A Data Compression Application for Wireless Sensor Networks Using LTC Algorithm

Renu Sharma

Indiana University, Purdue University

Indianapolis, IN

rensharm@iupui.edu

Abstract **- In this paper the author represents energy efficient data compression application based on LTC (Lightweight Temporal Compression) algorithm in wireless sensor networks (WSNs). WSNs are essentially constrained by motes' limited battery power and networks bandwidth. The author focuses on data compression algorithm which effectively supports data compression for data gathering in WSNs. Data reduction before transmission such as by compression will significantly decrease the resource usage. Therefore, the main idea of this paper is to show how a data compression application such as collection tree protocol (CTP) is used for data collection from different sensor nodes into the root node in order to increase the network lifetime. LTC algorithm is used to minimize the amount of error in each reading. In the context of the use of wireless sensor network technology for environmental monitoring the two main elementary activities of wireless sensor network are data acquisition and transmission. However, transmitting/receiving data are power consuming tasks in order to reduce transmission associated power consumption; we explore data compression by processing information locally. Since, the inception of sensor networks in-network processing has been touted as enabling technology for long-lived deployments. Radio communication is the overriding consumer of energy in such networks. Data reduction before transmission either by compression or feature extraction will directly & significantly increase network lifetime. In many applications where all data must transport out of networks data may be compressed before transporting. The chosen compression technique can operate under stringent resource constraints of low-power nodes and induces tolerable errors. This paper evaluates temporal compression scheme designed especially for mica motes. By using LTC it is possible to compress data up to -20 to -1. Furthermore this algorithm is simple and requires little storage as compared to other compression techniques. The proposed application is implemented on the tinyOS platform using the nesC programming language. The author conducts simulation via TOSSIM and real-world testbed FlockLab to evaluate their work. The result demonstrates the significance of the application.**

Index Terms **- Lightweight Temporal Compression (LTC), Wireless Sensor Networks (WSN), Collection Tree Protocol (CTP).**

I. INTRODUCTION

Wireless sensor networks (WSNs) are extremely important for continuous monitoring in fields like environmental science, water resources, ecosystems, and structural health and health-care applications. In such applications, a large amount of observation data in a monitoring sensornet needs to be transferred to data sink for analyses. One of the greatest challenges in the construction of large scale wireless sensor networks (WSNs) with practical applicability is the development of a mechanism that allows the network to operate for prolonged periods of time relying solely on the limited amount of energy that can be stored in or harvested by wireless sensor nodes. Data communication is the main factor which is responsible for draining the energy reserves of the network. Techniques to reduce the amount of information transmitted by sensor nodes are of great interest. One effective approach to reduce data communication in the network is to compress the information locally before it is transmitted.

When they know which feature in a dataset will be of interest, network processing algorithms can be tuned to focus on such features and it disregards the extraneous information. When it is not known what will be of interest, data reduction can be achieved by using either lossless compression techniques or by lossy compression techniques. The lossless compression technique reduces data representations slightly, but preserves all data precisely. The lossy compression technique that represents data approximately and thus can reduce data. Scientists do not know which features will be interesting therefore they are trying to collect as much of the original dataset with the least amount of information lost. A number of collection protocols have been proposed in the area of WSNs, including collection tree protocol (CTP) [1], Flush, Fetch, Wisden and Fusion. These protocols focused on reliable data transport in WSNs to address wireless link dynamics, rate and congestion control but none of them considered a data compression approach.

Although data compression is a well-established research area, despite the extraordinary advances in the computational capability of embedded devices, most existing algorithms still cannot be directly ported to wireless sensor nodes because of the limited hardware resources available particularly program and data memory. Even though many of the time-honored compression algorithms could be executed in modern wireless sensor nodes, they would leave few resources available for the nodes to carry out other tasks such as sensing and communication. More importantly, these nodes would have significantly fewer opportunities to enter sleep modes and attain the energy efficiency that motivated the use of a compression algorithm in the first place. Therefore, a number of data compression methods specifically designed for WSNs have been proposed in the past few years. What many of these methods have in common is the fact that they make use of the correlation of the data acquired by the sensor nodes in order to achieve high compression ratios while employing computationally inexpensive algorithms.

