An Evaluation of Compression Schemes for Wireless Networks

Kirsten Dolfus and Torsten Braun
Institute for Computer Science and Applied Mathematics
University of Bern
Bern, Switzerland
Email: dolfus, braun@iam.unibe.ch

***Abstract*—Data gathering, either for event recognition or for monitoring applications is the primary intention for sensor network deployments. In many cases, data is acquired periodically and autonomously, and simply logged onto secondary storage (e.g. flash memory) either for delayed offline analysis or for on demand burst transfer. Moreover, operational data such as connectivity information, node and network state is typically kept as well. Naturally, measurement and/or connectivity logging comes at a cost. Space for doing so is limited.**

Finding a good representative model for the data and providing clever coding of information, thus data compression, may be a means to use the available space to its best. In this paper, we explore the design space for data compression for wireless sensor and mesh networks by profiling common, publicly available algorithms. Several goals such as a low overhead in terms of utilized memory and compression time as well as a decent compression ratio have to be well balanced in order to find a simple, yet effective compression scheme.

I. Introduction

The availability of wireless networks in industrialized countries for accessing personal and business data anytime and anywhere is nowadays almost taken for granted. Furthermore, with wireless sensor networks (WSNs) becoming more and more popular for data acquisition close to a source of interest, the spread of wireless technology even in areas far away from typical hotspots has started. From a network operators' perspective, however, a well-functioning network has to be monitored and maintained accurately in order to provide the expected service. A standard technique to verify correct operation or track exceptional network states is for all participating nodes to periodically log status information, which may then be analyzed offline if necessary. As a consequence, storage for such log data has to be provided in the first place on each networked node. Dependent on the node class, thus especially when looking at embedded hardware, storage space may however be very limited. Even worse, data transfer from battery-powered devices is expensive in terms of energy needed for transmission, directly influencing network lifetime, which restricts the usage of logs to only the bare minimum.

The problem of storage shortage is not new and commonly addressed by applying a compression algorithm to lower the amount of data to be stored. Hence, compression can be seen as a technique to expand on-site storage facilities. There is though a second advantage data compression provides when utilized in networked application scenarios: As the size of data to be transferred is limited, not only the direct energy expenditure for transmission and reception for each link is decreased, but this effect nicely accumulates if it is sent on a multi-hop path. Furthermore, compression decreases the overall network load which once again impacts the probability for collisions on the wireless medium and hence quality of service parameters such as transmission latency.

It has to be evaluated carefully, whether there is an actual gain from an energy perspective [1]. Our motivation to look into compression in the first place has been that it is a great use case for so called self-* frameworks, which autonomously monitor a system and decide on appropriate reactions in case of (anticipated) error states. Such a framework has been built by extending and applying the FACTS middleware [2] for WSNs. It allows to specify rules given predefined system state, e.g., it may trigger the application of compression to data logs in case the system is running short in storage and it may schedule its execution to not collide with higher priority tasks. The actual tradeoffs that have to be considered in this decision process are part of future research.

This study is concerned with evaluating standard compression algorithms that are suitable to run on constraint networking nodes. Information encoding is supposed to be available as a concurrent, application-independent feature, not as a primary goal itself. Therefore, consumed resources such as CPU cycles, energy, memory in terms of RAM and ROM should be as low as possible while the actual gain in terms of compression ratio should be reasonable. Since especially in WSNs data precision is of utmost importance, only lossless algorithms have been evaluated.

II. Related Work

In general, compression is concerned with encoding information in fewer bits than in the original encoding scheme. This requires a two phase process, where in a first phase, a good model for probabilities for certain input has to be found and in a second, these probabilities have to be encoded. Models may range from statistical models on progression of input values to a simple identification of repeated patterns, thus redundancy, in input values. Input values with high probabilities, thus low entropy, will then be encoded with a smaller number of bits than information with high entropy, thus low probability.

Common coder components are most of the times very generic and based on Huffman or arithmetic codes.

Approaches to decrease the amount of data to be actually sent from or stored within a wireless device have been extensively studied on different layers of the protocol stack, see [3] for a good overview. In this area, compression schemes may not only be distinguished into the well-known categories of lossless and lossy algorithms, but also whether compression is implicitly or explicitly addressed, performed on a single node or in a networked manner.

