Advance Access publication on January 20, 2010
doi:10.1093/comjnl/bxp128

Adaptive Linear Filtering Compression on Realtime Sensor Networks

AARON KIELY[1], MINGSEN XU[2,*], WEN-ZHAN SONG[2], RENJIE HUANG[2] AND BEHROOZ SHIRAZI[2]

[1]*Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA, USA*
[2]*Sensorweb Research Laboratory, Washington State University, Vancouver, WA, USA*
[*]*Corresponding author: mingsen_xu@wsu.edu*

We present a lightweight lossless compression algorithm for realtime sensor networks. Our proposed adaptive linear filtering compression (ALFC) algorithm performs predictive compression using adaptive linear filtering to predict sample values followed by entropy coding of prediction residuals, encoding a variable number of samples into fixed-length packets. Adaptive prediction eliminates the need to determine prediction coefficients a priori and, more importantly, allows compression to dynamically adjust to a changing source. The algorithm requires only integer arithmetic operations and thus is compatible with sensor platforms that do not support floating-point operations. Significant robustness to packets losses is provided by including small but sufficient overhead data to allow each packet to be independently decoded. Real-world evaluations on seismic data from a wireless sensor network testbed show that ALFC provides more effective compression and uses less resources than an alternative recent work of lossless compression, S-LZW. Experiments in a multi-hop sensor network also show that ALFC can significantly improve raw data throughput and energy efficiency. We also implement the algorithm in our real sensor network, and show that our linear prediction based compression algorithm significantly improves data reliability and network efficiency.

Keywords: adaptive linear filtering compression (ALFC); lossless compression; wireless sensor networks

Received 15 August 2009; revised 20 December 2009
Handling editor: Yu-Chee Tseng

1. INTRODUCTION

Wireless sensor networks have attracted significant interest from the research community for a broad range of applications [1–3]. Several recent applications involve high data rate signals, such as monitoring industrial plants [4], volcano hazards [5, 6] and civil structures such as buildings or bridges [7, 8]. For these high rate applications, collecting high-fidelity data subject to the limited radio bandwidth available to sensor nodes presents a key challenge. In addition to the limited physical bit rate of the radios used in low power platforms, radio links may experience frequent packet losses due to congestion, interference, and multipath effects. These problems are exacerbated over multi-hop routing paths. There is a fundamental tradeoff between the network size and the total quantity of raw signals that can be collected.

Data compression is an important tool to maximize data return over unreliable and low rate radio links. For example, it has been shown in [9] that in some applications a compression computation yielding only a single byte of data reduction may be worth spending between roughly 4000 (Chipcon CC2420) and 2 million (MaxStream XTend) cycles of computation.

Sensor networks have some significant limitations such as limits on computational capability, memory and packet size, as well as high packet loss rates and small energy supplies. Those constraints present significant challenges in designing compression schemes for sensor networks.

At present, sensor nodes (e.g. MICAz, TelosB) usually have relatively modest computational capabilities and do not support floating-point arithmetic, thus requiring a low-complexity compression approach that can operate in real time without the use of floating-point operations. Even on platforms that support floating-point arithmetic (e.g. iMote2), it may be desirable to avoid such calculations to reduce power consumption.

Simultaneous transmission from multiple nodes through multi-hop relays within a network can lead to congestion, collision and high packet losses, demanding a compression scheme that allows packets to be decompressed even when preceding packets have been lost. Lossless compression is frequently

demanded by science users to preserve fidelity of critical data (e.g. earthquake data). But packet losses may be unavoidable in realtime high-fidelity sensor networks due to limited bandwidth or node mobility. Motivated by this, some sensor network applications are designed to provide reliable delivery of critical data and best-effort delivery of non-critical data.

Many sensor platforms employ hardware that uses the IEEE 802.15.4 standard and limits the maximum packet size to 128 bytes. The TinyOS operating system sets the default data payload length to 29 bytes and many systems recommend a packet size of no more than 90 bytes. Providing independently decodable packets under such limitations requires that increased attention be paid to overhead information included with each packet.

In this paper, we present a lightweight compression scheme for sensor networks, called adaptive linear filtering compression (ALFC). Our intended application is compression of seismic data, but this method may work well for other data types.

Our method relies on adaptive prediction, which eliminates the need to determine prediction coefficients a priori and, more importantly, allows the compressor to dynamically adjust to a changing source. This is particularly important for seismic data because the source behavior can vary dramatically depending on seismic activity. Predicted sample values are used to losslessly encode source samples using a variable-length coding scheme. We map each sample value to a non-negative integer and then encode the resulting sequence using Golomb codes. This general strategy is used in the Rice entropy coding algorithm [10, 11] and the LOCO-I image compressor [12], among myriad other applications. We also alter the prediction approach for the first few samples in the packet so that it does not rely on sample values in preceding packets.

To eliminate floating-point operations, we make use of rational approximations to real-valued quantities used in linear prediction.

We have implemented and evaluated the proposed algorithm in a multi-hop sensor network testbed with a seismic data feed. The testbed experiments show that our ALFC algorithm can significantly increase the quantity of received data and energy efficiency, and provide better performance compared with a recent alternative approach [9] in the literature. Partial results of this paper were published in [13, 14]. Our new contributions include Section 4.3, which evaluates the performance of ALFC in a recently deployed sensor network on Mount St. Helens volcano. The results show that ALFC performs effectively not only on the lab testbed, but also in the active volcano field, where the seismic data is highly dynamic.

The rest of the paper is organized as follows. In Section 2, we review related work on predictive lossless seismic data compression and data compression in sensor networks. We introduce our proposed linear prediction, packetization and entropy coding algorithm in Section 3, and present our evaluation experiments in Section 4. Finally, we conclude our paper and present potential future research in Section 5.

2. RELATED WORK

2.1. Predictive lossless seismic data compression

Lossless predictive compression generally consists of a prediction stage, where each sample value is predicted based on past sample values, and a coding stage, where an entropy coding method is applied to losslessly encode prediction residuals (the difference between predicted and actual sample values). A variety of different approaches for the prediction and entropy coding stages have been used in the literature for lossless compression of seismic data.

The least complex predictors include simply using the previous sample value for prediction, as in the approaches described in [15], or using linear prediction with fixed coefficients determined a priori, as in [16]. Better performance can be obtained at the expense of higher complexity by selecting prediction coefficients separately optimized for each block of samples, as is done in [17, 18] for two different source models. More sophisticated prediction approaches in the literature include the use of a 20-stage adaptive recursive least squares lattice adaptive filtering approach [19], several repeated stages of adaptive linear prediction using gradient adaptive lattice filters on very large blocks of samples [20], and a source model designed to incorporate structural system information when external exogenous input information is available [21]. Mandyam *et al.* [22] make use of a somewhat more conventional adaptive linear filtering approach, as in our approach. Unlike our sensor network application, however, the work of Mandyam *et al.* [22] is applied to much larger blocks of data (each at least 14 000 data samples), and employs somewhat higher complexity prediction, requiring more arithmetic operations and making use of floating-point prediction coefficients.

For lossless encoding of prediction residuals, approaches include a method introduced in [23] based on arithmetic coding and used in [16, 18–22]. Slightly lower compression effectiveness is provided by the simpler bi-level coding approach described in [17]. The bi-level coding method has similar complexity to our approach, which relies on Golomb codes, but note that selecting the coding parameters to obtain the best performance from bi-level coding assumes knowledge of the standard deviation of the prediction residuals being encoded in the block. Even lower complexity *ad hoc* entropy coding approaches can be found in the compression methods described in [15].

The most significant differences between our approach and these related works are our design requirement that packets can be decompressed independently as a means of dealing with potentially high packet losses, and the constraints imposed by small packet size and limited computational capabilities. Higher computational capabilities would permit higher complexity compression approaches, e.g. performing a least-squares optimization to determine optimal prediction coefficients, optimizing the filter prediction order for each packet, using floating point coefficients, or using arithmetic

coding. The use of relatively short packets means that the problem of efficiently encoding overhead information (e.g. to indicate coding parameters or prediction coefficient information) becomes more significant than in typical seismic data compression scenarios. We also note that nearly all of the other approaches encode a fixed number of samples into variable-length encoded units, while our approach encodes a variable number of samples into fixed length packets.

2.2. Compression in sensor networks

The literature includes many papers on data compression for sensor networks, though many of these approaches have not been evaluated in real sensor network testbeds.

Much prior work focuses on exploiting high spatial correlation in data from fixed sensors in dense networks. In [24, 25], distributed source coding approaches are used to exploit data dependencies between nodes in a sensor network. The approach in [24] minimizes the amount of inter-node communication for compression using both a quantized source and correlated side information within each individual node. This is a good approach in principle, but the degree of correlation with the side information is essential to the performance of the algorithm and normally not well known in practice. In [25], an intelligent data gathering node determines the degree of correlation between data from different source nodes, and based on this information tells each data node the allowable amount of compression. Each data source node executes a very simple compression algorithm that allows correlation between sources to be exploited even when the node does not have access to data from neighboring nodes. Note that experiments in [25] were performed using light, temperature and humidity sensors, and it is unclear how well this compression approach would work in the presence of seismic events, when the nature of the data dependency between nodes might change dramatically and without warning. Yu *et al*. [26] investigate a tunable compression scheme in sensor networks, which exploits spatial correlation between sample data. It trades the computation cost against communication cost by tuning the compression complexity. The authors propose a flow model and the approaches for deciding the optimal flow for both shortest path tree (SPT) and minimal Steiner tree (MST), based on different data correlations and relative computation costs.

