Energy Efficient Communication for Reliability and Quality Aware Sensor Networks

Cristiano Pereira,[†] Sumit Gupta,[†] Koushik Niyogi,[†] Iosif Lazaridis,[†]
Sharad Mehrotra,[†] Rajesh Gupta[§]
[†]School of Information & Computer Science [§]Dept. of Computer Science & Engineering
University of California at Irvine University of California at San Diego
{cpereira, kniyogi, sumitg, iosif, sharad}@ics.uci.edu, gupta@cs.ucsd.edu

Technical Report # 03-15
April 21, 2003

Nodes in a sensor network are typically severely constrained by the amount of power available to them. Furthermore, power consumption by the wireless radio in a sensor node is an order of magnitude higher than the power consumption of computation on the node. We present two techniques that exploit this characteristic of sensor nodes to reduce the power requirements by reducing the amount of communication in a client-server sensor network. These techniques are: (1) a compression technique that produces an approximate representation of the actual data at the sources within *bounded quality guarantees*, and (2) a *delta encoding* technique that sends only the difference between the previous value and the current value of the data. Whereas the first technique sends data intermittently based on the rate of data value change, the difference technique sends data at regular intervals. We present results for experiments conducted on a testbed consisting of mobile units that collect data and communicate it to a server for processing. These results demonstrate a reduction of 86% and 80% in data sent when using the compression and delta encoding techniques respectively, and energy savings of 43% and 22%.

Contents

List of Figures

List of Tables

1 Introduction

Advances in microelectronic systems have made it possible to provide truly mobile, networked 'nodes' that enable distributed data gathering and computation. These nodes have enabled a range of applications, many of which fall under the broad category of *sensor networks* [15, 6, 2, 7, 13]. In many sensor network applications, nodes in the network have to frequently communicate data with each other or with a central server. Also, sensor nodes have limited power sources and communication over wireless radios is a key cause of battery drain in these devices [12].

The focus of this work, therefore, is to reduce the amount of data communication and thereby, the power consumed by the radio. The application domain targeted by our approach is a network of sensors that send collect data from the field and sent it to a server for further processing. Applications include monitoring air quality at a chemical plant, vital statistics of patients at home and location and condition (temperature, vital statistics) of assets (soldiers and equipment) in a battlefield.

We adopt an approach that exploits two characteristics of this application domain: (a) the small change in data values during nominal operation (for example, vital statistics like heart beat and body temperature exhibit low rate of change), and (b) the lower power cost of performing computations on a sensor node compared to communication over the wireless radio.

In this paper, we present two techniques to reduce the amount of data that a sensor node has to send a server. The first technique sends an approximate representation of the data instead of the actual data itself. This technique sends data only if the data value changes beyond a pre-defined threshold. This threshold is a bounded measure of *quality* and can be defined by the user based on criticality of the data value. In effect, we compress the actual data collected by the sensor and send a quantized representation of the data. Hence, we term this technique as the *compression* technique.

The second technique, called *delta encoding*, exploits data locality and sends only the difference in the current and last data value. Whereas, the compression technique is lossy, the delta encoding technique provides accurate data. Also, delta encoding sends data at regular intervals versus the compression technique that sends data only when data changes beyond the threshold. Regular communication is a requirement in some systems; for example, while monitoring the vital statistics of patients.

Our sensor network comprises of 50 devices able to communicate with each other and the fixed wireline network over the wireless 802.11b interface. The nodes in our network send data such as battery power, GPS coordinates and network bandwidth utilization to a central server. Note that, data such as these, varies significantly over time when compared to vital statistics, temperature, or air quality. Hence, the techniques that we present to reduce communication are even more effective for data that remains close to its nominal value.

The rest of this paper is organized as follows. In Section 2 we present the related work. Section 3 shows the quality aware compression algorithm, followed by Section 4, which presents the reliability aware delta encoding technique. Section 5 briefly describes the system architecture from which the experimental data was collected and Section 6 shows the results on the data. Finally Section 7 presents the conclusions and future work.

2 Related Work

The compression technique trades quality for reduced communication. The issue of quality and communication trade-off has been studied by Olston *et al.* [4] to reduce communication between the server

and a single source. Interval-based approximations are stored at the receiver end, supplied as guarantees by the producer. Our work differs in employing a general model of approximate replication which considers temporal latency and combines data compression and prediction. The adaptation problem was also studied by Deolasee *et al.* [5] for web data.

Chen *et al.* [3] propose to perform on-line regression analysis over time series data streams. We also propose to fit models to time series, but once again our motivation is to improve system performance, rather than regression analysis. A useful extension to our work would be to use some of the ideas in [3] to address correlations between multiple time series that a single sensor may be monitoring.

The idea behind the difference technique is not new. A similar technique is used to compress video frames in the MPEG-1 video protocol. In this protocol, full image frames are encoded only when the image on the frame changes significantly over the last frame. At other times, intermediate frames are encoded that contain only the difference between the current and last frame. Mogul *et al.* [11] presented a trace analysis for delta encoding and compression on HTTP traffic. The idea is to reduce communication by only sending the "deltas" or differences between dynamic web page contents. Korn and Vo [9] present an algorithm and platform independent data format for implementing both data differencing (delta encoding) and data compression. The paper also presents empirical results on the effectiveness of the data format proposed.

3 Quality Aware Data Management

The compression techniques have been built as part of a data management system called QUASAR, for *Quality Aware Sensor Architecture*. Since the data we monitor varies over time, it can be thought of as a time series data. We begin with some definitions and then present an algorithm for data compression.

