PINCO: a Pipelined In-Network COmpression Scheme for Data Collection in Wireless Sensor Networks

Tarik Arici*, Bugra Gedik**, Yucel Altunbasak*, Ling Liu**

*Electrical and Computer Engineering, **College of Computing

Georgia Institute of Technology

Abstract— We present PINCO, an in-network compression scheme for energy constrained, distributed, wireless sensor networks. PINCO reduces redundancy in the data collected from sensors, thereby decreasing the wireless communication among the sensor nodes and saving energy. Sensor data is buffered in the network and combined through a pipelined compression scheme into groups of data, while satisfying a user-specified end to end latency bound. We introduce a PINCO scheme for single-valued sensor readings. In this scheme, each group of data is a highly flexible structure so that compressed data can be recompressed without decompressing, in order to reduce newly available redundancy at a different stage of the network. We discuss how PINCO paremeters affect its performance, and how to tweak them for different performance requirements. We also include a performance study demonstrating the advantages of our approach over other data collection schemes based on simulation and prototype deployment results.

I. INTRODUCTION

A. Motivation and Problem Statement

Technological progress in wireless networks, low-power circuit design, and micro electro-mechanical systems (MEMS) has led to the production of tiny sensor devices about a cubic inch in size which enable applications that connect the physical world with pervasive networks. These sensor devices do not only have the ability to communicate information across the sensor network, but also to cooperate in performing more complex tasks, like signal processing, data aggregation and compression in the network rather then out of the network.

Motes developed at UC Berkeley [17] and manufactured by Crossbow Inc. [5] are one example of these tiny sensor devices. With their small physical size, sensing and computing capabilities, motes are highly practical and currently used in a variety of fields from habitat and environmental monitoring to different data collection applications [7] [11] [4] [13] [14].

There are three common features to all these variety of sensor network applications. The first one is the need for designing simple algorithms given that motes have highly constrained resources (*i.e.* energy, storage, and processing) [13] [14]. The second one is the need for minimizing communication among motes, since wireless communication is the primary energy consumer in these networks (*e.g.* sending a single bit of data can consume the energy of executing a thousand instructions) [15]. The third one is that, sensor data communicated throughout the network are likely to have both spatial and temporal redundancy being activated by the same phenomenon of interest [12] [9]. With these three features that set sensor networks apart from traditional networks, sensor applications have the main goal of collecting data from the network.

In-network aggregation schemes, simple in implementability, decrease wireless communication among motes by reducing redundancy in sensor measurements according to an aggregation function [13] [14]. However, the extracted data in response to a query is only a summary (aggregate) of sensor readings and this limits the feasible data aggregation applications to a set of applications that can only be formulated with an aggregation function. Unless the aggregation function have some properties such as *duplicate insensitivity, exemplarity* and *monotonicity* [19], the amount of communication in the network easily approaches that of a centralized, out-of-the-network approach. Also, several queries and statistical measures cannot be supported using aggregation. In contrast, our approach decreases wireless communication independent of the properties of the posed queries, and is applicable to any kind of query.

Our proposed Pipelined In-Network COmpression (PINCO) scheme for data collection in wireless sensor networks exchanges latency for energy by first buffering and delaying mote measurements, and then reducing available redundancy in spatial, temporal, and spatio-temporal domain among the sensor data in a buffer by compressing them into groups of data (GDs). An end-to-end (e2e) latency bound specified by the user when querying the network is satisfied by appropriately choosing the maximum waiting time of data in a mote's buffer. PINCO scheme for single-valued data is an error-resilient compression scheme. Each group has its own anchor data and therefore can be decompressed independent of other groups. A very important feature of PINCO is that GDs are compressed as highly flexible structures so that they can be recompressed without decompressing. Recompression exploits the likely available redundancy among GDs at later stages of the mote network while propagating towards the basenode. Without the need for decompression it does not impose any additional cost on the mote's highly constrained resources.