This paper proposes a new and computationally simple compression technique developed for the context of habitat monitoring that obtains up to -20 to -1 reduction in the amount of environmental data that needs to be transmitted on certain data sets. The proposed technique called Lightweight Temporal Compression (LTC) [2] introduces a small amount of error into each reading, and is bound by a control knob. LTC has been implemented and integrated into the query engine component of the Extensible Sensing System (ESS) application.

The rest of the paper is organized as follows. In section II we describe the relevant work. The deployment approach is explained in section III. In section IV we describe the implementation with the use of our LTC algorithm. Section V explains the analysis of the work. The conclusion and future work is described in section VI.

II. RELATED WORK

In the literature on compression methods for WSNs both lossy and lossless approaches that exploit the high temporal correlation of the sensor node data can be found [4]. In a variation of the run length encoding (RLE) method for data compression in WSN known as K-RLE approximates a string of measurements with values in the range as the pair, which defines the precision of the method [4]. Although lossy compression methods can generally achieve high compression ratios at the expense of moderate accuracy losses. In many WSN applications it may not be clear before data collection how much information can be disregarded without compromising the overall purpose of the system. Event-based communication approaches attempt to resolve this problem by limiting the transmission of sensor data to respond to user queries [5]. However, in many cases, the user may not be able to formulate queries without observing the raw sensor data beforehand. As a consequence, a number of lossless compression methods for WSNs have been proposed. S-LZW [6] is an adaptation of the celebrated Lempel-Ziv-Welch (LZW) algorithm [7] for resource-constrained wireless sensor nodes. The first major work in sensor network data processing is Directed Diffusion [3]. Diffusion provides framework for in-network processing. Wavelet compression has been proposed for the use in sensor networks by Ganesan et al. in Dimensions [7, 8, 9]. Wavelet compression is a good match for sensor networks. Two popular and successful text compression schemes, namely BZIP2 based on Burrow Wheelers Transform (BWT) [14] and GZIP based on LempelZiv (LZ) family were designed to be used by PC grade devices for the compression of data files. Lossless Entropy Compression [LEC] computes the differences of consecutive sensor measurements and divides them into groups whose sizes increase exponentially. Each group corresponds to the number of bits required to represent the measurement differences. These groups are then entropy coded using a fixed compression table based on the baseline JPEG algorithm to compress the DC coefficients of an image. The compressed symbols are formed by concatenating the group number and the index of the element within the group. The authors reported high compression ratios for actual environmental data collected by WSNs.

In Adaptive Linear Filtering Compression (ALFC) an adaptive linear filter is used to predict the future samples of the dataset and the prediction errors are compressed using an entropy encoder. In order to account for the limited computational capabilities of wireless

sensor nodes, the method employs a quantization mechanism. Adaptive prediction avoids the requirement of defining the filtering coefficients while still allowing the system to adjust to dynamic changes in the source. The authors showed that ALFC achieves higher compression ratios than previous methods while requiring significantly fewer hardware resources.

III. METHOD

A. System Model

Wireless sensor networks (WSNs) modelled by graphs G= (V, E) [1] which consists of set of nodes V and set of edges ECV^2. Here V represents sensor nodes and edges in E represent wireless links among the nodes. The sensor nodes are battery-operated whereas sinks are power limited. Transmitting and receiving in the sensor nodes are the most energy-consuming operations. Studies have shown that approximately 3000 instructions could be executed for the same energy cost as sending a bit for 100 m by radio. In general, receiving has comparable energy cost as transmitting. Existing WSN lossless and lossy compression algorithms follow different principles and thus none of those algorithms applied to both lossless and lossy compression. For example, recent lossy compression algorithms such as LTC and PLAMiS are based on piecewise linear approximation, and would result in more compressed bits than the raw data bits when applied to lossless compression.

B. Extensible Sensing System

Extensible Sensing System (ESS) [2] is under development in the San Jacinto Mountains. The project is basically designed to provide spatially dense environmental, physiological, and ecological information to scientists. ESS mainly focuses on monitoring microclimate and other physical characteristics of plants and animal habitats, encompassing below ground root observation and sensing, water movements through soils near roots, hydration status of lichens. In ESS motes are connected to the weather sensing boards. ESS architecture consists of three components: a sampler, a routing in-network processing framework and a query processor. The sampler coordinates the sampling request from query engine and tasks the appropriate sensor drivers to collect data. Diffusion's one-phase-pull protocol [2] is used by ESS to transport across motes to and from microservers. Query processing provides data in which most of the ecologists are interested. Currently query processor supports five query types. As new query types are developed, the query processor will be extended to support them. In order to increase node lifetime, lightweight temporal compression has been added to the query processor.

