squared Relative Error			
	SBR	Wavelets	DCT	Histograms	SBR	Wavelets	DCT	Histograms
5%	9,631	29,938	15,714	165,241	922	38,477	9,019	139,528
10%	5,071	12,349	10,173	45,610	503	19,186	3,002	62,337
15%	3,192	7,998	6,767	23,311	325	12,885	1,400	36,812
20%	2,170	5,821	5,661	15,581	222	10,954	1,192	34,820
25%	1,527	4,468	4,791	11,340	158	6,915	823	33,237
30%	1,091	3,537	4,157	8,689	116	3,865	721	30,010

Table 3: Errors Varying the Compression Ratio for Phone Call Dataset

Dataset	Error over GetBase()		
	GetBaseSVD()	Linear Regression	GetBaseDCT()
Weather	10.55	4.47	6.44
Phone	1.13	1.32	1.19
Stock	2.08	2.77	2.99

Table 5: Comparison to Alternative Base Signals

Dataset	Transmission									
	1	2	3	4	5	6	7	8	9	10
Weather	6	6	1	0	3	0	2	3	0	1
Phone	3	6	0	1	0	0	2	0	0	1
Stock	3	0	0	2	1	0	0	0	2	0

Table 6: Number of Inserted Base Intervals per Transmission

improve the results of our method. The compression ratio was set to 10%. We notice that `GetBase()` performs a lot better in the *Weather* dataset, up to 10 times better than the alternative algorithms. For the *Phone Call* and the *Stock* data the differences are smaller but still significant.

5.3 Analysis of SBR

We now analyze several characteristics of the SBR algorithm, including its running time, the number of base intervals it selects for inclusion in the base signal and the quality of its decisions.

In Figure 5 we plot the average time of each transmission operation for the Stock dataset, when the size of the transmitted data is varied from 5% to 30% of the data size, the size of n varies from 5,120 to 20,480,[8] and the size of the base signal is 1,024. Since we have not yet ported our code to the StrongARM platform, we executed this experiment on a Irix machine using a 300MHz processor. As expected (see Section 4.3) the running time scales linearly with the size of the transmitted data. Notice that SBR is significantly faster when greater reduction is obtained. For many practical applications, we expect to use a compression ratio of 10% (or even less), where running time varies from 5.6 to 30 seconds depending on the value of n.

The SBR algorithm dynamically decides the number of base signal values to use for an upper bound M_{base}. We now compare SBR against a straight-forward implementation that populates all the available space for the base signal. In Figure 6 we plot the error of only the initial transmission as the size of the base signal is varied, manually, from 1 to 30 intervals for the *Phone, Stock* and *Weather* datasets. For this initial transmission we populated the entire space of the base signal using the `GetBase()` algorithm. For each dataset we also show the selection that the SBR algorithm made, when deciding how many base intervals to populate. For presentation purposes the errors for each dataset have been divided by the error of the approximation when using just one interval. We set the size of each stock, phone and weather data file to 3072, 2048 and 5120 values respectively, in order for all datasets to have exactly the same size, and the $TotalBand$ value to 5012, which results to a compression ratio ($TotalBand/n$) of about 16%.

The fixed value of the compression ratio implies that an increase in the size of the base signal results in a decrease in the number of intervals used to approximate the data values in order to keep the total space constant. After some point, the benefit of storing more intervals for the base signal is outweighted by the increase in the error that we get due to the reduced number of intervals used for the approximation. It is interesting to see that the optimal case occurs for a base size of between 7 (for the *Weather* dataset) and 9 base intervals (for the *Stock* dataset), which correspond to just 2.9% to 3.75% of the data size at the first transmission. The SBR algorithm made the optimal choice for the *Phone* and *Weather* datasets and produced a near-optimal solution for the *Stock* dataset (it selected to insert 6 base intervals, instead of 9). We remind that the M_{base} base signal values need to be kept in the memory of the sensor in order to perform the approximation. Our results suggest that a very small fraction of memory needs to be sacrificed for these values.

For the same data setup, we report in Table 6 the number of in-

[8]By varying the value of M. We always used data from 10 stocks.