Implicit compression has been introduced by Deshpande et. al. [4] with their model-driven approach to data acquisition in wireless sensor networks. Since sensor readings are subject to spatio-temporal redundancy, a model of common value evolution is built in advance and deployed on the sensor nodes as well as on a sink node. Only in case measured values do not conform to the model, nodes send their values towards the sink, otherwise the value is suppressed.

Explicit compression is however the common case and also the focus of this article. In the wireless networking context, algorithms taking in advantage of spatio-temporal properties of input data have in particular been extensively studied in the past: A multitude of algorithms, e.g. [5], [6], [7] explored the integration of aggregation techniques within the routing layer, or described and evaluated concepts to benefit from a small value dispersion of measurements [8], [9]. Redundancy, as caused by repetition e.g. found in log files for networking logs, is naturally another source of overhead that may be eliminated for improved storage capacity [10], [11] or to minimize energy spent for data transfer [12]. Here, transformation techniques for input values by recognition of periodicity of input files have proven to have a positive effect on the overall compression ratio.

A different approach has been explored by Marcelloni et al. [13]: In order to keep the algorithm as simple as possible and to avoid complex computations on embedded nodes, their solution relies on a two-phase coding process based on a lookup table of the size of the analog-digital converter and compresses the raw bits of a sensor reading. Here, a codeword is a hybrid of unary and binary codes supplied by an adequate dictionary similar to the one used for DC coefficient coding in JPEG compression. Since the size of the dictionary is fixed and encoding is done via mapping, the algorithm is well suited for on-the-fly compression. However, the obtainable compression ratio is highly dependent on a good mapping strategy.

All of the algorithms described above have been specifically designed for wireless (sensor) networks. While this tailoring can be a source of great optimization in case exactly the type of input data is matched, general-purpose implementations of compression techniques, such as huffman encoding, arithmetic coding [14], or Lempel-Ziv-Welsh [15] variations allow for a more flexible usage. However, due to their size, the majority of the compression tools nowadays used on standalone machines are not simply transferable to possibly embedded hardware [16]. Algorithm choice, therefore, needs to include a validation of applicability.

III. Profiling of existing algorithms

Data compression may be perceived as a utility to be used on demand: As long as performance goals are met and critical resources are not wasted, the choice of a suitable algorithm is arbitrary. While algorithms specifically tailored for a certain data type such as sensor measurements may be advantageous when tight constraints have to be respected, general-purpose compression schemes enable a quicker adoption due their holistic implementations. Accordingly, the selection of a specific algorithm is highly dependent on the target area and on the degree of freedom in terms of resources such as time for pre-deployment performance profiling, execution time at runtime, memory and energy consumption of utilized compression scheme.

In this section, we will evaluate and compare a number of publicly available algorithms. All of them, quickly introduced in the following part, have been applied to different data sets obtained in real world deployments, namely the CTI-Mesh [17] wireless mesh network experiments and wireless sensor network data gathered with the help of the TARWIS platform [18].

A. Algorithms

In this study, we focus on lossless, stand-alone (thus non-distributed) compression schemes to lower the amount of data stored within a network node or transmitted across the network. The performance of an algorithm is to a non-negligible part based on the quality of its implementation. Fair comparison is difficult when relying on freely available source code. In order to minimize the impact of the implementation strategy, we opted for source code written in the same language (C), and, where ever possible, by the same author. Hence, the standard implementations of huffman, arithmetic and LZW coding are available from [14]. MiniLZO, an implementation of the LZO library which at its core is a LZ77 variant is accessible at [19], while S-LZW, an LZW variant explicitly designed for sensor measurements, corresponds to the remarks in [12] and may be downladed from the author's homepage.The benchmark suite that has been evaluated on the available data sets comprises the following algorithms:

- **Huffman coding (huff):** Static huffman coding.
- **Adaptive huffman coding (ahuff):** Huffman coding using an adaptive symbol table.
- **Arithmetic coding (arith-0):** Simple arithmetic coding, no preceding symbols influence probability range of a symbol.
- **Arithmetic coding (arith-1):** Arithmetic coding order 1, thus 1 previous symbol influences encoding of a symbol.
- **Arithmetic coding (arith-1e):** See above, only escape characters are also regarded during encoding.
- **Arithmetic coding (arith-n):** Order n arithmetic coding.
- **LZSS (lzss):** Modified LZ77 version with 12-bit sliding window.
- **LZW (lzw12):** LZW implementation with 12-bit symbols.