Compression scheme is very important in Wireless sensor networks. Compression does not fix the lack of scalability of routing tree but by reducing aggregate traffic by a constant factor can minimize the problem significantly. When applications store the data locally, compression is used to reduce the data representation so more data can be stored. Compression increase network lifetime. Since the ratio is the dominant consumer of energy in a sensor node, saving due to compression directly translates into a lifetime extension for network nodes.

IV. IMPLEMENTATION

A. Collection Tree Protocol

The Collection Tree Protocol (CTP) provides for best-effort anycast datagram communication to one of the collection roots in a network. CTP is widely regarded as a reference protocol for performing data collection in Wireless Sensor Networks and its specification is provided in TinyOS1 Enhancement Proposal 123 [8]. Gnawali et al. also report a throughout description and performance evaluation of CTP in realistic settings, demonstrating the ability of the protocol to reliably and efficiently report data to a central collector [9, 10]. A TinyOS implementation of CTP is available within the TinyOS 2.1 distribution and therefore directly usable for implementing WSNs applications. CTP is designed based on Carrier Sense Multiple Access – Collision Avoidance (CSMA – CA) which not suitable for industrial applications.

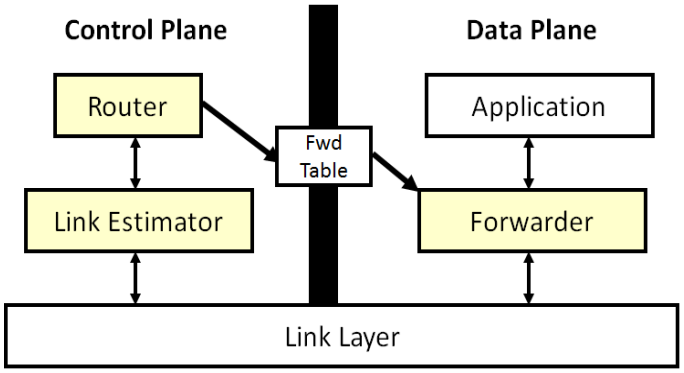


Figure 1: Common Architecture

The main task of collection protocol is to construct a tree and maintain it. For this purpose CTP uses beacons or routing messages to initialize and maintain its routing table. There are three main components in implementation of CTP and they are:

1. Routing Engine
2. Forwarding Engine
3. Link Estimator

Details about these components are discussed below:

1) Routing Engine

The Routing Engine, an instance of which runs on each node, takes care of sending and receiving beacons as well as creating and updating the routing table. This table holds a list of neighbors from which the node can select its parent in the routing tree. The table is filled using the information extracted from the beacons. Along with the identifier of the neighboring nodes, the routing table holds further information, like a metric indicating the "quality" of a node as a potential parent.

In the case of CTP, this metric is the ETX (Expected Transmissions), which is communicate by a node to its neighbors through beacons exchange. A node having an ETX equal to n is (expected to be) able to deliver a data packet to the sink with a total of n transmissions. The ETX of a node is defined as the "ETX of its parent plus the ETX of its link to its parent" [8]. More precisely, a node first computes, for each of its neighbors, the link quality of the current node-neighbor link. This metric, to which we refer to as the 1-hop ETX, or ET X1hop, is computed by the LE. For each of its neighbors the node then sums up

the 1-hop ETX with the ETX the corresponding neighbors had declared in their routing beacons. The result of this sum is the metric which we call the multi-hop ETX, or ETXmhop. Since the ETXmhop of a neighbor quantifies the expected number of transmissions required to deliver a packet to a sink using that neighbor as a relay, the node clearly selects the neighbor corresponding to the lowest ETXmhop as its parent. The value of this ETXmhop is then included by the node in its own beacons so as to enable lower level nodes to compute their own ETXmhop. Clearly, the ETXmhop of a sink node is always 0.
The frequency at which CTP beacons are sent is set by the beacon interface. The implementation is available in the CTP libraries that are provided by tinyOS.