B. Related Work

To our knowledge there exists no previous work that has proposed an in-network compression scheme that bounds the latency and enables recompression for mote-based sensor networks. But schemes for pushing the query into the network, processing the data in the network, and compression for data reduction at the source are proposed for wireless sensor networks in general [9] [12] [13] [14] [16]. Research on in-network data processing for sensor networks is mainly focused on data aggregation and will be summarized and then compared with our approach in Section III. Also, motivation for compression rather than aggregation will be discussed.

II. MOTE-BASED NETWORKS

In this section, we briefly present the mote hardware platforms available, and a routing algorithm for propagating sensor data from motes to the basenode in the presence of mobility and network losses.

A. The Mote Platform

The two available mote hardware platforms are rene and mica developed at UC Berkeley and manufactured by Crossbow Inc. Mica nodes are the most recent harware platform and have 4 MHz Atmel [1] processors with 128 KB of programming memory, and 512 KB of data memory. These nodes are equipped with a 916 MHz, single channel, low-power radio from RFM [3] with 50 kbps transmission rate using on off keying (OOK) modulation. A suit of sensor boards for a variety of sensors including light, acoustics, magnetic field, power, acceleration, and temperature are available and can be connected to the mote using a 51 pin expansion slot.

A mote consumes 1 μJ for transmitting and 0.5μJ for receiving a single bit. Furthermore it consumes 0.8 μJ for executing roughly 100 instructions [10]. This shows that extremely simple in-network schemes for reducing the communication by increasing the computation is needed for mote-based networks.

The mote radio antenna is an omni-directional antenna so that radio communication channel is a broadcast medium. A transmitted packet is received not only by the destination node, but all the nodes in one-hop local neighborhood of the source node.

B. Wireless Routing Algorithm

Researchers have proposed different ad-hoc routing algorithms [9] [12] [18] for sensor networks. These algorithms take into consideration the data-centric and many-to-one nature of sensor data flows created by propagating sensor measurements of the same phenomenon from many nodes to the basenode. PINCO requires the routing protocol to provide the delivery of the query request to all nodes in the network, finding at least one route from sensor nodes back to the basenode for data collection and learning the number of hops traversed on the longest route passing through a node. A simple routing algorithm capable of these three basic services and proper for motes with its simplicity is a tree-based routing algorithm.

We use a tree-based routing algorithm for supporting PINCO. In this algorithm, one mote is chosen as the *root* node usually because it is the node that interfaces between the sensor and the wireline network. Before receiving a tree advertisement message all nodes have their *hop-distance* to the root set to the maximum possible value. The root node initiates the routing tree formation process by broadcasting a tree advertisement message (TAM) specifying its own node id and hop-distance from the root (*i.e.* zero in this case). All the motes in the network set the sender of TAMs as their *parent* while minimizing their hop-distance to the root. We preferred hop-distance minimization over latency minimization in making the parent selection decision for the tree formation process. Latency minimization for parent selection does not form the optimum routing tree for data collection since latency highly depends on the MAC layer

contention, and random defer and backoff times [8]. After selecting or updating its parent, each node rebroadcasts the TAM inserting its own id and hop-distance (*i.e.* one plus its parent's hop-distance). TAMs disseminate into the network completely in this fashion and all nodes learn their parent and hop-distance from the root.

In order to satisfy the user-specified e2e latency bound on sensor measurements every mote has to set an appropriate maximum waiting time for data in its buffer. For this purpose, a mote has to know the length of the longest tree branch passing through itself that is connecting the furthest leaf node and the root. In order to learn this from the network, every mote first has to decide whether the tree formation process has ended. This can be learnt for free by snooping on the channel to see if TAMs are rebroadcasted by the neighbor nodes. Once the tree formation process has ended, leaf nodes transmit their hop-distance to their parents and parent nodes suppress all the hop-distance reports other than the maximum hop-distance reported and in turn transmit it to their own parents. This suppressing of smaller hop-distance values is very similar to the negative acknowledgement suppressment in scalable multicast protocols and computing MAX aggregates as in [13]. Then maximum waiting time in a mote's buffer is easily calculated at every node by dividing e2e latency bound by the length of the longest branch passing through it.