Compression Ratio	Average SSE Error				Total Sum Squared Relative Error			
	SBR	Wavelets	DCT	Histograms	SBR	Wavelets	DCT	Histograms
5%	2,900	8,094	12,677	199,150	113	20,974	29,625	182,027
10%	918	3,020	7,146	46,805	37	11,054	8,653	43,701
15%	364	1,582	4,757	23,711	17	5,481	4,825	26,068
20%	139	894	3,814	14,157	9	5,310	3,339	14,780
25%	46	516	3,120	10,486	5	5,172	6,115	11,118
30%	11	297	2,680	6,894	3	5,109	1,579	9,591

Table 4: Errors Varying the Compression Ratio for the Mixed Dataset

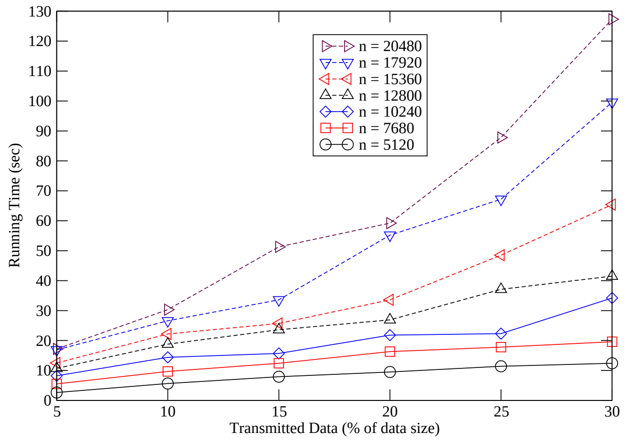

Figure 5: Average Running Time vs TotalBand

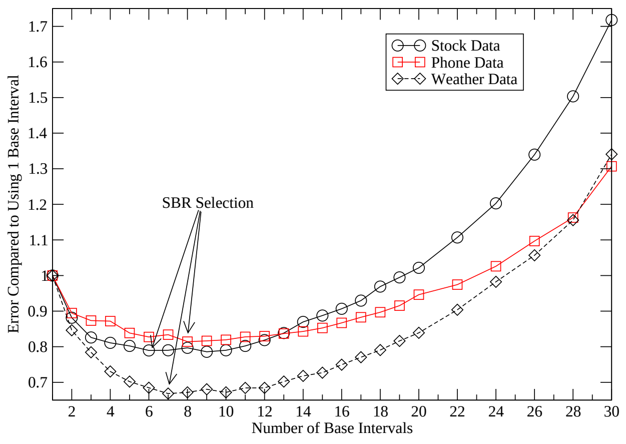

Figure 6: SSE error vs base signal size

serted base intervals during the 10 transmissions. As we can see, most base intervals are inserted during the first two transmissions. We notice that there are many transmissions during which no new base intervals are inserted, and that the different datasets seem to contain a widely different number of features, with the Weather dataset containing the most features, and the Stock dataset containing the fewest.

The small number of intervals inserted in the base signal after the initial transmissions allows us to consider executing the SBR algorithm only periodically, or when the quality of the approximation degrades, in the case of constrained environments. For the other transmissions, the approximation may be performed by simply using the significantly faster `GetIntervals()` algorithm.

6. CONCLUSIONS

We presented a new data compression technique, designed for historical data collected in sensor networks, which however can also be applied in compressing multiple time series in general. Our method splits the recorded series into intervals of variable length and encodes each of them using an artificially constructed *base signal*. The values of the base signal are extracted from the real measurements and maintained dynamically as data changes. Our method easily adapts to different error metrics by simply changing the Regression subroutine used. It can also be modified to provide strict error bounds or a combination of error and space bounds.

In our experiments we used real datasets from a variety of fields (weather, stock and phone call data). Using the sum-squared error and the sum-squared relative error of the approximation, our method significantly outperformed in accuracy approximations obtained by using Wavelets, DCT and Histograms.

A key to our method is the use of the base signal for encoding piece-wise linear correlations among the data values. We emphasize here that our method does not only apply to linear datasets; in fact none of the data we used are linear in nature. Linearity is exploited when encoding the correlations of the data values and the base signal. An interesting question is to what extent non-linear encodings over the base signal values would benefit the approximations obtained without sacrificing complexity. We plan to investigate this path in the future.