2) Forwarding Engine

The Forwarding Engine, as the name says, takes care of forwarding data packets which may either come from the application layer of the same node or from neighboring nodes. As we will detail in section 3.4, the FE is also responsible of detecting and repairing routing loops as well as suppressing duplicate packets. As mentioned above, the ability of detecting and repairing routing loops and the handling of duplicate packets are two of the tree features TinyOS TEP 119 requires to be part of a collection protocol. The third one, i.e., a mean to estimate the 1-hop link quality, is handled in CTP by the Link Estimator.

3) Link Estimator

The Link Estimator takes care of determining the inbound and outbound quality of 1-hop communication links. As mentioned before, we refer to the metric that expresses the quality of such links as the 1-hop ETX. The LE computes the 1-hop ETX by collecting statistics over the number of beacons received and the number of successfully transmitted data packets. From these statistics, the LE computes the inbound metric as the expected number of transmission attempts required by the neighbor to successfully deliver a beacon. Similarly, the outbound metric represents the expected number of transmission attempts required by the node to successfully deliver a data packet to its neighbor.

B. Data Compression

Data compression involves encoding information using fewer bits than original representation. Compression can be performed by lossy compression scheme or lossless compression scheme. Lossless compression reduces bits by identifying and eliminating statistical redundancy. There is no information lost in lossless compression. On the other hand, lossy compression reduces bits by identifying unnecessary information and removing it. The process of reducing the size of a data file is called as data compression. Compression is very useful because it helps to reduce resource usage like data storage space or transmission capacity. Data compression involves trade-offs among various factors, including the degree of compression, the amount of distortion introduced, and the computational resources required to compress and uncompressed data. Lossy data compression is the converse of lossless data compression. In this scheme, some loss of information is acceptable. There is corresponding trade-off between preserving information and reducing the size. Fig2 presents data compression, in which sensor data is compressed with the help of energy efficient compression algorithm. The compressed data then transmitted to the decompressor. The output of decompressor is nothing but the reconstructed data.

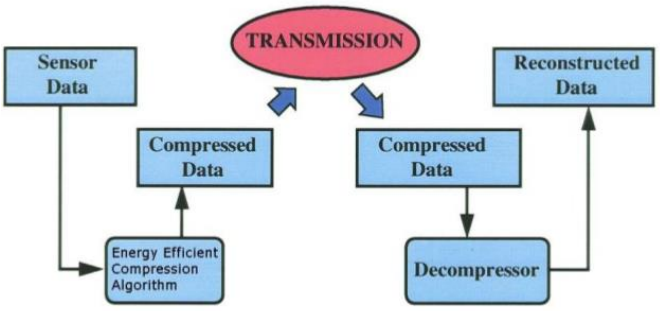


Figure 2: Data Compression [14]

C. Algorithm

This section describes the factors like sensor noise and environmental data which motivates LTC. Later part shows performance of LTC and shows how it is feasible for motes.

1) Sensor Noise and Environmental Data

When sensor is sampling it will produce a range of readings due to noise. Sensors manufacturers specify sensors operating range as well as accuracy. Lightweight temporal compression (LTC) [2] is designed to compress data when sensor accuracy is expressed in terms of margin, and the probability distribution of error is either uniform or unknown. LTC represent a sample by value within the margin that is used to maximize compression. The drawback of LTC is that it may convolute the original error distribution when that distribution is not uniform. When one can characterize data with the model, best compression technique is simply to return the parameters to that model that best fit the dataset. Environmental data such as temperature and humidity have nice property that they are usually continuous in the temporal dimension and at small enough time windows are approximately linear. Environmental phenomena are inherently very complex and difficult to model. We don't assume any model for environmental data. Outliers and changes in slope are the important environmental features. We guarantee a match on a point-by-point basis, ensuring that any reconstructed data point is no more than some error margin from its sample point. Linear least squares, which uses the residual squared as a quality metric, does not preserve outliers LTC performs well even the margin is set to be much less than the manufacturer's specified error margin.