Several methods have been proposed using wavelets and their variants in analysing and compressing the sensed data. Ganesan's DIMENSIONS [27] was one of the first systems addressing multi-resolution data access and spatio-temporal pattern mining in a sensor network. In [27], nodes are partitioned into different clusters and organized in a multi-level hierarchy. Within each cluster, the cluster head performs a two-dimensional wavelet transformation and stores the coefficients locally. These coefficients are passed to the next level in the hierarchy for wavelet transformation at a coarser resolution. While it demonstrates promising results, it relies on two key assumptions that limit its general applicability: (i) that nodes are distributed in a regular grid and (ii) that cluster heads can always communicate with their parents. Wagner [28] proposed an architecture for distributed wavelet analysis that removes the assumption about grid regularity. Moreover, Ciancio and Ortega [29] propose an algorithm for performing the wavelet transformation by tracing through the path in the minimum spanning tree and applying the wavelet filter along the path. It minimizes inter-node communication by transmitting partial coefficients forward and updating future sensors until the full coefficients are computed. This approach implicitly assumes that the path will be sufficiently long, so that wavelet analysis will be effective. Pattern *et al*. [30] propose SenZip, an architectural view of en route compression, which can interact with the routing components and makes use of 2D wavelet-based compression. SenZip allows for distributed compression configuration in each sensor node for routing based on the given aggregation tree. But aggregation node must wait for all the data from its children set before it can compute the partial coefficient, which introduces a non-negligible delay under the heavy traffic.

While several authors propose compression algorithms based on spatio-temporal correlation, some recent works do not assume any correlation based on spatial relationship of sensor nodes or on temporality between data packets. As such, they are more general and are better equipped to handle the sorts of low-correlation situations common to mobile sensor networks. Baar and Asanovi [31] develop an energy-aware lossless compression algorithm and shows that radio communication is the major contributor to power consumption. It shows that sending a single bit is equivalent to performing several hundred addition operations. When profiling such usage, it becomes obvious where the most energy can be saved, and that is in reducing the number of bits sent over the radio. Sadler and Martonosi [9] propose and evaluate a family of basic lossless compression algorithms, S-LZW, based on LZW [32] and tailored to static and mobile sensor networks and applicable to a wide range of applications. The authors discuss additional steps for transforming or preconditioning the data to further reduce energy consumption. S-LZW with Mini Cache (S-LZW-MC) is an extension that makes use of a cache of the most recently used entries. This additional cache serves as an appendix to the standard S-LZW dictionary, under the assumption that the sensor's incoming data tends to be repetitive over short intervals. The S-LZW-MC with Burrows–Wheeler transform (BWT) or structured transpose (ST) variations provide increased overall energy savings [31]. In the dictionary-based methods such as LZW and its extensions, senders and receivers maintain the same hash-indexed dictionary, either static or dynamic, and take the same symbol to represent the incoming data strings. However, as discussed by Baar and Asanovi [31], some packets may get lost during transmission and the dictionary may lose synchronization. Packet loss throughout the sensor network can be prohibitive for S-LZW compression algorithms that depend

on the reliable reception of preceding packets for decompression to be possible.

Some recent works also adopt linear prediction coding to data compression in wireless sensor networks. Puthenpurayil *et al*. [33] examine the performance of different variations of the linear predictive coding compression algorithm on different hardware configurations, but this algorithm is not robust to packet losses on unreliable networks. Huang and Liang [34] apply second-order linear prediction with coefficients determined *a priori* based on training data. Compression is applied in sensor networks to humidity sensor data using small (29-byte) packets. Rather than employing a compression scheme that allows independent decoding of packets to provide robustness to packet losses, they assume that lost packets will be re-transmitted as many times as needed. Alippi *et al.* [35] consider the problem of microacoustic/seismic data compression in wireless sensor networks. An algorithm is employed to automatically detect events, and no data are transmitted unless an event is detected. Prediction approaches considered are limited to (at most) taking the differences between successive samples. Compression is performed on blocks of 512 samples, which is fairly large compared with our application.

3. ADAPTIVE LINEAR PREDICTION COMPRESSION

In this section, we describe our linear predictive compression approach, originally presented in [13]. We model a sensor as a one-dimensional data source that produces integer-valued samples $x_1, x_2, \ldots$. The instrument has a dynamic range of b bits, and without loss of generality, we may assume that each sample value is in the range $[-2^{b-1}, 2^{b-1}-1]$. In our intended application, the source is a single component seismometer with instrument dynamic range of $b = 16$ bits. Each node has relatively modest computational power, and so our compression approach must have relatively low complexity. Encoded sample values are transmitted using fixed-length packets, and so we would like to losslessly encode as many samples as possible in each packet. A significant additional problem is that packets are often lost on the channel. For this reason we impose the additional constraint that the decoding of a received packet must not depend on the contents of other packets. We assume that time stamp information is already included with each packet, so the decoder can properly synchronize received sample values once they are decoded. Our compression approach relies on an adaptive linear prediction of sample values and entropy coding of prediction residuals.

3.1. Prediction

3.1.1. *Adaptive linear prediction*

We maintain a running estimate of the mean input signal value $\hat{\mu}_i$. This estimate is used to compute a de-biased version of the source samples

$$d_i = x_i - \hat{\mu}_i.$$

We apply Mth order adaptive linear prediction to the de-biased signal d_i. I.e. the predicted value $\hat{d}_i$ is a linear combination of the preceding M de-biased values,

$$\hat{d}_i = \sum_{j=1}^{M} w_j \cdot d_{i-j} = \mathbf{w}_i^{\mathrm{T}} \mathbf{u}_i. \tag{1}$$

Here $\mathbf{w}_i = [w_1, w_2, \ldots, w_M]^{\mathrm{T}}$ is a vector of weight coefficients that are adapted to the source and $\mathbf{u}_i = [d_{i-1}, d_{i-2}, \ldots, d_{i-M}]^{\mathrm{T}}$ is the vector of the preceding M de-biased sample values. The predicted sample value is

$$\hat{x}_i = \hat{\mu}_i + \hat{d}_i.$$

The estimation error, or prediction residual, is

$$e_i = x_i - \hat{x}_i = d_i - \hat{d}_i.$$

The prediction $\hat{x}_i$ is used to losslessly encode x_i using a variable length coding scheme described in Section 3.3. The entropy coding procedure takes into consideration the fact that x_i is an integer while $\hat{x}_i$ is usually not.

After encoding x_i, we use the sign algorithm [36] to update the weight vector:

$$\mathbf{w}_{i+1} = \mathbf{w}_i + \alpha \cdot \mathbf{u}_i \cdot \mathrm{sign}(e_i)$$

and we update the mean value estimate via

$$\hat{\mu}_{i+1} = \hat{\mu}_i + \beta \cdot (x_i - \hat{\mu}_i).$$

Here α and β are parameters that control the adaptation of the weight vector and mean estimate to the source statistics.

It might seem more natural to perform mean estimation as part of the sign algorithm instead of as a separate step. Under this alternative, one could define extended vectors $\mathbf{w}'_i = [w'_0, w'_1, \ldots, w'_M]^{\mathrm{T}}$ and $\mathbf{u}'_i = [\kappa, x_{i-1}, x_{i-2}, \ldots, x_{i-M}]^{\mathrm{T}}$ where κ is some fixed constant. Then we could predict the value of sample x_i directly as $\hat{x}'_i = \mathbf{w}'^{\mathrm{T}}_i \mathbf{u}'_i$. We do not adopt this alternative approach because the prediction of the first sample value in a packet becomes less straightforward and because compression effectiveness becomes somewhat more sensitive to parameter selection.

3.1.2. *Prediction using integer arithmetic*

To eliminate floating-point operations in the basic algorithm of Section 3.1.1, we use rational approximations to real-valued quantities to produce a version of the algorithm that requires only integer arithmetic. Specifically, the real-valued quantities $\hat{d}_i$, $\hat{x}_i$, $\hat{\mu}_i$, e_i and $\mathbf{w}_i$ are approximated using rational values:

$$\hat{d}_i = \hat{D}_i / 2^R,$$
$$\hat{x}_i = \hat{X}_i / 2^R,$$

$$\hat{\mu}_i = \hat{\Omega}_i/2^R,$$
$$e_i = E_i/2^R,$$
$$\mathbf{w}_i = \frac{1}{2^R}\mathbf{W}_i.$$

Here R is some fixed integer, $\hat{D}_i$, $\hat{X}_i$, $\hat{\Omega}_i$, E_i are integer variables, and $\mathbf{W}_i$ is a vector of integers. The value of R effectively controls the resolution at which linear prediction calculations are performed; we use $R = 14$ in our experiments. Since the sample values being predicted are integer-valued, compression effectiveness should not be highly sensitive to the value of R used, provided that R is not so large that arithmetic overflow occurs. In this version of the algorithm, we perform round-off operations that result in d_i being integer-valued, and, consequently, $\mathbf{u}_i$ being a vector of integers. The adaptation parameters α and β are chosen to be $\alpha = 2^{-A}$, $\beta = 2^{-B}$ for some integers A and B so that the multiplication needed to perform the updates can be accomplished via bit-shift operations.