- **LZW (lzw15v):** LZW with variable-sized symbols up to 15-bit.
- **MLZO (mlzo):** MiniLZO, lightweight implementation of LZO.
- **SLZW (slzw):** LZW variant for sensor networks with specific transformations of the input data and a small cache.

B. Data

This study is explicitly targeting wireless networks, hence an evaluation of data from real-world traces is mandatory. In general, logfiles feature a repetitive structure: Always when necessary, and this may either be in case of a pre-specified incident (upon error/anomality detection or threshold-based) or simply in a periodic manner, data on network characteristics of interest is stored in a logfile for later on offline analysis. Since each write operation to a logfile features the same attributes and the probability for recurrence of values is high for certain types of attributes, logfiles are good candidates for the usage of compression algorithms. It is, however, noteworthy that among different types and application scenarios for the installation of networking nodes, the actual data that is logged naturally differs: Mature IP networks will most likely log connectivity data denoting link quality, information on routing tables and network properties to derive information on quality of service parameters (e.g. latencies and bandwidth). Sensor networks in contrast typically neither have the storage/communication capacity for extensive QoS considerations nor is it in the focus or interest of their deployment. The amount of data for mirroring communication QoS in WSN deployments will, therefore, be limited to a minimum, e.g., to reconstruct network topologies. Rather, logging will concentrate on sensor measurements, this being the primary intention for sensor network deployment in the first place. Both types of logfiles, thus traces from wireless IP-based networks and those from a sensor network, have been available for evaluation.

1) Wireless Mesh Network Data: The first type of files has been acquired from a feasibility study on a 5GHz outdoor wireless mesh network (WMN) using directional antennas [17]. This network has been set up to prove the suitability of WMNs to function as access networks for remote locations such as weather stations to a fiber network. The WMN consisted of six mesh nodes deployed in the area of Neuchâtel, Switzerland and operated for several months. Specifically, the traces feature the following data points (recorded every ten minutes):

- **Fping measurements (fping):** All nodes issued fping commands to all other nodes of the network with latencies being recorded.
- **Host and network association data (hna):** HNA data is specific for the OLSR routing protocol and reflects a node's capability to function as a gateway to an external network with associated netmasks and gateway addresses denoted in the logfile.
- **QoS measurements (qos):** In this data set quality of service parameters including link quality measurements and ETX values (expected transmission count) are present.
- **Link statistics (stats):** Specific properties of links to other nodes, including channel information on available data rate and received signal strength (RSSI).
- **Multiple interface detection (mid):** IP addresses and aliases for connected nodes are denoted in this data set.
- **Routing tables (rt):** These logs enable to reconstruction of routing paths as OLSR routing tables are collected.
- **Topology characteristics (topo):** This set features the link qualities and costs for connections to other nodes in the network.
- **Traceroute information (trace):** Information gathered after issuing the traceroute command to learn about routing paths and latencies.

The evaluation in Section III-C uses the data collected by one node for one month. The results are based on the average performance of the algorithms on these logfiles.

2) Wireless Sensor Network Data: Sensor network data has been drawn from a week-long periodic sampling of all twenty nodes of the TARWIS platform [18] installed at the University of Bern. This testbed consists TMote Sky sensor nodes [20], set up in the different rooms of the institute of computer science on campus. The data sets are made up of the following data points and measurements, which have been taken every five minutes:

- **Timestamps:** Each entry is tagged with a timestamp in order to allow for correlation of data of different nodes if necessary.
- **Light measurements:** Photosynthetical active radiation (PAR) and thermophile shadowband radiation (TSR) are acquired and denoted in Lux.
- **Temperature and humidity measurements:** All nodes capture the current temperature in degree Celsius and the relative humidity of their surroundings.
- **RSSI measurements and neighborhood list:** Via periodic broadcasts, neighborhood lists, thus direct links to other sensor nodes, are populated and data on the received signal strength is collected.

Once again, the upcoming results represent the average performance of the algorithm for the different key properties. The data set of all nodes have been used for evaluation to grant a suitable evaluation base.

C. Evaluation

Profiling different, off-the-shelf, publically available compression algorithms to gain insights in how far they are suitable for usage on wireless mesh or sensor nodes has been the goal of this paper. Besides general parameters typical for comparison in this domain, we have been specifically interested in memory usage for algorithm execution, as this is often the limiting factor on embedded hardware. Since we assume data decompression to be shifted to an entity outside the deployed network with no constraints on resource usage, the following results reflect solely characteristics of the compression step.