7. REFERENCES

[1] N. Ahmed, T. Natarakan, and K.R. Rao. Discrete cosine transform. In *IEEE Trans. on Computers, C-23*, 1974.

[2] A. Cerpa and D. Estrin. ASCENT: Adaptive Self-Configuring sEnsor Network Topologies. In *INFOCOM*, 2002.

[3] K. Chakrabarti, M. Garofalakis, R. Rastogi, and K. Shim. Approximate Query Processing Using Wavelets. In *Proceedings of the 26th VLDB Conference*, 2000.

[4] J. Chen, D.J. Dewitt, F. Tian, and Y. Wang. NiagaraCQ: A Scalable Continuous Query System for Internet Databases. In *Proceedings of ACM SIGMOD Conference*, 2000.

[5] Y. Chen, G. Dong, J. Han, B. W. Wah, and J. Wang. Multi-Dimensional Regression Analysis of Time-Series Data Streams. In *Proceedings of VLDB*, 2002.

[6] R. Cheng, D. V. Kalashnikov, and S. Prabhakar. Evaluating Probabilistic Queries over Imprecise Data. In *Proceedings of ACM SIGMOD Conference*, 2003.

[7] M. Cherniack, M. J. Franklin, and S. B. Zdonik. Data Management for Pervasive Computing. In *Proceedings of VLDB*, 2001.

[8] A. Deligiannakis, Y. Kotidis, and N. Roussopoulos. Hierarchical in-Network Data Aggregation with Quality Guarantees. In *Proceedings of EDBT*, 2004.

[9] A. Deligiannakis, Y. Kotidis, and N. Roussopoulos. Data Reduction Techniques for Sensor Networks. Technical report, University of Maryland, July 2003.

[10] D. Estrin, R. Govindan, J. Heidermann, and S. Kumar. Next Century Challenges: Scalable Coordination in Sensor Networks. In *MobiCOM*, 1999.

[11] D. Ganesan, D. Estrin, and J. Heidermann. DIMENSIONS: Why do we need a new Data Handling architecture for Sensor Networks? In *HotNets-I*, 2002.

[12] M. Garofalakis and P. B. Gibbons. Wavelet Synopses with Error Guarantees. In *Proceedings of ACM SIGMOD Conference*, 2002.

[13] J. Heidermann, F. Silva, C. Intanagonwiwat, R. Govindanand D. Estrin, and D. Ganesan. Building Efficient Wireless Sensor Networks with Low-Level Naming. In *SOSP*, 2001.

[14] J.M. Hellerstein, M.J. Franklin, S. Chandrasekaran, A. Descpande, K.Hildrum, S. Madden, V. Raman, and M.A. Shah. Adaptive Query Processing: Technology in Evolution. In *IEEE DE Bulletin 23(2)*, 2000.

[15] C. Intanagonwiwat, D. Estrin, R. Govindan, and J. Heidermann. Impact of Network Density on Data Aggregation in Wireless Sensor Networks. In *ICDCS*, 2002.

[16] S. Khanna and W. C. Tan. On Computing Functions with Uncertainty. In *Proceedings of ACM PODS Conference*, 2001.

[17] J. Lee, D. Kim, and C. Chung. Multi-dimensional Selectivity Estimation Using Compressed Histogram Information. In *Proceedings of ACM SIGMOD Conference*, 1999.

[18] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. TAG: A Tiny Aggregation Service for ad hoc Sensor Networks. In *OSDI Conf.*, 2002.

[19] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. The Design of an Acquisitional Query processor for Sensor Networks. In *Proceedings of ACM SIGMOD Conference*, 2003.

[20] Y. Matias, J. S. Vitter, and M. Wang. Wavelet-Based Histograms for Selectivity Estimation. In *Proceedings of ACM SIGMOD Conference*, 1998.

[21] R. Motwani, J. Widom, A. Arasu, B. Babcock, S. Babu, M. Datar, G. Manku, C. Olston, J. Rosenstein, and R. Varma. Query Processing, Resource Management, and Approximation in a Data Stream Management System. In *Proceedings of CIDR*, 2003.