Each iteration of the integer version of the prediction algorithm consists of the following steps:

(i) Compute
$$\hat{D}_i = \mathbf{W}_i^{\mathrm{T}}\mathbf{u}_i. \quad (2)$$
(ii) Compute
$$\hat{X}_i = \hat{D}_i + \hat{\Omega}_i.$$
(iii) Encode the integer sample value x_i using the rational predicted value $\hat{x}_i = \hat{X}_i/2^R$.
(iv) Compute the (integer) de-biased value d_i
$$d_i = x_i - \lfloor(\hat{\Omega}_i + 2^{R-1} - 1)/2^R\rfloor.$$
(v) Compute the prediction error
$$E_i = d_i \cdot 2^R - \hat{D}_i.$$
(vi) Update the weight vector
$$\mathbf{W}_{i+1} = \mathbf{W}_i + \mathrm{sign}(E_i)\lfloor(2^R\mathbf{u}_i + (2^{A-1} - 1)\mathbf{I})/2^A\rfloor,$$
where $\mathbf{I}$ denotes a vector of ones, and the floor operation is applied to each component of the vector.
(vii) Update the mean value estimate:
$$\hat{\Omega}_{i+1} = \hat{\Omega}_i - \lfloor(\hat{\Omega}_i - x_i \cdot 2^R + 2^{B-1} - 1)/2^B\rfloor.$$

3.2. Encoding into independent packets

To ensure that samples in a packet can be decoded without requiring preceding packets to be available to the decoder, we make the following modifications at the start of each packet:

(i) We place encoded quantized versions of $\hat{\mu}_i$ and $\mathbf{w}_i$ at the beginning of each packet. The values of $\hat{\mu}_i$ and $\mathbf{w}_i$ are set to these quantized versions in both the encoder and decoder. Quantization is uniform, using some numbers Q_μ bits of resolution for the value of $\hat{\mu}_i$ and Q_w bits for each component of $\mathbf{w}_i$.
(ii) We alter the prediction approach for the first M samples in the packet so that it does not rely on sample values in the preceding packet.

We describe these modifications in further detail below.

3.2.1. *Quantization*

The range of possible values of $\hat{\mu}_i$ is, at least in principle, equal to the range of possible sample values x_i. This range is uniformly partitioned into 2^{Q_μ} bins, and the index of the quantizer bin containing the value of $\hat{\mu}_i$ is encoded in the packet using Q_μ bits. The quantizer index could be encoded a little more efficiently using a variable length code since smaller magnitude values of $\hat{\mu}_i$ are presumably more likely than larger magnitudes, but we did not investigate such a scheme.

Each component of $\mathbf{w}_i$ is clipped as needed to ensure that its magnitude does not exceed some cap 2^C (we use $C = 2$ in our experiments). Each component is then quantized using a uniform quantizer with 2^{Q_w} bins spanning the range $[-2^C, 2^C]$. We make use of a simple variable length coding scheme to exploit the fact that the components of $\mathbf{w}_i$ are usually non-increasing in magnitude and alternating in sign, i.e.

(i) $|w_1| \geq |w_2| \geq |w_3| \geq \cdots$
(ii) $\mathrm{sign}(w_j) = (-1)^{j+1}$

We say that a quantized weight vector is *ordinary* if it satisfies these conditions.

We use a single bit to indicate whether the quantized weight vector is ordinary. If it is not ordinary, the weight components are sent uncoded using an additional $M \cdot Q_w$ bits. If the weight vector is ordinary, we encode $|w_1|$ directly using $Q_w - 1$ bits, and for $j > 1$, we encode the value of $|w_j|$ using $\lceil\log_2 |w_{j-1}|\rceil$ bits. In our experiments on seismic data, the quantized weight vector was ordinary more than 90% of the time.

3.2.2. *Predicting the initial samples in a packet*

Since the procedure for decoding a packet cannot depend on the contents of another packet, for the first M samples in a packet we must modify the regular prediction approach since we do not have enough data to perform the calculations of equations (1) or (2) directly. Instead, we do the following:

(i) The first de-biased sample value in a packet is predicted to be zero. That is to say, for the first sample, prediction makes use of the quantized bias estimate but not the weight vector.
(ii) For subsequent samples in the packet, when the calculation in equations (1) or (2) would require the use of samples from the preceding packet, the first de-biased sample value is repeated enough times to artificially produce M de-biased sample values to fill the vector $\mathbf{u}_i$.
(iii) Updates of the weight vector $\mathbf{w}_i$ (or $\mathbf{W}_i$) are not performed for the first M samples in a packet, during which time this modified prediction strategy is in effect.

3.2.3. Packet overhead and parameter tradeoffs

Compression-related packet overhead consists of the following:

(i) The quantized value of $\hat{\mu}_i$ (encoded using Q_μ bits)
(ii) The quantized value of $\mathbf{w}_i$ (encoded using at most $M \cdot Q_w + 1$ bits)
(iii) The value of an index indicating which variable length code was used to encode prediction residuals (using $\lceil \log_2 b \rceil$ bits); see Section 3.3.

We summarize the tradeoffs involved in selecting compression parameters:

(i) Higher order prediction (a larger value of M) allows for more accurate prediction, but increases the number of components of $\mathbf{w}_i$, thus generally increasing the amount of overhead used to encode $\mathbf{w}_i$ at the start of each packet.
(ii) Higher resolution quantization of $\hat{\mu}_i$ and $\mathbf{w}_i$ (i.e. larger values of Q_μ and Q_w) increases prediction accuracy but also increases packet overhead.
(iii) Larger adaptation step sizes (larger values of $\alpha = 2^{-A}$, $\beta = 2^{-B}$) allow faster initial adaptation and adaptation to a dynamically changing source, but provide worse steady-state performance when the source is not rapidly changing. In our experiments, we set $A = 15$ and $B = 8$.

3.3. Entropy coding

Source sample values are collected in a buffer. After each source sample arrives, we determine whether the encoded bit cost of the samples in the buffer exceeds the available space in the packet. If not, then we proceed to the next sample. Otherwise, we encode the samples in the buffer (excluding the newest one) using the entropy coding procedure described in the remainder of this section and reset the buffer to contain only the newest sample.

Thus, with the arrival of each new sample, we must determine whether the accumulated samples fit within a packet. Since the coding option used for a packet can change as new samples arrive, explicitly computing the encoded length of the samples in the buffer is not always as simple as incrementing an encoded bit count with the cost of the new sample. The obvious brute-force approach is to simply apply the entropy coding procedure to the samples in the buffer.

Fortunately, we can often avoid the brute-force calculation by bounding the encoded bit cost to quickly identify cases where the packet can accommodate the accumulated samples. For example, the entropy coding procedure guarantees that the average bit cost to encode n samples is no more than $n \cdot b$ bits. As another example, if we have computed the bit cost to encode the first $(n-1)$ samples, we can bound the cost to encode n samples based on a bound on the incremental cost to encode a single sample. We omit further details of our approach to bounding the encoded bit cost.

Our entropy coding problem then is to efficiently encode a length-n sequence of integer-valued samples x_i given real-valued predictions $\hat{x}_i$. To do this, we map each sample value to a non-negative integer and then encode the resulting sequence of non-negative integers using a Golomb code. This general strategy is used in the Rice entropy coding algorithm [10, 11] and the LOCO-I image compressor [12], among myriad other applications.

3.3.1. Mapping

It is sensible to refine the predicted value $\hat{x}_i$ to take into account the fact that the true sample value x_i is an integer and is constrained by the instrument dynamic range. Accordingly, we define

$$[\hat{x}_i] = \min\{\max\{\text{round}(\hat{x}_i), x_{\min}\}, x_{\max}\},$$

where $x_{\min} = -2^{b-1}$ and $x_{\max} = 2^{b-1} - 1$ are the minimum and maximum possible sample values. We use this refined prediction to calculate the integer-valued prediction residual

$$\tilde{e}_i = x_i - [\hat{x}_i].$$

We map the signed integer quantity $\tilde{e}_i$ to a non-negative integer f_i using a slight variation on the mapping used in [10, 11]:

$$f_i = \begin{cases} 2|\tilde{e}_i| - \delta_i & \text{if } |\tilde{e}_i| \le \theta, \\ |\tilde{e}_i| + \theta & \text{otherwise.} \end{cases}$$

Here we define

$$\theta = \min\{[\hat{x}_i] - x_{\min}, x_{\max} - [\hat{x}_i]\}$$

and

$$\delta_i = \begin{cases} 1, & \text{sign}(\tilde{e}_i) = \text{sign}(\hat{x}_i - [\hat{x}_i]), \\ 0 & \text{otherwise.} \end{cases}$$

This mapping is invertible and ensures that $f_i \in [0, 2^b - 1]$, i.e. f_i is a non-negative integer with dynamic range that matches that of the original source. More significantly, the mapping assigns smaller magnitude residuals $\tilde{e}_i$ to smaller values of f_i. Since smaller magnitude prediction residuals should occur more frequently than larger magnitudes, we would like to encode f_i using a variable length code that assigns shorter codewords to smaller integers, such as the codes that we discuss next.