Due to need for a tool for memory estimations, which have been carried out with Valgrind [21], the experiments have been executed on a 2.4 GHz Intel x86 machine running MAC OS X. Except for the execution time, results will not differ when profiling is run on wireless mesh or sensor nodes. The execution time measurements provided below will nevertheless offer a clear intuition towards the performance on respective hardware, yet have to be repeated for a dedicated system on demand.

1) Compression ratio: The first and foremost parameter that seeks attention is the compression ratio a certain algorithm can provide. The compression ratio cr is calculated using the compressed data size cd and the original, uncompressed size ud as follows:

$$cr = 100 - (cd * 100)/ud$$

Due to the presence of redundancy in all data traces, the compression ratio is in general higher than 50%, see Figure 2. The best compression ratio with an average of 92% for the wireless mesh data and 67% for the sensor network traces can be granted by the order-n arithmetic coding. For the former set, the algorithm clearly benefits from the great coding possibilities since the WMN traces feature many IP addresses with similar network parts. The different LZW variants (lzss, lzw12, lzw15v and mlzo) target roughly similar values between 82% and 88% for WMN and 49% and 60% for WSN data. As expected, the Huffman implementations and the simple arithmetic coding yield almost the same results, around 54% compression ratio regardless of the type of input data. The only surprising result is the bad performance of SLZW, especially in regard to the sensor network data set. Although it has been specifically designed for WSNs, the algorithm provides with 35% the poorest compression ratio in this evaluation. This may partially be accounted to a lack of a fixed-size format of logfile entries due to dynamically-sized neighborhood lists, which prohibits the exploitation of input transformation, and thus the advantage of using a small cache.

2) Execution time: Another important metric to evaluate a compression algorithm is its execution time. Naturally, the more time is spent for computation, the more energy will be consumed by the processor. Furthermore, in case compression is applied in the domain of sensor networks or generally on embedded devices, operating systems may not support concurrent operations. Thus, a long-running execution of a compression algorithm will block the processor for a significant amount of time, in which, e.g., reactivity to important operational events is not given, the node is not responsive, and in the worst case, will fail to handle its tasks correctly. Figure 4 displays the average time in seconds needed to compress 1 MB of input data gathered from the mesh, and the sensor network, respectively.

The outstanding algorithm in terms of time for execution is the MLZO implementation: With an average of 5ms for WMN, and 10ms for WSN data, it is the fastest algorithm in this benchmark suite and roughly a factor 10 better than the next well performing algorithms (lzw12, lzw15v and slzw). The simplicity of the Huffman implementations and the order-0 arithmetic coding are directly reflected in their runtime results, which range between 7 to 15ms. Since much more information is exploited in higher order arithmetic coding, the time for their execution naturally increases and is with 500ms to even 700ms by a factor hundred slower than the MLZO algorithm.

3) Memory consumption: Each functionality pushed onto a wireless node will consume text memory thus ROM for its installation as well as RAM at runtime when being executed. It has been often times pointed out that especially the latter one is very limited, especially on embedded nodes, for reasons of energy expenditure for RAM refreshing. The tight boundaries given by the node have thus to be respected, keeping additionally in mind that compression is in most cases supposed to run as a complementary service and not as the primary application.

Figure 5 displays the text segment occupation in bytes (B) by the various implementations. All algorithms have been compiled with the same compiler settings. Since most implementations have not been optimized for size and much memory is occupied by extensive I/O functionality in their original implementation, all sources files for such file handling have been removed for a better comparison. ROM utilization ranges between roughly from 2.5kB to 5kB for the algorithms with the smallest memory footprint (lzw variants, huffman and arithmetic coding order 1) up to 15kB for SLZW as supplied by the authors. The latter is once again surprisingly high, as especially the authors of a sensor network algorithm should be aware that this will block almost one third of a typical sensor nodes storage capacity (48kB for the Tmote sky). However, all evaluated algorithms would fit into the text segment of a typically dimensioned wireless mesh or sensor node.