The root node periodically initiates a new tree formation process for adaptation to changing network dynamics caused by mobility, or addition or deletion of motes. In order to select the most recent parent and forming loop free routing trees, each TAM originated by the root node contains an increasing sequence number. Every node strictly prefers higher sequence numbered TAMs.

After the formation of the routing tree, motes start sending sensor data to their parents, which in turn relay the data to their parents, and so on. In Subsection III-C we discuss how PINCO compresses single-valued data in the mote buffers into GDs as it propagates towards the root. In the next section we discuss data collection schemes in wireless sensor networks.

III. DATA COLLECTION IN WIRELESS SENSOR NETWORKS

In this section we classify data collection schemes, discuss their respective merits and weaknesses, and present PINCO, an in-network compression scheme for data collection in wireless sensor networks.

A. Plain Data Collection

Plain data collection is an approach where each sensor node sends its measurement to a basenode, at which data is stored and processed [6]. Since in a sensor network, nodes are deployed in close physical proximity of a stimulus of interest for increasing the accuracy and reliability of the collected information, each sensor reading has to be propagated back to the basenode over multi-hops using the wireless communication channel. The disadvantage of this approach is the high communication cost it incurs on the node's small energy budget due to the large amount of data communicated in the network. However, this approach does not put any constraint on the structure of the requested

query and has the advantage of being able to support an unlimited set of queries and statistical measures on the collected data.

B. In-network Data Aggregation

In-network aggregation is an approach that has recently been studied in detail by several researchers [13] [14] [12]. In-network aggregation reduces the communication between the nodes by combining data from different sources according to an application specific aggregation function. In this approach, sensor nodes transmit only the data that is sufficient to answer the aggregate query at hand, which can be one of the traditional aggregate operations MIN, MAX, AVG, and COUNT or any other aggregate query that has a suitable function for performing in-network aggregation. A representative query is given below in Example 1.

Example 1: Consider a network of sensor nodes that collect temperature data and route to a basenode. A continuous aggregate query like "report back the maximum temperature every t secs" can be efficiently computed by pushing the query into the network so that each node only sends data that is the maximum of the temperature values produced by all the nodes in the subtree rooted at that node. There are two approaches to in-network aggregation, namely: pipelined aggregation [14] and interval based aggregation [13]. We describe each of them below:

1) Pipelined Aggregation: In pipelined aggregation each node combines the data it received from its children in the previous interval with the data it currently produced. To elaborate, first consider a leaf node in the routing tree. The leaf sensor node produces a data item periodically every t seconds and as it produces a data item, it immediately transmits it to its parent node. For the non-leaf node, at the time it has received data from its children it has already transmitted its previous aggregate to its parent. Therefore it combines the data received from its children with the data it recently produced by taking the maximum (following Example 1) and transmits the resulting aggregate to its parent.

It is clear that pipelined aggregation has much lower communication cost compared to the plain data collection approach. However, in the pipelined approach some level of temporal redundancy in the sensor data is needed since aggregates are computed by combining data from different time intervals.

2) Interval Based Aggregation: Interval based aggregation is very similar to pipelined aggregation, except that nodes aggregate data sensed for the same time interval. In this approach, each node waits for its children to transmit their data until a certain time assigned by its parent, and then combines the received data with its own data for the same time interval and transmits the aggregate to its parent. Therefore, unlike pipelined aggregation, interval based aggregation does not need temporal redundancy in the sensor data for achieving good performance. But in this scheme the query re-evaluation interval (*i.e.* t) has to be larger than the latency time between the basenode and the furthest leaf node in the routing tree. This might be a limitation for sensor applications when the network topologies result in deep routing trees. Also child nodes transmitting data in time intervals specified by their parents

may lead to synchronization between them and increase the contention in the MAC level.