3.3.2. Variable length coding

For positive integer m, the mth Golomb code [37] defines a reversible prefix-free mapping of non-negative integers to variable length binary codewords.

We restrict our choices to codes for which $m = 2^k$ for some non-negative integer k. As noted in [37], coding in this case becomes especially simple. The codeword for the integer j consists of the unary representation of $\lfloor j/2^k \rfloor$ (that is, $\lfloor j/2^k \rfloor$ zeros followed by a one) concatenated with the k least significant

bits of the binary representation of j. Following the convention of [12], we refer to this special case as a Golomb-power-of-2 (GPO2) code with parameter k.

The samples in a packet are either all sent uncoded, using b bits for each sample, or they are all encoded using the same GPO2 code with some fixed parameter k. The coding option selected is explicitly encoded as part of the packet overhead, as described in Section 3.2.3. For a source with b-bit dynamic range, the cost of using code parameter $k \geq b-1$ is always at least as large as the cost of sending the samples uncoded [38]. Thus, in our application, when we use a GPO2 code, it must have parameter k satisfying

$$0 \leq k \leq b-2.$$

This gives us $(b-1)$ GPO2 code choices, along with the uncoded option, so our selected code can be indicated using $\lceil \log_2 b \rceil$ bits of overhead.

The coded samples do not usually perfectly fill a packet; following the last encoded sample in the packet, any remaining unused bits in the packet (fill bits) are set to zero. Because each GPO2 codeword begins with a unary-encoded value, these fill bits will never be mistaken for a coded sample value, and so the decoder can correctly determine the number of samples encoded in the packet.

We now turn our attention to the problem of selecting a coding option that efficiently encodes some number n of non-negative integers $f_1, f_2, \ldots, f_n$. The traditional solution to this problem is the Rice algorithm's brute-force approach: explicitly compute the coding cost of each option and select the best one [10, 11]. But it was shown in [38] that the brute-force approach is unnecessarily complex; if we simply compute the sum

$$F = \sum_{i=1}^{n} f_i,$$

then the mean value F/n allows us to narrow the possible optimum code choices to at most three candidates. Furthermore, by simply comparing this mean value to a list of pre-defined thresholds, we can perform code selection in a way that gives compression effectiveness that is quite close to that obtained under an optimum code selection.[1]

Our code selection procedure uses the following steps (further details and mathematical background can be found in [38]):

(i) If the mean value F/n is sufficiently large, the f_i are sent uncoded. Specifically, we first check if $F/n > \mu_b^{\dagger} 1/2^{2^{2-b}} - 1$ ($\mu_{16}^{\dagger} \approx 23637$). If this condition is satisfied, then the uncoded option is selected. Otherwise, proceed to the next step.

(ii) Compute K as follows. If $F/n + \frac{49}{128} < 1$ then $K = 0$; otherwise, K is the unique non-negative integer satisfying $2^K < Fn + \frac{49}{128} \leq 2^{K+1}$. This can be implemented in C source code as:

```
for (K=0; (n<<(K+1))<= F+(N*49>>7); K++)
    ;
```

(iii) Assign $k = \min\{K, b-2\}$.

(iv) Compute the bit cost of using the GPO2 code with parameter k to encode $f_1, f_2, \ldots, f_n$. If this cost exceeds the uncoded cost (which is $n \cdot b$ bits) then we use the uncoded option; otherwise, we select the GPO2 code with parameter k.

Analysis in [38] shows that the GPO2 code parameter k selected under this strategy is always within one of the optimum parameter values. Our experiments with seismic data samples suggest that the increased bit rate due to occasional suboptimum code selection is negligible.

3.4. Numerical example

In this section, we will present a numerical example to illustrate each step of the ALFC compression procedure. Here we use parameter values $A = 15$, $B = 8$, $R = 14$, $M = 3$. The initial weight vector is: $\{w_1, w_2, w_3\} = \{1.5, -1.25, 1.0\}$. As stated in Section 3.1.2, the rational approximation to real value is adopted to generate only integer arithmetic. Suppose the incoming samples are: $\mathbf{x} = \{x_0, x_1, x_2, x_3, \ldots\} = \{90, 94, 92, 124, \ldots\}$. The first M de-biased values are $\{d_0, d_1, d_2\} = \{-4, 2, 0\}$. Next, we will illustrate how to apply ALFC to compress the specific value of sample i, here we select $i = 4$.

In the first stage, we demonstrate how to predict a sample value.

1. Applying equation 2, we compute $\hat{D}_i = \mathbf{W}_i^{\mathrm{T}} \mathbf{u}_i = \sum_{j=1}^{M} 2^R w_j \cdot d_{i-j} = -139\,264$. The first M sample predictions are slightly different from the rest samples in the packet, that is, if $i < j$, $d_{i-j} = d_0$.
2. We update the input mean estimate from the previous estimate and the last arrived sample: $\hat{\Omega}_i = \hat{\mu}_{i-1} \cdot 2^R - \lfloor (\hat{\mu}_{i-1} \cdot 2_{i-1}^R - x_{i-1} \cdot 2^R + 2^{B-1} - 1)/2^B \rfloor = 96 \cdot 2^{14} - 256 = 1\,572\,608$. The last estimate of mean input value $\hat{\mu}_{i-1} = 96$. And the last arrived sample value $x_{i-1} = 92$.
3. We can compute the $\hat{X}_i = \hat{D}_i + \hat{\Omega}_i = 1433344$ and the $\hat{x}_i = \hat{X}_i / 2^R = 87.48$. Here, we get our prediction sample value of 87.48.
4. We compute the integer de-biased $d_i = x_i - \lfloor (\hat{\Omega}_i + 2^{R-1} - 1)/2^R \rfloor = 28$. Here the current sample value $x_i = 124$.
5. We compute the prediction error: $E_i = d_i \cdot 2^R - \hat{D}_i = 598016$.
6. We need to update the weight vector for the next sample prediction, if we have collected more than M samples:

[1] The Rice algorithm also differs from our coding problem in that the Rice coder encodes a fixed number of samples into a variable number of encoded bits, whereas in our problem we encode a variable number of samples into fixed-length packets.

$\mathbf{W}_{i+1} = \mathbf{W}_i + \text{sign}(E_i) \cdot \lfloor (2^R \mathbf{u}_i + (2^{A-1} - 1)\mathbf{I})/2^A \rfloor = \{24\,574, -20\,481, 16\,384\}^T$. If the $i < M$, then we do not update the weight vector.

Next, we demonstrate how to do the mapping and encoding based on the predicted sample value. First, we round the predicted value: $[\hat{x}_i] = \min\{\max\{\text{round}(\hat{x}_i), x_{\min}\}, x_{\max}\} = 87$. Here, the predicted value has been rounded down from 87.48 to 87. Second, we calculate the prediction residual: $\tilde{e}_i = x_i - [\hat{x}_i] = 37$, which is smaller than $\theta = 32680$. Third, we map the integer residual to a non-negative integer $f_i = 73$. Fourth, suppose we finally select the $k = 5$ according to the k selection procedure. f_i is encoded as two parts: unary representation of 001 and binary representation of last five least significant bits of 01001. Therefore, the final output of encoded value is 00101001.

In the decompression procedure of ALFC, we first decode the parameters in the beginning of each packet. And we can start to do the prediction in the same way as compressor does, generating predicted value. Finally, we can losslessly restore the real sample value based on the decoded prediction residual and the predicted value in decompressor.

4. EXPERIMENTS AND EVALUATIONS

We have conducted compression evaluations on a real sensor network testbed including several sensor nodes. One such node is pictured in Fig. 1. The core component of each node is an iMote2 sensor mote, a new generation hardware platform for wireless sensor networks.

The iMote2 CPU core frequency can be configured to operate from 13 to 416 MHz. In our evaluation, we configured the PXA271 processor on the iMote2 to operate in a low voltage (0.85 V) and low frequency (13 MHz) mode to conserve energy. An MDA320CA sensor board is connected to the iMote2 through a serial peripheral interface, and a low-pass filter with cut-off frequency of 100 Hz is added to the sensor board to reduce noise.

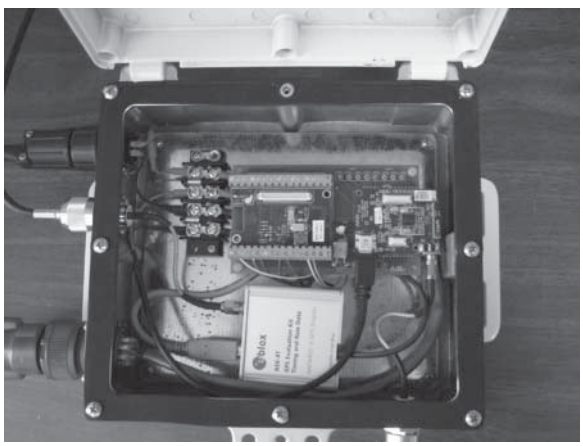

FIGURE 1. A sensor node in OASIS [6] project.