The results presented in Figure 6 have been acquired using the Valgrind framework for program profiling [21]. Here, the peak RAM usage in bytes is displayed on a logarithmic scale. All implementations use approximately the same amount of stack memory, consuming around 500 bytes. Heap allocation however differs tremendously, with the top user being MLZO occupying 3MB. This is absolutely not suitable for embedded hardware, where available RAM is dimensioned between 5 and 10kB. Also, higher order arithmetic coding or lzw15v are not portable as they are onto wireless nodes unless memory optimizations will be undertaken beforehand. Good candidates, however, are the SLZW and adaptive huffman coding, as well as the simple lzw12 and lzss algorithms.

IV. Conclusions

Data compression can be a very comfortable tool on wireless nodes to gain storage space and possibly spend less energy on transmissions. In this study, we evaluated different compression algorithms how they may be simply deployed on such nodes without any previous changes to their implementations. The results obtained and summarized in Table IV may hence be read as an evaluation of the given implementation.

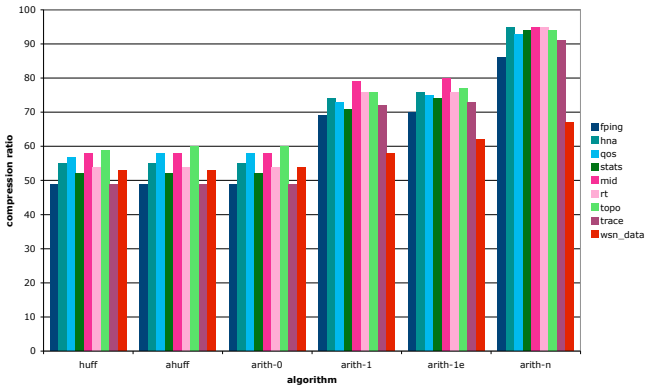

Fig. 1. Average compression ratio of Huffman and arithmetic coding algorithms obtained for CTI-Mesh and Tarwis-Measurements.

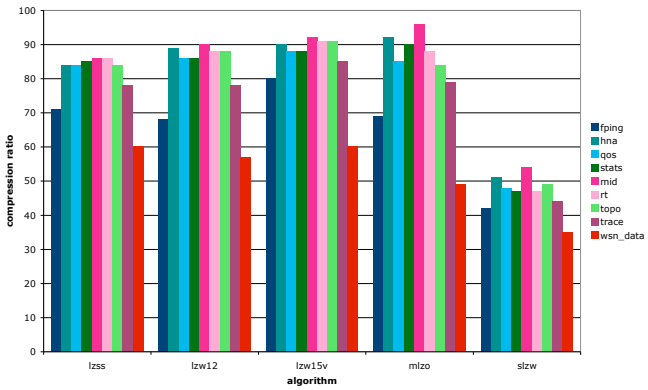

Fig. 2. Average compression ratio of LZW variants obtained for CTI-Mesh and Tarwis-Measurements.

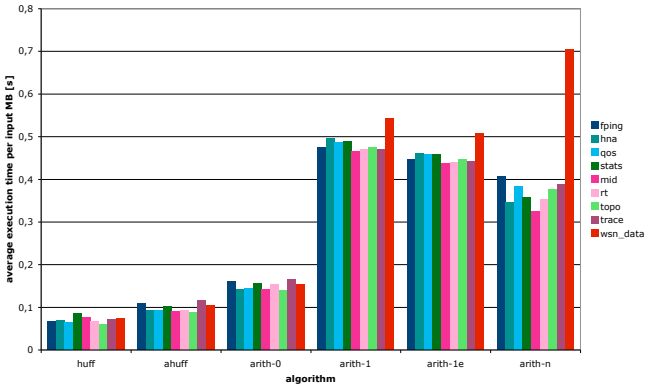

Fig. 3. Average time for compressing CTI-Mesh and Tarwis-Measurements with Huffman and arithmetic coding algorithms on a MAC OS X platform.

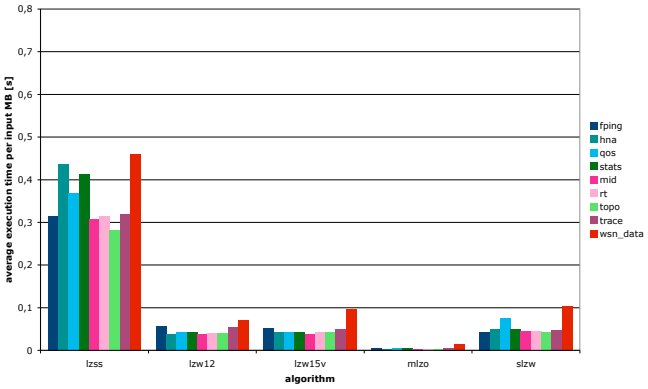

Fig. 4. Average time for compressing CTI-Mesh and Tarwis-Measurements with LZW variants on a MAC OS X platform.