In general, all aggregation approaches are targeted for applications where only aggregates on the sensor data is of interest. Several queries and statistical measures cannot be supported using aggregation; variance, median, histograms, maps, and complex SQL type queries to name a few. Also, if the aggregation function of the requested query lacks some properties such as *duplicate insensitivity* , *monotonicity*, or *exemplarity* [19] the amount of communication occuring in the network can quickly approach that of a plain data collection scheme.

C. In-network Compression

In many applications such as habitat monitoring, full data collection is desirable in order to perform off-line mining of the collected data or to support complex queries on it. Consider the simple application from Example 1 in Section III-B. Assume that instead of a maximum temperature aggregate we are interested in mining the time evolving temperature maps for moving patterns of an object. In case there exists a catalog describing positions of each sensor node, retrieving full data from the network including the node identifiers and timestamps, will enable us to perform such operations. However, using plain data collection will require too much energy consumption, and therefore decrease system lifetime. Our approach to this problem is to perform in-network compression so that full data collection is possible while reducing the energy consumption.

The PINCO algorithm given in Algorithm 1 is applicable to data of any dimension and size. Researchers may also implement their own *merge()* function to exploit redundancies characteristic to their application or to test their own compression algorithms[1]. We next describe our proposed in-network compression scheme for single-valued sensor readings (*e.g.* light, acoustics, magnetic field, power, acceleration, and temperature).

Pipelined In-network Compression for Single-valued Data:

PINCO trades higher latency for reduced energy consumption. Sensor data is buffered at each node and then delayed for some appropriate time satisfying the e2e latency bound as explained in Subsection II-B. Available redundancy is reduced by exploiting the commonalities among the buffered data. Since each node buffers its own data and it is possible to compress data sensed at the same time intervals, PINCO does not require temporal redundancy for good performance although it is a pipelined compression scheme.

A data item produced is a tuple of three values; an n-bit sensor measurement, a node identifier, and a timestamp. It is denoted by a tuple shown as ⟨*measurement,nodeid,timestamp*⟩. A compressed group of items is also a tuple but with four attributes. The first attribute is the shared prefix which is the most significant bits that are common to all sensor measurements in that group. The shared prefix length (*spl*) is a system parameter that specifies how much commonality in the sensor measurements is required when forming a group. The second

---

[1] PINCO source code for both the ns-2 and TinyOS-0.6 will be made available on the Internet to the researchers.

Algorithm 1 PINCO Algorithm

Description: $receiveDataGroup$ function is called when a new data item is produced or a new data group is received from a child node.

function $receiveDataGroup(DataGroup\ ng)$

  for all $g \in buffer$ **do**

    {Merge the new group to the current group. Leave the remaining (if any) in the new group. If a merge is not possible the new group is not modified. Otherwise a full merging is done if possible, in which case the remaining is empty. If the current group gets full during merging, a partial merge is performed and the un-merged portion of the new group remains.}

    $ng = g.merge(ng)$

    {If the current group is full, send it and remove it from the buffer.}

    if $g.isFull$ **then**

      $sendDataGroup(g)$

      $buffer.remove(g)$

    end if

    {If the new group is empty, exit loop.}

    if $ng.size = 0$ **then**

      break

    end if

  end for

  {If the new group is non-empty, add it into the buffer as a new group.}

  if $ng.size \neq 0$ **then**

    $ng.entryTime = currentTime$

    $buffer.add(ng)$

  end if

  {While the buffer is full, evict the group with the largest size and send it.}

  while $buffer.size > maxBufferSize$ **do**

    $g = argmax_{g \in buffer} g.size$

    $sendDataGroup(g)$

    $buffer.remove(g)$

  end while

Description: $scanBuffer$ function is called periodically to send expired data groups to the parent node.

function $scanBuffer()$

  for all $g \in buffer$ **do**

    {If the current group got expired, send it and remove it from the buffer.}

    $waitTime = currentTime - g.entryTime$

    if $waitTime >= maxWaitingTime$ **then**

      $sendDataGroup(g)$

      $buffer.remove(g)$

    end if

  end for

---

attribute is a list of suffixes where each suffix is a data item's $n - spl$ least significant bits. The third and fourth attributes are lists of node identifiers and timestamps respectively. The order of the node identifiers and timestamps match with the order of the sensor measurements. A GD is denoted by a tuple shown as ⟨*shared prefix,suffix list,nodeid list,timestamp list*⟩.