To provide real data sets to the node for compression evaluation, we wired the sensor connectors to a PCI-DAC6703 (Digital Analog Converter) board installed in a PC. The PCI-DAC6703 provides 16 channels of 16-bit analog output and eight digital I/O bits. We read real data sets from an archive and wrote them to the PCI-DAC6703 board. In this way the PCI-DAC6703 simulated real sensor data during the experiments. To form a multi-hop network, we set the radio power to level 2, which is the lowest setting.

In our compression experiments, we use the following U. S. Geological Survey (USGS) seismic data sets[2]:

1. FEQ: Frequent earthquake swarm data from the old dome, collected on 26 September 2004.
2. LLA: Low-level activity, some rockfalls in crater, collected on 20 July 2007.
3. NS: Noisy storm with rain, wind, flood and mudflows, from the station on the crater floor, collected on 6 November 2006.
4. S-H: High-quality (i.e. high signal-to-noise ratio) seismic data collected on 10 August 2009 from a geophone seismic sensor.
5. S-L: Low-quality seismic data collected on 10 August 2009 from a seismometer sensor.

The instrument dynamic range of original data set of FEQ, LLA and NS are 12 bits, which include useful data for evaluating event trigger algorithms and compression algorithms. And the dataset S-H and S-L are 16-bit data. The data streams injected into the testbed are regenerated as 16-bit sample based on the source seismic data.

Users of ALFC compression must select the values of several parameters described in Section 3. To observe the impact of parameter choices, we measured the compressed bit rate obtained under different choices of parameter values using the S-H seismic data set as input to the testbed. And the S-H was chosen as the test set because it is representative of the data we expect to see in our application. Here we fix $R = 14$ and $M = 3$. Section 4.1 presents further evaluations when the prediction order M is varied. We set the radio packet payload to 56 bytes, which represents a compromise between larger packets, which tend to be more vulnerable to corruption during radio transceiving, and shorter packets, which spend a larger fraction of their payload on overhead and thus yield less effective compression.

As discussed in Section 3.1.2, smaller values of A and B result in faster adaptation to a dynamically changing source, but lead to worse performance on a steady-state data input. Table 1 compares the performance of some different combinations of A and B. Of the combinations considered here, $A = 15$ and $B = 8$ has the best compression performance and relatively less standard deviation, which is the deviation of the bit rate over the packets in the data set. So we use these values for all further

[2] http://oasisvalve.vancouver.wsu.edu:8080/valve3/.

TABLE 1. Impacts of different parameter selections

A	B	Compressed bit rate (bits/sample)	Standard deviation
4	8	16.86	0.327
8	8	11.24	0.482
15	**8**	**6.52**	**0.894**
20	8	8.38	0.650
15	4	7.38	0.638
15	15	17.92	0.341
15	20	17.92	0.327

Bold values indicate the number of max value, which is also the output of desired parameters.

TABLE 2. Impacts of different quantization resolutions

Q_μ	Q_w	Compressed bit rate (bits/sample)	Standard deviation
5	5	14.87	0.661
5	11	16.00	0.000
10	10	7.38	0.248
11	**5**	**6.52**	**0.765**
15	15	6.85	0.468

Bold values indicate the number of max value, which is also the output of desired parameters.

results hereinafter. Finally, Table 2 shows the performance of some different combinations of quantizer resolution parameters Q_μ and Q_w. We select $Q_\mu = 11$, $Q_w = 5$, because this combination produces the lowest compressed bit rate of the values evaluated.

4.1. Compressed bit rate and resource usage

We also compare the compression performance of ALFC with the S-LZW compressor presented in [9], which is a recently developed compressor developed for sensor networks, discussed in Section 2.2. S-LZW is a dictionary-based compressor that partitions input data into small blocks to achieve a balance between dictionary size and the 'hit' rate, i.e. the rate at which matches are found in the dictionary. Both ALFC and S-LZW are low-complexity lossless compression algorithms that are designed for sensor networks and do not depend on dependencies between nodes in the network. S-LZW-MC-BWT can achieve the best compression among the relevant S-LZW variations.[3] It is worth keeping in mind that dictionary-based methods such as S-LZW variations are much less robust to packet losses than ALFC.

Table 3 lists the compressed bit rates achieved on all the data sets using the S-LZW-MC-BWT compressor with different mini-cache sizes, and for ALFC with different prediction orders. It is important to note that in each of our experiments, the input data are first low-pass filtered within the node (as described above) before the lossless compression algorithm is applied. This step has the effect of reducing high frequency noise, and thus presumably improves compression effectiveness.

Over the range of parameters evaluated, ALFC demonstrates an advantage in compression effectiveness for all the data sets. Both compressors provide much better compression on the better-behaved SS data set. The volatile data in the seismic data set reduces the MC's hit rate in S-LZW-MC-BWT, and reduces the accuracy of linear prediction in ALFC. Interestingly, on the low quality seismic data set, first-order prediction outperforms the higher order prediction in ALFC.

We next compare the code size and memory usage of the algorithms. The default memory usage of S-LZW-MC is about 2 kB, which is smaller than other compression algorithms presented in [9]. Our ALFC implementation uses even less memory: 768 bytes, which makes it truly lightweight and suitable for resource-constrained sensor nodes. Our ALFC implementation also has smaller code size of 19.4 kB, compared with 32 kB required for S-LZW-MC. S-LZW-MC-BWT has even larger code size because it includes also a Burrows–Wheeler transform. Computation cost comparisons between S-LZW-MC-BWT and ALFC are somewhat less direct because the former is implemented on an 8 MHz TI MSP430 processor (with 10 kB RAM and 48 kB on-chip flash memory) while the latter is implemented on the iMote2 running at 13 MHz. Prediction order was observed to have little impact on the compression speed of ALFC. Compared with S-LZW-MC-BWT, our ALFC implementation provides slightly slower but more effective compression. Fig. 2 shows compression effectiveness and computation cost of our ALFC implementation for the different data sets.

Figure 2 shows the compressed bit rate of ALFC when the prediction order M varies. ALFC performs better on the S-H data set than on S-L data sets, with an average compressed bit rate of 7.08 bits per sample. For the FEQ data set, $M = 5$ achieves the best compressed bit rate of 6.96 bits per sample. Fifth- order prediction also delivers the best performance for the LLA data set. For the NS data set, the compressed bit rate of $M = 3$ is 9.30, which is 11.7% better than that of $M = 4$. $M = 1$ also performs better than $M = 4$, suggesting that higher order linear prediction is not worth the added overhead on some data sets.

Besides the compressed bit rate, we also evaluate local energy savings under ALFC, i.e. the energy saved when nodes use compression and transmit compressed packets rather than raw packets. Here we take into account the energy for computation, memory access and radio communication. The iMote2 uses a CC2420 radio transceiver.[4] Fig. 3 shows that for all of the test

[3] Sadler and Martonosi [9] also introduce the S-LZW-MC-ST variation, but this compressor assumes that the pattern of the data to be compressed is known in advance and attempts to reorder the data to improve compression performance. Since we assume no such advance knowledge, we make no comparisons with this variation.

[4] http://focus.ti.com/lit/ds/symlink/cc2420.pdf.

TABLE 3. Average compressed bit rate (bits/sample) using S-LZW-MC-BWT and our linear predictive algorithm.

	S-LZW-MC-BWT				Proposed algorithm			
	Mini-cache size				Prediction order *M*			
Data set	8	16	32	64	1	3	4	5
FEQ	8.74	8.51	8.19	**8.06**	7.69	7.66	8.74	**6.96**
LLA	9.12	8.96	8.67	**8.58**	9.20	8.70	9.47	**8.42**
NS	10.62	10.56	10.50	**10.37**	9.41	**9.30**	10.53	9.94
S-H	8.51	8.42	8.16	**7.90**	7.76	6.52	7.55	**6.48**
S-L	10.59	10.62	10.53	**10.37**	**9.15**	9.72	9.90	9.94

Bold values indicate the number of max value, which is also the output of desired parameters.

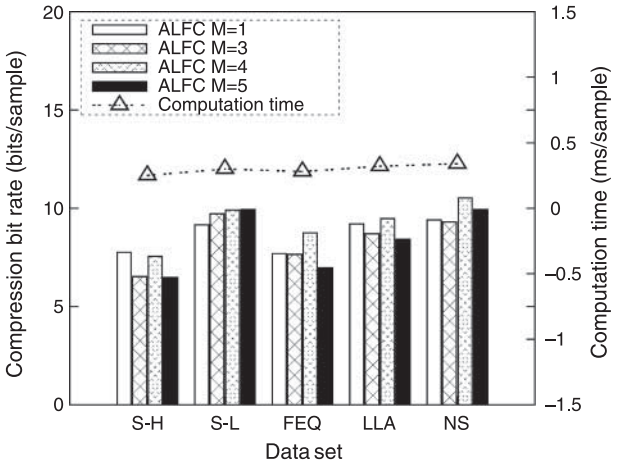

FIGURE 2. Comparison of compressed bit rate and computation cost.