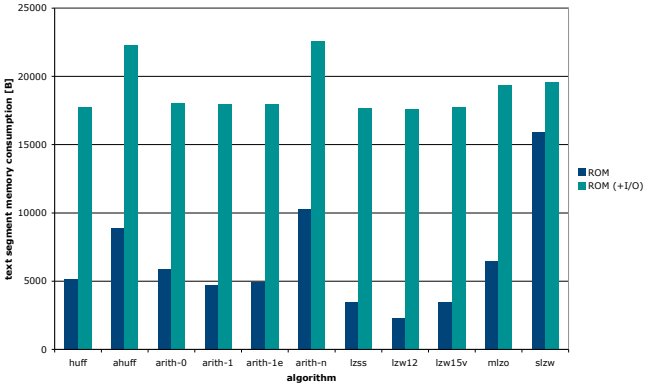

Fig. 5. ROM memory occupied by the implementation of the different algorithms with and without I/O operations in bytes.

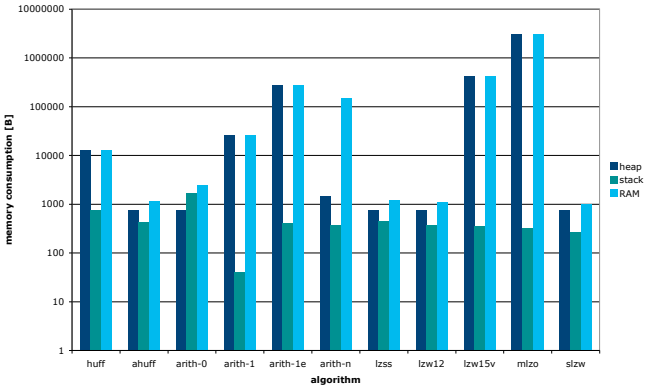

Fig. 6. Peak RAM memory usage of the different algorithms without I/O in bytes.

algorithm	ratio	exec time	ROM usage	RAM usage
huff	0	+	0	0
ahuf	0	+	0	+
arith-0	0	+	0	0
arith-1	+	-	0	-
arith-1e	+	-	0	-
arith-n	++	-	0	-
lzss	+	-	+	+
lzw12	+	+	++	+
lzw15v	+	+	+	-
mlzo	+	++	0	-
slzw	-	+	-	+

TABLE I
OVERVIEW OF BENCHMARK SUITE OVERALL PERFORMANCE.

Overall, we can conclude from this study that there is not the one, outstanding algorithm which suits all requirements. Rather, the choice has to be made according to the primary goals of application and restrictions posed by the utilized wireless node platform. When optimizing for speed, MLZO should be the choice unless resource restrictions regarding RAM memory exist. If a good compression ratio is mandatory, yet execution time and memory restriction may be disregarded, a higher order arithmetic coding algorithm is an appropriate candidate. LZW variants score well in most categories; especially the simple LZW implementation with 12bit symbols offers a solid performance in all categories. However, the attained values for SLZW are disappointing. Once again, the reasons for its failure can most probably be accounted to the structure of the data, hindering all proposed optimizations to come to their full potential.

REFERENCES

[1] K. Barr and K. Asanovic, "Energy-Aware Lossless Data Compression," in *Proceedings of the 1st International Conference on Mobile Systems, Applications and Services*, San Francisco, USA, 2003, pp. 231–244.

[2] K. Terfloth and J. Schiller, "Efficient Configuration and Control of SANETs using FACTS," in *In Proceedings of HeterSanet Workshop, colocated with ACM MobiHoc '08*, Hong Kong (SAR), May 2008.

[3] N. Kimura and S. Latifi, "A Survey on Data Compression in Wireless Sensor Networks," in *Proceedings of the International Conference on Coding and Computing*, vol. 2, 2005, pp. 8–13.