2) Lightweight Temporal Compression Algorithm

In LTC, leverages temporal linearity [12] is used to compress data. The LTC algorithm is depicted in Figure 3. The x-axis represents time and y-axis represents value. The first point, which we will call z, is unchanged and becomes the first endpoint of the line segment approximation. The second sample is transformed into a vertical line segment, m, stretching a distance e above and below the measured sample, as shown in 3a.The length e is the desired error margin for each sample, and the vertical segment has length 2e. An upper and lower bound on all possible lines that can represent z and m accurately are drawn in black dotted lines. When the next sample is taken and transformed into a vertical segment, n, this set of lines is trimmed in order to represent it, as shown in 3b. If the upper line, highLine, is above the upper limit of n, it is lowered. Likewise, if the lower line, lowLine, is below the lower limit of n, it is raised.

Figure 3c shows a measurement whose lower bound is above highLine, indicating that no

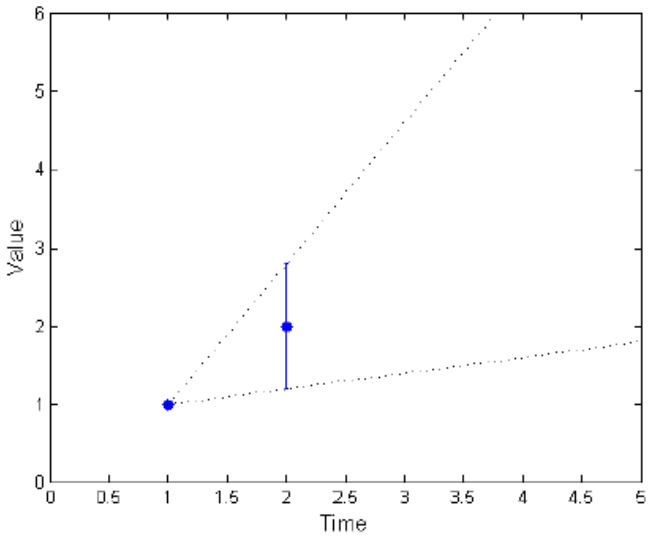
line segment can accurately represent it. Once this occurs, the data is capped off. All data before the current sample can be represented accurately by z and the midpoint between the current highLine and lowLine. This midpoint is shown as the third green x in 3c. Only z is transmitted to the FC, and the midpoint becomes the starting point for the next line segment. HighLine and lowLine connect this midpoint to the upper and lower extremes of the out-of-bounds measurement.

LL – Lower Limit

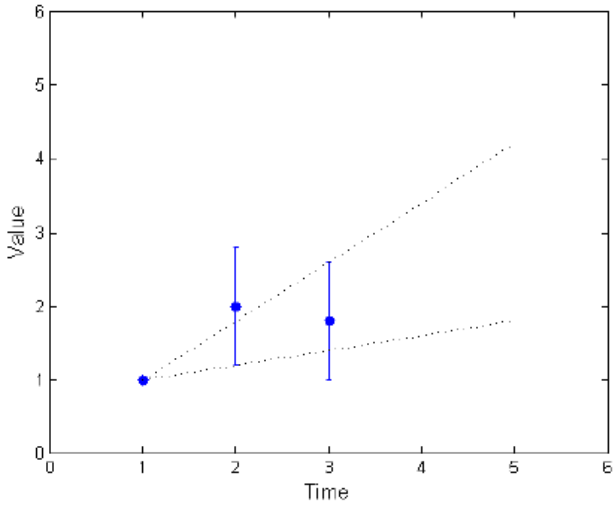
UL – Upper Limit

ul – Highest point of segment

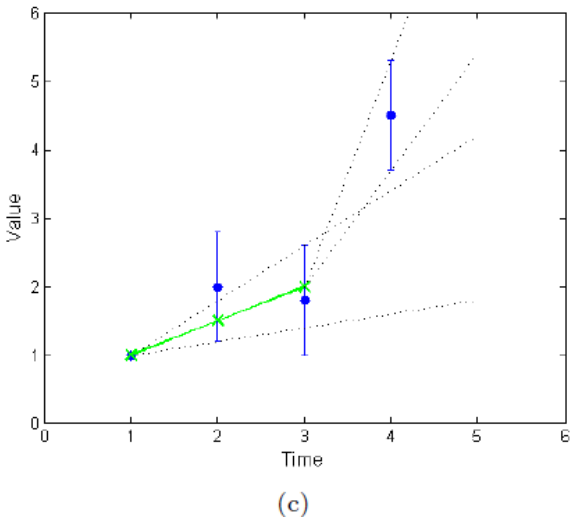
ll – Lowest point of segment



(a)



(b)



(c)

Figure 3: Original LTC algorithm

LTC algorithm is as follows.