Any data item produced is converted to a GD by itself before being buffered. During a GD's lifetime in a node's buffer, it is merged with newly produced GDs by the node itself or recently received GDs from the child nodes, if possible. Two GDs can be merged only if they have the same shared prefix values. The merge operation is done in a straightforward manner preserving the order in the four attributes of the groups. A similar compression scheme can also be implemented on the timestamp list further increasing the energy savings.

If any new data item added to an existing group has the same sensor measurement value with the shared prefix of the group, its suffix value (*i.e.* zero) is not inserted to the suffix list. But its node identifier and timestamp values are inserted to the head of the corresponding lists. When decompressing zero is assigned to a data item's missing suffix value. This way it is possible to achieve a significant reduction in the bit rate of the measurement data which remains entirely lossless.

Having larger buffers reserved for GDs will increase the compression ratio by exploiting temporal redundancy among the sensor measurements to a greater extent. However, this will increase the observed e2e latency which will become a consideration if the application is online monitoring rather than offline monitoring.

If some level of error is tolerated in the collected data, PINCO will further compress GDs with some loss in the sensor measurements. Assume the user can tolerate an error of ϵ, then any suffix smaller than ϵ in the suffix list will be set to zero and deleted from the list increasing the compression ratio of a GD.

IV. PERFORMANCE EVALUATION

A. Simulation-based Performance Evaluation

We implemented PINCO in the ns-2.26 snapshot of the ns-2 simulator [2] and used its CSMA model as the medium access control protocol. Tree-based routing protocol and PINCO scheme is implemented as separate ns-2 modules. To simulate mote networks, packet size is set to mote packet size (30 bytes). ns-2 connectivity model is used which is based on the geographic distance between the nodes.

The data-collection application in our simulations creates the temperature map of the scenario field. Sensor node's in the field are immobile, and their geographic positions are cataloged during the deployment period and stored in the basenode where the requested query is initiated. Two temperature distribution patterns are used. In the first one, temperature in the field varies with the distance from a central point where the temperature is sampled from a normal distribution every second. In the second one, temperature at each node is sampled from a uniform distribution independent of the others. Number of nodes in the field is chosen as 50, 100, and 150 for different simulation scenarios and the simulation time is 300 seconds.

We have observed that minimum-hop based routing tree created *straggler* nodes in our simulations. Stragglers are nodes that miss some of the broadcasted TAMs because of the MAC level contention and therefore connect to the routing tree via *backward links* in which the recipient of the TAM is closer from the basenode than the transmitter. In order to avoid straggler nodes we improved our tree formation algorithm so that any node hearing a TAM with hop distance two or more greater than its own hop distance rebroadcasts a TAM thereby enforcing all feasible shortest paths in the routing tree among a node's neighbors.