[22] C. Olston, J. Jiang, and J. Widom. Adaptive Filters for Continuous Queries over Distributed Data Streams. In *Proceedings of ACM SIGMOD Conference*, 2003.

[23] C. Olston and J. Widom. Offering a Precision-Performance Tradeoff for Aggregation Queries over Replicated Data. In *Proceedings of VLDB*, 2000.

[24] V. Poosala and Y. E. Ioannidis. Selectivity Estimation Without the Attribute Value Independence Assumption. In *Proceedings of the 23th VLDB Conference*, 1997.

[25] V. Poosala, Y. E. Ioannidis, P. J. Haas, and E. J. Shekita. Improved Histograms for Selectivity Estimation of Range Predicates. In *Proceedings of ACM SIGMOD Conference*, 1996.

[26] L. Qiao, D. Agrawal, and A.E. Abbadi. RHist: Adaptive Summarization over Continuous Data Streams. In *Proceedings of CIKM*, 2002.

[27] E. Shih, S.-H. Cho, and N. Ickes et al. Physical Layer Driven Protocol and Algorithm Design for Energy-Efficient Wireless Sensor Networks. In *Proceedings of MOBICOM*, 2001.

[28] S. D. Viglas and J. F. Naughton. Rate-based Query Optimization for Streaming Information Sources. In *Proceedings of ACM SIGMOD Conference*, 2002.

[29] J.S Vitter and M. Wang. Approximate Computation of Multidimensional Aggregates of Sparse Data Using Wavelets. In *Proceedings of ACM SIGMOD Conference*, 1999.

[30] Y. Yao and J. Gehrke. The Cougar Approach to In-Network Query Processing in Sensor Networks. *SIGMOD Record*, 31(3):9–18, 2002.

[31] S. B. Zdonik, M. Stonebraker, M. Cherniack, U. Cetintemel, M. Balazinska, and H. Balakrishnan. The Aurora and Medusa Projects. *IEEE DE Bulletin*, 2003.

Appendix

Construction Using SVD

It can be proven that any real $N \times n$ matrix can be written as: $R = U \times \Lambda \times V^t$, where U is a column-orthonormal $N \times r$ matrix, r is the *rank* of matrix R, Λ is a diagonal $r \times r$ matrix of the eigenvalues λ_i of R and V is a column-orthonormal $n \times r$ matrix. The columns of V are the eigenvectors of matrix $R^t \times R$. Similarly, the eigenvalues of $R^t \times R$ are the squares of λ_is: $R^t \times R = V \times \Lambda^2 \times V^t$

For R=A (our collected measurements), $R^t \times R$ captures the similarities among the columns of A (each collected sample). SVD can be used for approximating $R^t \times R$ by keeping the first few eigenvectors (columns of matrix V). Informally, each eigenvector captures linear trends among the rows of A (the $\vec{Y_i}$s). We here propose the use of SVD as a competitor to the `GetBase()` algorithm for generating a base signal from the data. We sketch the new algorithm (`GetBaseSVD()`) below.

1. For each row of A, list all non-overlapping intervals of length W. This gives us $\frac{M}{W}$ intervals per row and $K = \frac{N \times M}{W}$ intervals overall.
2. Build a $K \times W$ matrix R whose rows are the intervals of the previous step.
3. Compute the SVD of $R = U \times \Lambda \times V^t$. Return the first $Store$ columns of V.

By definition, V is an $r \times W$ matrix (r=$rank(R)$) of the eigenvectors of $R^t \times R$. The eigenvectors are ordered from left to right in V. The first column of V contains the eigenvector (of length W) that corresponds to the largest eigenvalue of $R^t \times R$. The algorithm returns the top-$maxIns$ eigenvectors of total size $maxIns \times W$. These constitute the base signal from `GetBaseSVD()`.

Construction Using DCT

Assuming that we are to use base intervals of length W, we enumerate all frequencies f such that $0 \leq f \leq W$. For each frequency f, we define a base interval with values $cos(\frac{(2i+1)\pi}{2W} f)$, where $0 \leq i < W$. We call this algorithm `GetBaseDCT()`. We notice that we do not need to store these intervals implicitly, as they can be computed on the fly.