[4] A. Deshpande, C. Guestrin, S. R. Madden, J. M. Hellerstein, and W. Hong, "Model-driven Data Acquisition in Sensor Networks," in *VLDB '04: Proceedings of the Thirtieth international conference on Very large data bases*. VLDB Endowment, 2004, pp. 588–599.

[5] A. Scaglione and S. D. Servetto, "On the Interdependence of Routing and Data Compression in MultiHop Sensor Networks," in *Proceedings of the 8th annual international Conference on Mobile computing and Networking*, Atlanta, USA, 2002, pp. 140–147.

[6] S. Pattem, B. Krishnamachari, and R. Govindan, "The Impact of Spatial Correlation on Routing with Compression in Wireless Sensor Networks," in *Proceedings of the 3rd International Symposium on Information Processing in Sensor Networks*, Berkeley, USA, April 2004, pp. 28–35.

[7] S. S. Pradhan, J. Kusuma, and K. Racmchandran, "Distributed Compression in a Dense Microsensor Network," *IEEE Signal Processing Magazine*, vol. 19, no. 2, pp. 51–60, March 2002.

[8] C. Alippi, R. Camplani, and C. Galperti, "Lossless Compression Techniques in Wireless Sensor Networks: Monitoring Microacoustic Emissions," in *IEEE Workshop on Robotic and Sensor Environments*, Ottowa, Canada, 2007.

[9] C. Tharini and P. V. Ranjan, "Design of Modified Adaptive Huffman Data Compression Algorithm for Wireless Sensor Network," *Computer Science*, vol. 5, no. 6, pp. 466–470, 2009.

[10] D. Ganesan, B. Greenstein, D. Perelyubskiy, D. Estrin, and J. Heidemann, "An Evaluation of Multi-Resolution Storage for Sensor Networks," in *Proceedings of the 1st international Conference on Embedded Networked Sensor Systems*, New York, USA, 2003.

[11] N. Tsiftes, A. Dunkels, Z. He, and T. Voigt, "Enabling Large-Scale Storage in Sensor Networks Enabling Large-Scale Storage in Sensor Networks," in *Proceedings of the 8th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN 2009)*, San Francisco, USA, April 2009.

[12] C. Sadler and M. Martonosi, "Data Compression Algorithms for Energy-Constrained Devices in Delay Tolerant Networks," in *In Proceedings of the ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2006, pp. 265–278.

[13] F. Marcelloni and M. Vecchio, "An Efficient Lossless Compression Algorithm for Tiny Nodes of Monitoring Wireless Sensor Networks," *Computer Journal*, vol. 52, no. 8, pp. 969–987, 2009.

[14] M. Nelson and J.-L. Gailly, *The Data Compression Book*, 2nd ed. New York, USA: MIT Press, 1995.

[15] T. A. Welch, "A Technique for High-Performance Data Compression," *IEEE Computer*, vol. 17, pp. 8–19, 1984.

[16] C. Strydis and G. N. Gaydadjiev, "Profiling of Lossless Compression Algorithms for a Novel Biomedical Implant Architecture," in *Proceedings of the 6th IEEE/ACM/IFIP international conference on Hardware/Software codesign and system synthesis*, Atlanta, GA, USA, 2008.

[17] T. Staub, M. Anwander, K. Baumann, T. Braun, M. Brogle, K. Dolfus, C. Félix, and P. K. Goode, "Connecting Remote Sites to the Wired Backbone by Wireless Mesh Access Networks," in *Proceedings of the 16th European Wireless Conference*, Lucca, Italy, April 2010.

[18] P. Hurni, G. Wagenknecht, M. Anwander, and T. Braun, "A Testbed Management Architecture for Wireless Sensor Network Testbeds (TARWIS)," in *Proceedings of th 7th European Conference on Wireless Sensor Networks (EWSN)*, Coimbra, Portugal, February 2010.

[19] M. F. X. J. Oberhumer, "LZO version 2.03," http://www.oberhumer.com/opensource/lzo/, 2010.

[20] Moteiv, "Tmote sky datasheet," http://www.eecs.harvard.edu/ konrad/ projects/shimmer/references/tmote-sky-datasheet.pdf, 2006.

[21] C. Armour-Brown, J. Fitzhardinge, T. Hughes, N. Nethercote, P. Mackerras, D. Mueller, J. Seward, B. V. Assche, R. Walsh, and J. Weidendorfer, "Valgrind Release 3.5.0," http://valgrind.org/.