Simulation results have shown that for good performance, PINCO parameters have to be tweaked for specific application characteristics. Using our results we investigated the effect of PINCO parameters on two performance measures: observed e2e latency and transmission cost, in other words the amount of dissipated energy.
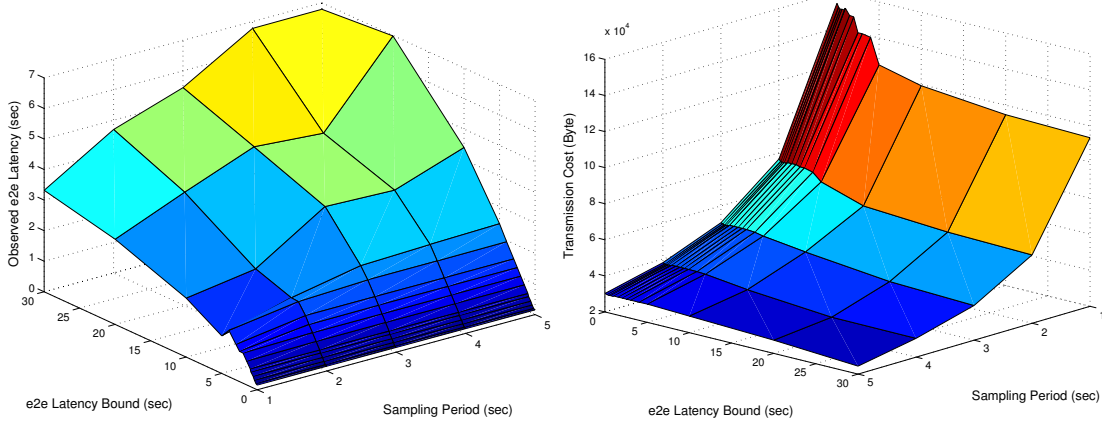
Fig. 1. Impact of e2e Latency Bound and Sampling Period
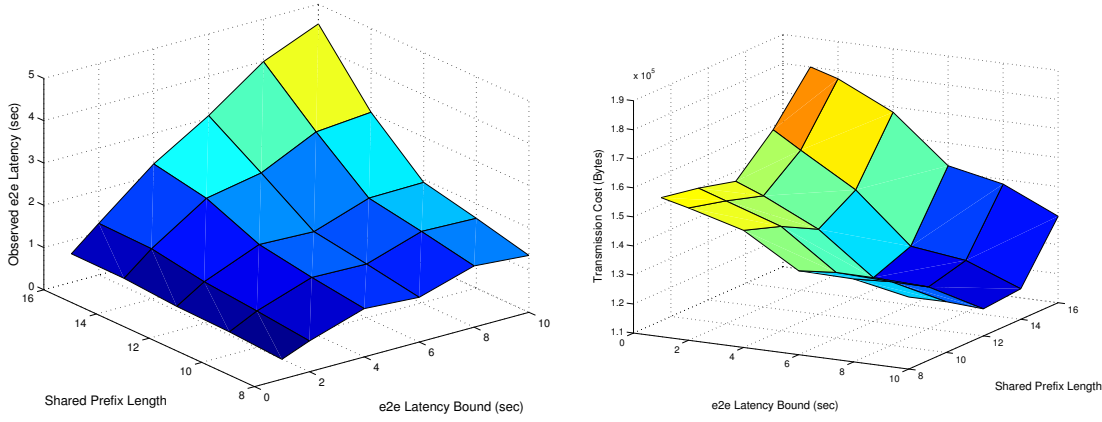

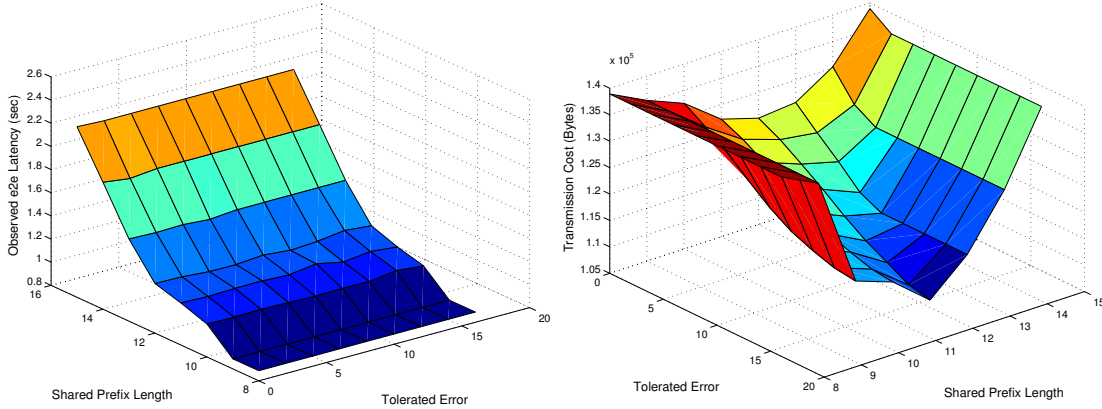Fig. 2. Impact of e2e Latency Bound and Shared Prefix Length