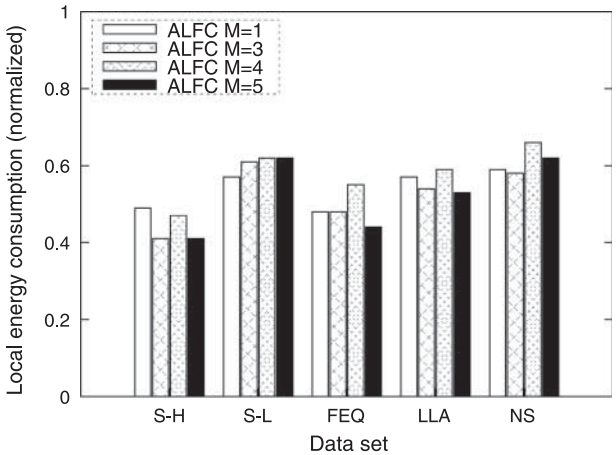

FIGURE 3. Normalized local energy consumption is the ratio of energy consumption with compression against the energy consumption without compression.

data sets, sensor nodes with ALFC consume less local energy than when no compression is used. For the seismic data sets from volcanos, fifth-order prediction minimizes local energy consumption. The figure also indicates that the local energy saving depends on the dynamics of the source data.

4.2. Network throughput and energy efficiency

The S-LZW-MC-BWT implementation of [9] partitions an input data sequence into 528-byte blocks that are each compressed independently. When relatively short packets are used (as in our experiments using 56-byte packets), a single block will almost always be too large to be transmitted using a single packet, and the loss of a single packet will render all subsequent packets for that block. By contrast, ALFC allows each packet to be decompressed independently and thus provides much more robustness to packet losses in this scenario. Thus, in a mobile or lossy sensor network with a high packet loss ratio, we would expect the effective data delivery ratio of ALFC to be higher than that of S-LZW-MC-BWT. In Section 4.1, we showed that the ALFC has better compression performance than that of S-LZW-MC-BWT. Reduced data transmission implies less network energy consumption as well.

In this section, we evaluate the performance of our ALFC algorithm in a multi-hop environment using various seismic data sets with varying network traffic. The testbed consists of five sensor nodes arranged in a multi-hop topology, shown in Fig. 4. The routing protocol is an enhanced version of MultihopLQI in TinyOS-1.x. The underlying MAC protocol is the default MAC protocol in TinyOS-1.x, which is a CSMA-based MAC protocol. A desktop PC is connected to the sink node to collect upstream data from all sensor nodes. The data injected into sensor nodes for our tests is from the FEQ data set, which contains 30 000 packets of continuous seismic data. The payload of each packet is 56 bytes. The injected data rate into each node ranges from 5.4 to 20.8 kbps. The duration of experiments ranges from 10 to 40 min, and results are averaged over five repeated experiments.

We first evaluate the energy saving in a network environment. Improved compression yields reduced transmissions, thus resulting in lower energy consumption from radio transceivers. In addition, for a given amount of raw data, increased

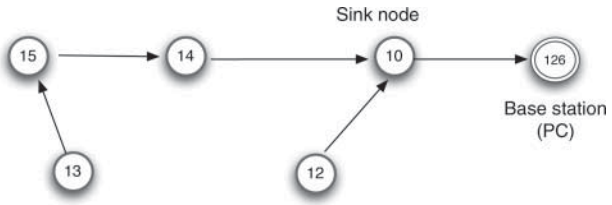

FIGURE 4. Topology of the multi-hop sensor network testbed.

compression reduces network load, which leads to fewer collisions and thus fewer retransmissions. We observed from Fig. 2 that fifth-order prediction provides the most effective compression among alternatives evaluated, and thus minimizes energy consumption over the network. Fig. 5a shows that the margin becomes even larger with increasing data rate. When the data rate is 20.8 kbps, the energy consumption without compression is as high as 2.5, which is almost twice that of ALFC with fifth-order prediction. Thus, we see that in this case compression significantly reduces energy consumption by reducing the total amount of transmissions.

We further evaluate the network efficiency by measuring the packet delivery ratio and energy efficiency. Packet delivery ratio refers to ratio of the successfully decompressed raw data packets at the gateway against the initially collected raw packets at sensor nodes. When the network is not saturated (e.g. the data rate from 5.4 to 10.8 kbps), the delivery ratio of compressed data flow is slightly lower than the case without compression. This is because compressed packets encode more raw data samples than uncompressed packets, and thus the loss of each compressed packet results in the loss of a larger number of samples. However, as the data rate increases, the network becomes saturated and congested, more packets are dropped due to buffer overflow or transmission collision. When the data rate increases from 10.8 to 14.1 kbps, the delivery ratio of the uncompressed data flow quickly drops from 0.92 to 0.69, while that of the compressed data flow only drops slightly from 0.9 to 0.8. It can be seen in Fig. 5b that fifth-order prediction provides the best delivery ratio of 0.74 at the data rate of 20.8 kbps, while that of the uncompressed data flow is only 0.5. The problem of buffer overflow failure and channel failure due to collision is mitigated by compression. This demonstrates the importance of using an effective compression algorithm to support high data-rate sensor networks.

Finally, we evaluate energy efficiency, which refers to the number of delivered bits per energy unit. It can be calculated based on the delivery ratio divided by the network energy consumption. Using fifth-order prediction, the energy efficiency index drops slowly from 2.4 to 1.0 as we increase the data rate from 5.4 to 20.8 kbps. Because of packet loss and network congestion, the energy efficiency of the data flow with no compression applied suffers from a sudden drop at the data rate of 10.8 kbps, and keeps decreasing to 0.2 at the data rate of 20.8 kbps. Fig. 5c indicates that, on average, energy efficiency is improved by more than a factor of two by our ALFC algorithm compared with the uncompressed case.

4.3. ALFC in a field network

We evaluated ALFC not only in the lab testbed, but also in a field network deployed for volcano monitoring. The OASIS [6] project deployed 15 stations on Mount St. Helens volcano in the summer of 2009 to monitor volcanic activities. ALFC is applied to mitigate the difficulty in collecting real-time high-fidelity signals given the limited bandwidth.

4.3.1. System configuration

Figure 6 depicts the deployment map of 15 stations around the crater of Mount St. Helens volcano. The full coverage of deployed sensor network reaches a diameter of about 6 miles, and it covers not only the crater area of volcano, but also some of the flank areas. The whole sensor network is divided into two subnetworks by allocating different radio channels. The first and second network branches are comprised of nodes 0–6 and nodes 7–14, respectively. Nodes 0 and node 7 serve as the sink nodes for their respective branches. The nodes form a multi-hop network with the topology shown in Fig. 7. To visualize the real time network topology and sensor data, we built an end-to-end system for the OASIS project. Two deployed branch networks

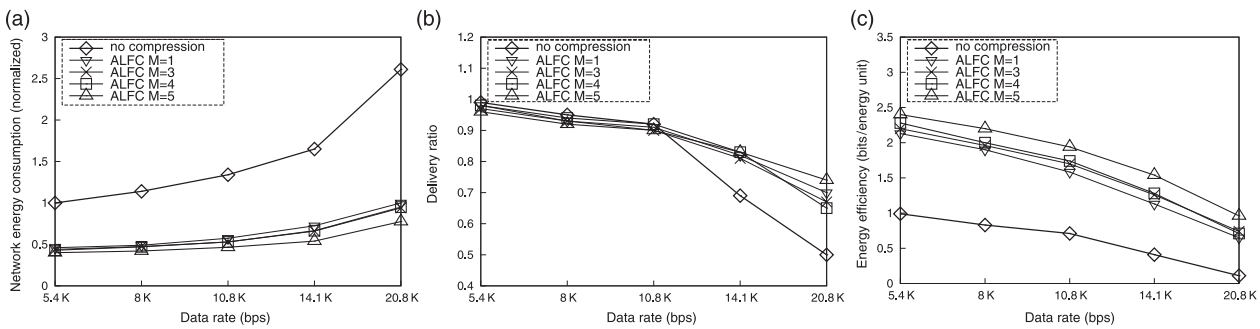

FIGURE 5. The performance evaluation based on the FEQ data set in a multi-hop network with varying data rate: (a) network energy consumption, (b) delivery ratio and (c) energy efficiency.

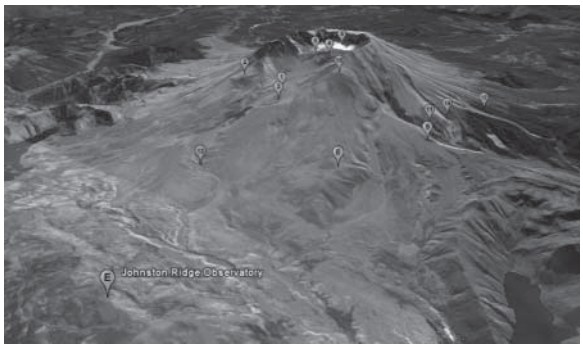

FIGURE 6. OASIS deployment map.