1. Initialization: Get the first data point, store into z. Get next data point (t2, v2), use it to initialize limits UL (UL is set to (t2, v2 + e)) and LL (LL is set to (t2, v2 − e).
2. Calculate the highLine to be the line connecting z and UL.
3. Calculate the lowLine to be the line connecting z and LL.
4. Get next data point. Transform the point to a vertical segment using the margin e. Let ul be the highest point of the segment. Let ll be the lowest point of the segment.
5. If highLine is below the ll or if the lowLine is above the ul then go to 9, else continue onto the next step
6. If highLine goes above ul then set UL to be ul.
7. If lowLine goes below ll then set LL to be ll.
8. Go to 2.
9. Cap off: output z to the output data stream.
10. Set z to be the point midway between UL and LL.
11. Set UL to be ul.
12. Set LL to be ll.
13. Go to 2

In the LTC algorithm, proper recovery of the time series depends not only on the current transmission of z but also on correct reception of the previous z. If a single packet is lost, two line segments representing the data cannot be recovered.

3) Compression Ratio

The compression ratio is very important metric for compression algorithms. Compression ratio is only one of the factors which determine the choice of a compression algorithm suited to WSNs. The performance of compression algorithm is totally depends on the compression ratio. In WSNs, having higher compression ratios means lesser amount of data to be transmitted. This means more energy savings. The compression ratio is calculated by following formula [11, 13]

$$Compression\ ratio = 1 - \frac{Compressed\ Size}{Original\ Size}$$

Where compressed size and original size are respectively, the sizes in the bits of the compressed and uncompressed bitstreams.

V. SIMULATION AND ANALYSIS

We have performed simulations to thoroughly evaluate our application. This section describes the simulators used in the experiment. We can use either TOSSIM or FlockLab testbed as a simulator. TOSSIM is used to simulate this application.

A. TOSSIM

TOSSIM is event simulator for TinyOS sensor networks. Instead of compiling a TinyOS application for a mote, users can compile it into the TOSSIM framework, which runs on a PC. This allows users to debug, test and analyze algorithms in a controlled and repeatable environment. As TOSSIM runs on a PC, users can examine their TinyOS code using debuggers and other development tools. TOSSIM is automatically built when you compile an application. The main goal of TOSSIM is to provide scalable and accurate simulation of TinyOS sensor network. TOSSIM provides configuration of debugging output at run-time. TOSSIM is nothing but a library. TOSSIM mainly consist of two programming interfaces i.e. C++ and python. Python behaves like a debugger. It allows us to interact with running simulation.TOSSIM is a discrete event simulator which has been designed to simulate sensor networks that use the TinyOS operating system. The main aim of TOSSIM is to provide a high fidelity simulation of TinyOS applications. TinyOS, its libraries are written in the nesC language. TinyOS is an operating system that's designed to manage the operations of a variety of mote devices as well as the sensors attached to them. TinyOS also provides a networking stack to allow the motes to form an ad-hoc network.

Since TinyOS has been built using nesC it is built using components and interfaces. This results in a lot of flexibility to select alternative components such as different networking modules that implement different protocols. The TinyOS executable consists of two threads. One thread is used to execute tasks and the other thread is used to execute the hardware event handlers.

B. Analysis

LTC is very useful compression algorithm. It is possible to sample at high rate with the help of this compression scheme. Even if we send the small amount of data, sampling at high rate is possible. Sampling at high data rate is useful to detect any environmental changes. This algorithm does not guarantee that the number of line segments used to represent a data set is minimal. An optimal algorithm need to store the entire data set before a minimal set of lines could be calculated. Most of the gain of LTC occurs well within the operating margin of error specified by the sensor manufacturers. The maximum margin of error between a raw data point and its corresponding post-compression reconstructed data point can be set arbitrarily for LTC. This serves as tuning knob to adjust tradeoff between compressed data size and accuracy. As accuracy decrease, high resolution components of a dataset may be lost.