Fig. 3. Impact of Tolerated Error and Shared Prefix Length

Since PINCO trades higher latency for reduced energy consumption, increasing the e2e latency bound decreases the transmission cost of our application (see Figure 1). When the sampling period is smaller, marginal decrease in the transmission cost is higher as an additional delay in a node's buffer increases the likelihood of a group getting merged with other recently received groups. As expected, observed e2e latency increases with increasing e2e latency bound. The amount of increase in the latency is smaller for small sampling periods because with a high data rate groups quickly reach their limit size (*i.e.* the packet size) and gets transmitted from the buffer.

For a new data item to be inserted into a group, its shared prefix has to match with that of a group. Therefore shared prefix length (spl) determines the number of groups in a buffer. SPL equal to 16 corresponds to a plain data collection approach with duplicate data suppression since all data items in a group must have the same sensor measurement. From Figure 2, we can see that energy dissipation is reduced with PINCO compared to a duplicate suppressing plain data collection scheme. Also observed e2e latency improves with optimum spl. Although

reducing the spl increases grouping, it does not necessarily increase the compression ratio. With a smaller spl, each suffix is larger in size, hence higher transmission cost on a node. Inspecting Figure 2, we can see that an optimum spl (in this case 12) has to be chosen depending on the variance among the sensor temperature readings.

In the following we present a mathematical analysis of grouping in a buffer to find the optimum spl. For simplicity we assume sensor readings are gaussian distributed with mean μ and variance σ^2. Since for high compression ratio spl has to be chosen appropriately so that it is neither too small resulting in large suffix values, nor too big decreasing the likelihood of grouping in a buffer, our objective function for maximization is the compression ratio (CR) given as below:

$$CR(spl) = spl * P\{x \, \epsilon \, GD\} \qquad (1)$$

and $P\{x \, \epsilon \, GD\} = P\{E[SP] < x < E[SP] + 2^{(n-spl)}\}$

where x is any sensor reading and n is its size, and $E[SP]$ is the expectation of the shared prefix. Since SP is the discretized form of x with step size of 2^{n-spl}, $E[SP]$ is the summation of rectangles approximating to μ. Assuming $E[SP]$ to be $\frac{spl}{n} * \mu$, a complex form for spl optimizing (1) can be found.

Inspecting the complex form for optimum spl, (2) can be chosen as its approximate estimator. Also, the standard deviation estimate $\hat{\sigma}$ can be found by low-pass filtering the recent values with $\alpha\hat{\sigma} + (1 - \alpha)(x - \hat{\sigma})$ where α is any number in [0,1] reflecting our expectation on the tendency of sensor data to change both in time and space.

$$\hat{spl} = n - log_2(\hat{\sigma}) \qquad (2)$$

However, adaptively adjusting spl brings overhead along with its simplicity because each GD has to indicate its spl. (2) must be used only when the system designer does not have any prior knowledge or tool to determine the optimum spl.

Transmission cost in an application decreases with the amount of tolerated error (ϵ) as can be seen in Figure 3. As the suffix values smaller than ϵ are deleted from the group, compression ratio increases at the expense of some tolerated loss added to the sensor readings.

B. Deployment-based Performance Evaluation

Encouraged by PINCO's good performance based on simulation results, we started implementing the PINCO scheme on rene2 platform using TinyOS version 0.6. Our initial results show that one-directional links are very likely in mote communication conflicting with ns-2 physical layer model assumption of connectivity which is based on georgraphical distance. We are planning to modify our routing protocol to utilize only bi-directional links by blacklisting one-directional links using MAC level acknowledgements. Although we still did not finish with the complete PINCO scheme implementation, initial results show that performance increase with PINCO is as expected and agreeing with the simulation results.