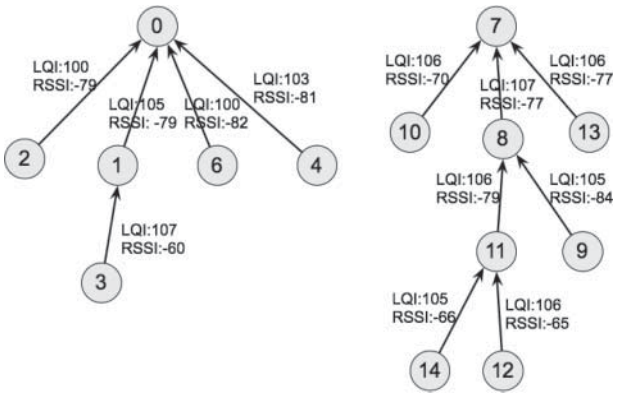

FIGURE 7. OASIS deployment topology.

are in charge of collecting real time high fidelity data. One gateway relays the data stream to a database through a 50-mile microwave link. In our lab, a customized version of the TinyOS SerialForwarder tool forwards the data between the sensor network and the Internet. Multiple end clients may connect to it, access the sensor data stream, and control the network in real time. The ALFC decompressor is conducted inside the SerialForwarder, so that the output is the decompressed data stream, which makes the ALFC compression transparent to end clients.

Each node collects data from several sensors, including seismic data, infrasonic data and lightning data, as well as GPS data. We also collect additional data to assess network performance, such as link quality index (LQI), received signal strength index (RSSI), parent ID, battery voltage, etc. All of these data streams are logged into the volcano analysis and visualization environment (VALVE) database for permanent storage. VALVE provides web interfaces allowing on-demand visualization of history sensing data from any location on the Internet. Based on the collected data, the deployment topology can be viewed in real time. The LQI and RSSI shown in Fig. 7 are retrieved from the collected data, and they are valuable for network diagnosis.

4.3.2. ALFC evaluation in volcano

Mount St. Helens is one of the most active volcanos in the world, and the seismic data are highly dynamic, and tend not to yield much compression. In Fig. 8, the vertical bars illustrate the average compressed bit rate of the seismic data on each sensor node from a typical 4-h continuous data stream. Nodes 0, 7 and 5 are not included in Fig. 8. The sink nodes (0 and 7) have no seismic sensors, while transmissions from node 5 cannot be received by other nodes, and thus node 5 is not connected to the network. An observable margin in the compressed bit rate can be found in node 1. Node 1 has an averaged compressed bit rate of about 7.44, which is 22.7% better than other nodes. That is because node 1 uses a high-fidelity geophone seismic sensor which produces data with lower noise levels than the other sensor nodes which use three low-cost MEMS accelerometers. According to the data collected by our deployed network, the deviation of the noise readings from an accelerometer is in the range of 0–100, while that from a geophone sensor is 0–20. Moreover, the standard error on node 1 indicates that compressed bit rate experiences a fairly large fluctuation. This is because node 1 is deployed in a much more seismically active spot, where much more seismic activity occurs. According to the data logged in our database, node 1 experienced about 60 seismic events in this 4-h period, while other nodes only have approximately 16 seismic events or fewer.

Figure 9a shows the compressed bit rate averaged over blocks of 280 samples at nodes 1 and 6, which are the two nodes experiencing the most seismic events, and Fig. 9b plots the seismic waveform data at these nodes. We make two observations from this figure. First, the bit rate is consistently much lower (typically by more than two bits/sample) at node 1 than at node 6. This difference is due to the lower noise level at node 1, as explained above. Second, the spikes in the compressed bit rate at the two nodes have correlation with the seismic events observable in Fig. 9b, though the correlation is

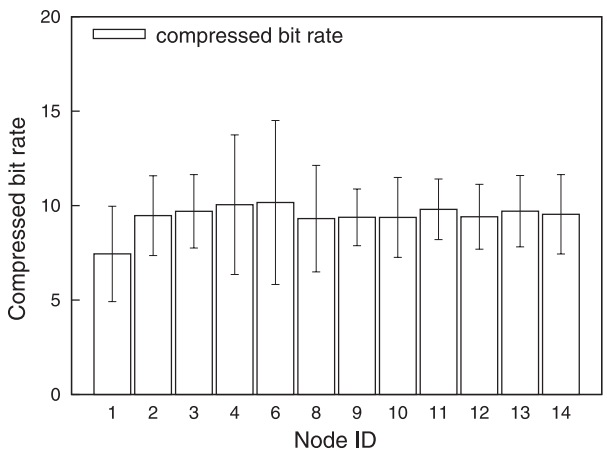

FIGURE 8. Averaged compressed bit rate of each node based on a 4-h seismic data stream.

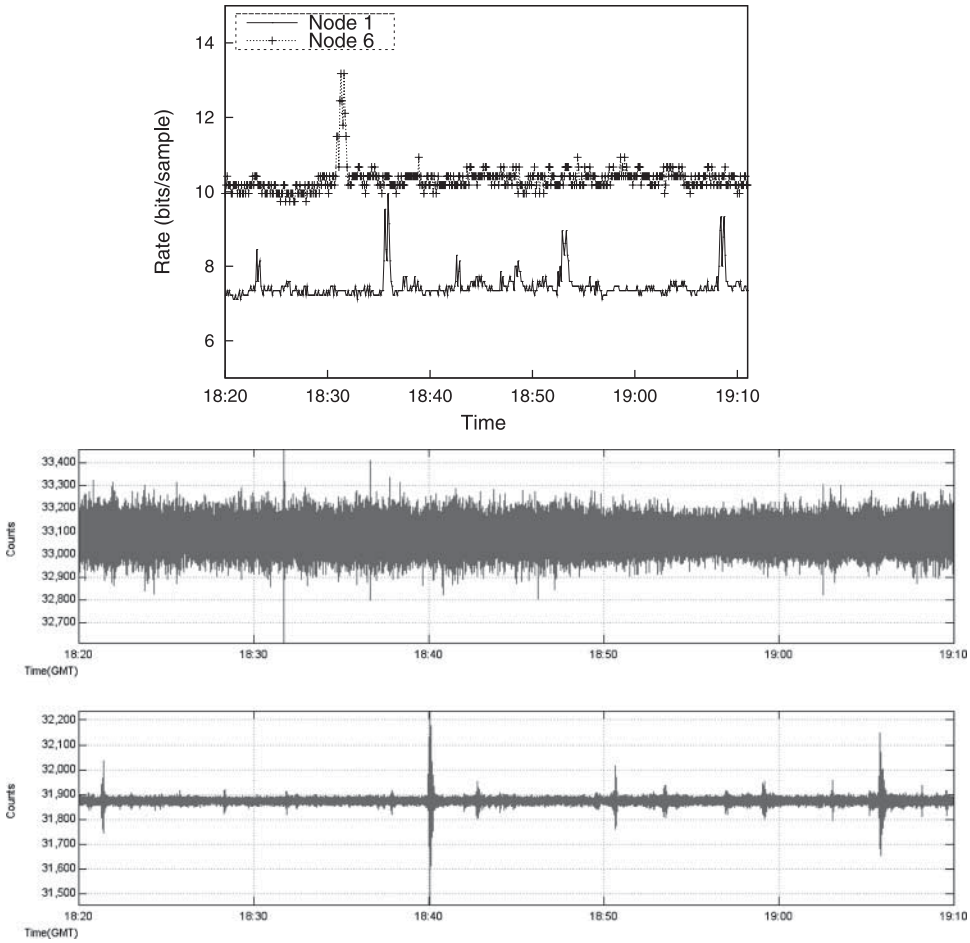

FIGURE 9. (a) Compressed bit rate. Each data sample has 16 bits before compression and (b) the seismic waveform of nodes 1 and 6—the geophone seismic sensor of node 1 has lower noise level.

not the exact match due to the fact that bit rate is calculated every 280 samples.

The results above show that ALFC performs effectively in a field network deployed on an active volcano that generates highly dynamic seismic data. The results also show the significant impact of sensor noise level on compression performance for the ALFC algorithm.

5. CONCLUSION AND FUTURE WORK

We have presented a lightweight lossless compression algorithm based on adaptive linear prediction of sample values and entropy coding of prediction residuals. The relatively low overhead of linear prediction makes it suitable for real-time high-fidelity compression. Decompression does not rely on previously received packets, hence compression is highly robust to packet losses. We have also evaluated our approach on a real sensor network testbed by injecting real seismic data sets, and in the field on an active volcano.

We note two areas for potential improvement. The first improvement would be to devise a more effective strategy for encoding overhead information than the one currently employed. Along these lines, we could envision strategies for joint quantization and encoding of overhead information. For example, we might want to alter the quantization of $\mathbf{w}_i$ depending on the magnitude of the bias estimate or the value of the GPO2 parameter k. Second, the algorithm would clearly be more practical if it were able to adaptively choose quantizer resolution based on observation of the data, rather than requiring user intervention to make this selection.

ACKNOWLEDGEMENTS

The research was carried out at the Jet Propulsion Laboratory, California Institute of Technology, and at Washington State University. We would like to thank all the colleagues in the OASIS project for their work and the reviewers for their valuable feedback.

FUNDING

The research described in this paper was supported by NASA ESTO AIST program and USGS Volcano Hazard program under the research grant NNX06AE42G.

REFERENCES

[1] Szewczyk, R., Mainwaring, A., Polastre, J., Anderson, J. and Culler, D. (2004) An analysis of a large scale habitat monitoring application. *Proceedings of 2nd ACM Conference on Embedded Networked Sensor Systems (SenSys 2004)*, Baltimore, MD, USA, Nov. 2004.

[2] Hill, J., Szewczyk, R., Woo, A., Hollar, S., Culler, D. and Pister, K. (2000) System architecture directions for networked sensors. *Proceedings of SIGOPS Oper. Syst. Rev.*, **34**, pp. 93–104, Dec. 2000.