VI. CONCLUSION & FUTURE WORK

In this paper, tunable lightweight temporal compression scheme (LTC) is proposed. This algorithm is simple and requires little storage as compared to other compression techniques. By using LTC, it is possible to compress data up to -20 to -1. It is possible to sample at high rate with the help of this compression scheme. The LTC algorithm is designed basically for mica motes with 8-bit processor, which has no hardware to handle floating point values. This limits the applications of LTC to compression of integer data only. Lossy compression schemes like LTC reduce bits by identifying unnecessary information and removing it. Data reduction before transmission such as by compression will significantly decrease the resource usage and increase network lifetime.

Long term goal is to check LTC'S ability to filter noise. It will be useful to perform feature extraction on motes. In addition, we plan to explore variations of LTC to address data streams with a low sampling rate

References

[1] Liang Y. and Erratt N.: 'Compressed data - stream protocol: an energy efficient compressed data - stream protocol for wireless sensor networks', IET Communication, 2011, pp. 2673-2683.

[2] Schoellhammer T., Greenstein B., Osterweil E., Wimbrow M., Estrin, D.: 'Lightweight Temporal Compression of Microclimate Datasets', Local Computer Networks, 2004. 29th Annual IEEE International Conference, 2004, pp. 515 - 426

[3] Henry Ponti Medeiros, Marcos Costa Maciel, Richard Demo Souza, and Marcelo Eduardo Pellenz, Research Article - "Lightweight Data Compression in Wireless Sensor Networks Using Huffman Coding" International Journal of Distributed Sensor Networks Volume 2014 (2014), Article ID 672921, 11 pages.

[4] E. P. Capo-Chichi, H. Guyennet, and J.-M. Friedt, "K-RLE: a new data compression algorithm for wireless sensor network," in Proceedings of the 3rd International Conference on Sensor Technologies and Applications (SENSORCOMM '09), pp. 502–507, Athens, Greece, June 2009.

[5] K. Romer, "Discovery of frequent distributed event patterns in sensor networks," in Proceedings of the 5th European Conference on Wireless Sensor Networks (EWSN '08), pp. 106–124, Bologna, Italy, 2008.

[6] C. M. Sadler and M. Martonosi, "Data compression algorithms for energy-constrained devices in delay tolerant networks," in Proceedings of the 4th International Conference on Embedded Networked Sensor Systems (SenSys '06), pp. 265–278, ACM, November 2006.

[7] T. A. Welch, "A technique for high-performance data compression," Computer, vol. 17, no. 6, pp. 8–19, 1984.

[8] Rodrigo Fonseca, Omprakash Gnawali, Kyle Jamieson, Sukun Kim, Philip Levis, and Alec Woo. TinyOS Enhancement Proposal (TEP) 123: The Collection Tree Protocol(CTP).www.tinyos.net/tinyos-2.x/doc/pdf/tep123.pdf.

[9] Omprakash Gnawali, Rodrigo Fonseca, Kyle Jamieson, and Philip Levis. CTP: Robust and Efficient Collection through Control and Data Plane Integration. Technical report, The Stanford Information Networks Group (SING), 2008. http://sing.stanford.edu/ pubs/sing-08-02.pdf.

[10] Omprakash Gnawali, Rodrigo Fonseca, Kyle Jamieson, David Moss, and Philip Levis. Collection Tree Protocol. In Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems (SenSys 2009), Berkeley, CA, USA, November 2009.

[11] Maher El Assi, Alia Ghaddar, Samar Tawbi, and Ghaddar Fadi "Resource efficient floating-point data compression using MAS in WSN" International journal of A d hoc, sensor & ubiquitous computing(IJASUC) vol.4. No5, October 2013.

[12] Daniel john parker "Exploiting Temporal and Spatial Correlation in Wireless Sensor Networks". April 2013.

[13] Francesco Marcelloni and Massimo Vecchio "An efficient lossless compression algorithm for tiny nodes of monitoring wireless sensor networks" The computer journal advance access published April 30, 2009.

[14] Prakash Mallavalli "Data compression in wireless sensor networks"