V. CONCLUSION AND FUTURE WORK

Data collection schemes for wireless senosor networks push the requested query into the network and process the raw data using an aggregation function. However, when full-data collection applications rather than aggregates on a phenomenon are needed, aggregation is not useful. In this paper, we have shown that in-network compression enables energy-efficient full-data collection applications. Our in-network compression scheme trades higher latency for lower energy consumption. With the presented PINCO scheme for single-valued data, it is possible to recompress data groups to reduce redundancies that are available in different stages of routing through the network.

We believe that PINCO is a very useful framework in handling energy-efficient full-data collection applications and helping us to discover tradeoffs when tweaking in-network compression schemes for different degrees of performance requirements. For this reason, as the continuation of this work we are planning to develop PINCO schemes for complex types of sensor data such as audio and video rather than single-valued data as it is in this work.

REFERENCES

[1] http://www.atmel.com/atmel/products/prod23.html. Atmel 8-bit RISC Processor.
[2] http://www.isi.edu/nsnam. ns-2 Network Simulator.
[3] http://www.rfm.com/products/data/tr1000.pdf. RF Monolithics.
[4] http://www.smalltimes.com. Small Times: Big News in Small Tech.
[5] http://www.xbow.com. Crossbow Technology, Inc.
[6] P. Bonnet, J. Gehrke, and P. Seshadri. Towards sensor database systems. In *Proceedings of the Second International Conference on Mobile Data Management*, volume 43, pages 551–558, Hong Kong, January 2001.
[7] D. Estrin, R. Govindan, J. S. Heidemann, and S. Kumar. Next century challenges: Scalable coordination in sensor networks. In *Mobile Computing and Networking*, pages 263–270, 1999.
[8] D. Ganesan, B. Krishnamachari, A. Woo, D. Culler, D. Estrin, and S. Wicker. An empirical study of epidemic algorithms in large scale multihop wireless networks, 2002. Submitted for publication, February 2002.
[9] J. S. Heidemann, F. Silva, C. Intanagonwiwat, R. Govindan, D. Estrin, and D. Ganesan. Building efficient wireless sensor networks with low-level naming. In *Symposium on Operating Systems Principles*, pages 146–159, 2001.
[10] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister. System architecture directions for networked sensors. In *International Conference on Architectural Support for Programming Languages and Operating Systems*, 2000.
[11] J. M. Kahn, R. H. Katz, and K. Pister. Next century challenges: Mobile networking for "smart dust". In *ACM/IEEE International Conference on Mobile Computing and Networking*, 199.
[12] B. Krishnamachari, D. Estrin, and S. Wicker. Impact of data aggregation in sensor networks. In *International Workshop on Distributed Event-Based Systems*, 2002.
[13] S. Madden, M. Franklin, J. Hellerstein, and W. Hong. Tag: a tiny aggregation service for ad-hoc sensor networks. In *OSDI*, December 2002.
[14] S. Madden, R. Szewczyk, M. Franklin, and D. Culler. Supporting aggregate queries over ad-hoc wireless sensor networks. In *IEEE Workshop on Mobile Computing Systems and Applications*, 2002.
[15] G. J. Pottie and W. J. Kaiser. Wireless integrated network sensors. In *Comm. ACM*, volume 43, pages 551–558, 2000.
[16] H. Wang, D. Estrin, and L. Girod. Preprocessing in a tiered sensor network for habitat monitoring. In *EURASIP JASP special issue of sensor networks*, volume 4, pages 392–401, March 2003.
[17] B. Warneke, M. Last, B. Liebowitz, and K. Pister. Smart dust: Communicating with a cubic-milimeter computer. In *Computer Magazine, IEEE*, pages 44–51, Piscataway, NJ, January 2001.
[18] A. Woo and D. Culler. A transmission control scheme for media access in sensor networks. In *Mobicom*, 2001.
[19] J. Zhao, R. Govindan, and D. Estrin. Computing aggregates for monitoring wireless sensor networks. In *IEEE International Workshop on Sensor Network Protocols and Applications*, Anchorage, AK, May 2003.