[3] Intanagonwiwat, C., Govindan, R. and Estrin, D. (2000) Directed diffusion: a scalable and robust communication paradigm for sensor networks. *Proceedings of the 6th annual international Conf. on Mobile computing and networking (MobiCom 2000)*, Boston, MA, USA, Aug. 2000.

[4] Krishnamurthy, L., Adler, R., Buonadonna, P., Chhabra, J., Flanigan, M., Kushalnagar, N., Nachman, L. and Yarvis, M. (2005) Design and deployment of industrial sensor networks: Experiences from the north sea and a semiconductor plant. *Proceedings of 3rd ACM Conf. Embedded Networked Sensor Systems (SenSys 2005)*, San Diego, CA, USA, Nov. 2005.

[5] Werner-Allen, G., Lorincz, K., Johnson, J., Lees, J. and Welsh, M. (2006) Fidelity and yield in a volcano monitoring sensor network. *Proceedings of 7th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, Seattle, WA, USA, Nov. 2006.

[6] Kedar, S., Chien, S., Webb, F., Tran, D., Doubleday, J., Davis, A., Pieri, D., Song, W.-Z., Shirazi, B. and Lahusen, R. (2008) Optimized autonomous space in-situ sensor-web for volcano monitoring. *IEEE Aerospace 2008*, Big Sky, MT, USA, Mar. 2008.

[7] Xu, N., Rangwala, S., Chintalapudi, K.K., Ganesan, D., Broad, Alan., Govindan, R. and Estrin, D. (2004) A wireless sensor network for structural monitoring. *Proceedings of 2nd ACM Conf. on Embedded Networked Sensor Systems (SenSys 2004)*, Baltimore, MD, USA, Nov. 2004.

[8] Pakzad, S.N., Kim, S., Fenves, G.L., Glaser, S.D., Culler, D.E. and Demmel, J.W. (2005) Multi-purpose wireless accelerometers for civil infrastructure monitoring. *Proceedings of 5th International Workshop on Structural Health Monitoring (IWSHM 2005)*, Stanford, Sep. 2005.

[9] Sadler, C.M. and Martonosi, M. (2006) Data Compression Algorithms for Energy-Constrained Devices in Delay Tolerant Networks. *Proc. 4th ACM Conf. on Embedded Networked Sensor Systems (SenSys 2006)*, Boulder, CO, USA, November 2006.

[10] Rice, R.F. (1983) Some Practical Universal Noiseless Coding Techniques, Part III. JPL Publication 83-17, Jet Propulsion Laboratory, March 1983.

[11] Consultative Committee for Space Data Systems (1997) CCSDS 121.0-B-1: Lossless Data Compression, Blue Book, issue 1, May 1997.

[12] Weinberger, M., Seroussi, G. and Sapiro, G. (2000) The LOCO-I lossless image compression algorithm: principles and standardization into JPEG-LS. *IEEE Trans. Image Process.*, **9**, pp. 1309–1324.

[13] Kiely, A.B. (2008) Lossless compression of seismic data into fixed-length packets. *IPN Progr. Rep.*, **42–173**, pp. 1–13.

[14] Kiely, A.B., Xu, M., Song, W.-Z., Huang, R. and Shirazi, B. (2009) Adaptive linear filtering compression on realtime sensor networks. *Proceedings of The 7th Annual IEEE International Conference on Pervasive Computing and Communications (PerCom)*, Galveston, TX, March 2009.

[15] SEED Reference Manual (2007) Standard for the Exchange of Earthquake Data, SEED Format Version 2.4. Incorporated Research Institutions for Seismology.

[16] Nijim, Y.W., Stearns, S.D. and Mikhael, W.B. (1996) Lossless compression of seismic signals using differentiation. *IEEE Trans. Geosci. Remote Sens.*, **34**, pp. 52–56.

[17] Stearns, S.D., Tan, L.Z. and Magotra, N. (1993) Lossless compression of waveform data for efficient storage and transmission. *IEEE Trans. Geosci. Remote Sens.*, **31**, pp. 645–654.

[18] Nijim, Y.W., Stearns, S.D. and Mikhael, W.B. (2000) Quantitative performance evaluation of the lossless compression approach using pole-zero modeling. *IEEE Trans. Geosci. Remote Sens.*, **38**, pp. 39–43.

[19] Magotra, N., McCoy, W., Livingston, F. and Stearns, S. (1995) Lossless data compression using adaptive filters. *Proceedings of 20th International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, Detroit, MI, USA, May 1995.

[20] Ives, R.W., Magotra, N. and Stearns, S.D. (2002) Effects of multiple-pass filtering in lossless predictive compression of waveform data. *IEEE Trans. Geosci. Remote Sens.*, **40**, pp. 2448–2453.

[21] Zhang, Y. and Li, J. (2006) Efficient seismic response data storage and transmission using ARX model-based sensor data compression algorithm. *Earthq. Eng. Struct. Dyn.*, **35**, pp. 781–788.

[22] Mandyam, G., Magotra, N. and McCoy, W. (1996) Lossless seismic data compression using adaptive linear prediction. *Proceedings of IEEE International Conference on Geoscience & Remote Sensing Symposium (IGARSS 1996)*, vol. 2, Lincoln, NB, pp. 1029–1031, May, 1996.

[23] Stearns, S.D. (1995) Arithmetic coding in lossless waveform compression. *IEEE Trans. Signal Process.*, **43**, pp. 1874–1879.

[24] Pradhan, S.S., Kusuma, J. and Ramchandran, K. (2002) Distributed compression in a dense microsensor network. *IEEE Signal Process. Mag.*, **19**, pp. 51–60.

[25] Chou, J., Petrovic, D. and Ramachandran, K. (2003) A Distributed and Adaptive Signal Processing Approach to Reducing Energy Consumption in Sensor Networks. *Proc. 22nd Annu. Joint Conf. IEEE Computer and Communications Societies (INFOCOM 2003)*, San Francisco, CA, USA, April 2003.

[26] Yu, Y., Krishnamachari, B. and Prasanna, V. (2008) Data gathering with tunable compression in sensor networks. *IEEE Trans. Parallel Distrib. Syst.*, **19**, pp. 276–287.

[27] Ganesan, D., Estrin, D. and Heidemann, J. (2002) Dimensions: Why Do We Need a New Data Handling Architecture for Sensor

Networks. *Proc. ACM Workshop on Hot Topics in Networks*, Princeton, NJ, USA, October 2002.

[28] Wagner, R.S., Baraniuk, R.G., Du, Shu., Johnson, D.B. and Cohen, A. (2006) An architecture for distributed wavelet analysis and processing in sensor networks. *Proc. 5th International Conference on Information Processing in Sensor Networks (IPSN 2006)*, Nashville, TN, USA, Apr. 2006.

[29] Ciancio, A. and Ortega, A. (2005) A Distributed Wavelet Compression Algorithm for Wireless Multihop Sensor Networks Using Lifting. *Proc. 30th IEEE Int. Conf. Acoustics, Speech, and Signal Processing (ICASSP 2005)*, Philadelphia, PA, USA, March 2005.

[30] Pattem, S., Shen, G., Chen, Y., Krishnamachari, B. and Ortega, A. (2009) Senzip: an architecture for distributed en-route compression in wireless sensor networks. In: *ACM/IEEE Int. Conf. on Information Processing in Sensor Networks*, San Francisco, CA, USA, April 2009.

[31] Barr, C.K. and Asanovi, K. (2003) Energy Aware Lossless Data Compression. *Proc. 1st Int. Conf. Mobile Systems, Applications and Services (MobiSys)*, San Francisco, CA, USA, May 2003.

[32] Welch, A.T. (1984) A technique for high-performance data compression. *IEEE Comp.*, **17**, pp. 8–19, June 1984.

[33] Puthenpurayil, S., Gu, R. and Bhattacharyya, S.S. (2007) Energy-Aware Data Compression for Wireless Sensor Networks. *Proc. 32nd Int. Conf.ence on Acoustics, Speech, and Signal Processing (ICASSP 2007)*, Honolulu, HI, USA, April 2007.

[34] Huang, F. and Liang, Y. (2007) Towards Energy Optimization in Environmental Wireless Sensor Networks for Lossless and Reliable Data Gathering. *Proc. IEEE Int. Conf. Mobile Ad hoc and Sensor Systems (MASS 2007)*, Pisa, Italy, October 2007.

[35] Alippi, C., Camplani, R. and Galperti, C. (2007) Lossless Compression Techniques in Wireless Sensor Networks: Monitoring Microacoustic Emissions. *Proc. 2007 International Workshop on Robotic and Sensors Environments (ROSE 2007)*, Ottawa, Canada, October 2007.

[36] Gersho, A. (1984) Adaptive filtering with binary reinforcement. *IEEE Trans. Inform. Theory*, **30**, pp. 191–199.

[37] Golomb, W.S. (1996) Run-length encodings. *IEEE Trans. Inform. Theory*, **12**, pp. 399–401.

[38] Kiely, A.B. (2004) Selecting the Golomb parameter in Rice coding. *IPN Progr. Rep.*, **42–159**, pp. 1–18.