3.1 Compressing Data with Quality Guarantees

A time series data can be defined as a sequence $S = \{s[1], s[2]...\}$ where $s[k]$ is the value at time instant k. We assume that time is discrete and denote the time domain as $T = \{1, 2...\}$. The time quantum corresponding to one step is the sampling period of the device. The observed series at time n is denoted by $S^n = \{s[1], s[2], ...s[n]\}$. To reduce communication related energy drain, our objective is to capture an approximate version $\hat{S}^n$ in our archive at the server. We may use several metrics such as the *EuclideanDistance* to compare the two time series. However we use a stronger notion of quality which says that the estimation of any individual sample $\hat{s[k]}$ should not be more different from $s[k]$ than an upper bound ε_{capt}. The parameter ε_{capt} is user defined and describes the quality of the approximate time series data of each device stored at the archive.

3.2 The Piecewise Constant Approximation

An attractive type of lossy compression is the *piecewise constant approximation*[1] (PCA) [8], whereby the time series S is represented as a sequence of K segments:

$$PCA(S^n) = \langle\ (c_1,\ e_1),\ (c_2,\ e_2),\ \ldots\ ,\ (c_K,\ e_K)\ \rangle$$

[1] This was called *Adaptive Piecewise Constant Approximation* (APCA) in [8] to distinguish it from a similar approximation (PAA) with equal segment lengths.

where e_k is the end-point of a segment and c_k is a constant value for times in $[e_{k-1}+1,\ e_k]$, or for times in $[1,\ e_1]$ for the first segment. In such an approximate representation, we estimate $s[k]$ as:

$$\widehat{s}[k] = \begin{cases} c_1 & \text{if } k \leq e_1 \\ c_m & \text{if } e_{m-1}+1 \leq k \leq e_m \end{cases}$$

Based on these definitions, we now reproduce our compression algorithms that were developed earlier in the context of database systems [10].

3.3 Poor Man's Compression

Poor Man's Compression (PMC) is a bare-bones form of compression that can be used to reduce the size of a time-series representation. It is an on-line algorithm, producing segments of a piecewise constant approximation (PCA) representation as new samples arrive. It requires only $O(1)$ space and performs $O(1)$ computation per sample. Hence, its overall time complexity for a series of size n is $O(n)$. This computation is interspersed with the arrival of samples; the compressed series is "ready to go" as soon as the last sample is processed. Hence the communication latency is minimized.

Let $s[i:j]$ be some time series. Lemma 1 supplies the necessary and sufficient condition for compressing $s[i:j]$ in a single segment in a manner that preserves the ε_{capt} guarantee.

Lemma 1 *The time series $s[i:j]$ can be compressed in a single segment $(c,\ j)$ with an error tolerance ε_{capt} iff:*

$$range[i:j] = \max_{i \leq k \leq j} s[k] - \min_{i \leq k \leq j} s[k] \leq 2\varepsilon_{capt}$$

The proof of this lemma is presented in [10].

3.3.1 Poor Man's Compression - Midrange

Figure 1 lists our compression algorithm: the Poor Man's Compression - Midrange (PMC-MR). This algorithm uses the converse of Lemma 1. PMC-MR takes a time series as input and produces a compressed time series as output. The algorithm monitors the range of its input. While this is less or equal to $2\varepsilon_{capt}$, it updates the range if needed (lines 11-12). When the range exceeds $2\varepsilon_{capt}$ at time n, then the algorithm outputs the segment ending at $n-1$ and a constant that is equal to the midrange of the preceding points (line 7). The algorithm then tries to compress the next set of samples, starting at time n (lines 8-9).

PMC-MR not only achieves the goal of compression, but satisfies an even stronger property: that *no* other PCA representation satisfying the ε_{capt} constraint, over *any* input time series can be a valid compression for that time series if it has fewer segments. PMC-MR is thus *instance optimal* not only for the class of on-line algorithms, but over *any* algorithm that solves this problem correctly. We state our claim formally as:

Theorem 1 *Let $S^n = s[1:n]$ be an aribitrary time series that must be approximated with a piecewise constant approximation that satisfies for all $k = 1, 2, \ldots, n$ that $|\widehat{s}[k] - s[k]| \leq \varepsilon_{capt}$. If the PMC-MR algorithm (Figure 1) produces a PCA(S^n) representation with K segments, then no valid PCA representation with $K' < K$ segments exists.*

The proof to this theorem is also in [10].

4 Reliability Aware Data Management

Sensors are often part of a network that requires a high reliability. If a node fails, the network has to reconfigure to adapt for the failed node. Thus, these nodes have to continuously send updates to the server. If a node does not send data for a certain time threshold, the server may assume the node is dead. For example, this may be the case for an asset in a battlefield; not sending updates may indicate loss of the asset. To minimize the amount of data transmitted in such high-reliability sensor networks, we propose a *delta encoding* technique, as discussed in the next section.

4.1 Delta Encoding

In the delta encoding technique, we transmit data at a pre-defined regular interval; however, only the difference between the last data value and current data value is sent. Although straightforward, this technique provides the quality requirement of sending the "I'm alive" signal periodically, while still being efficient in terms using the energy for data transmission.

The algorithm for the delta encoding technique is listed in Figure 2. The input to this algorithm consists of a series of N packets of different types of data. This data may be GPS coordinates, vital signs such as heart beat, body temperature and so on. Consider that $P[i]; i = 1, 2, ..., N$ and $Q[i]; i = 1, 2, ..., N$ represent the current and last values of the data packets. The delta encoding algorithm goes over all the N data packets, computes the difference between the data in the previous packet and the data in the new packet ($Q[i] - P[i]$, line 2), encodes it according to the data type (lines 3–8) and inserts it into the delta encoded packet series δ_{enc} (line 9).

Data type plays an important part in delta encoding, since it determines the precision with which we have to encode the data. This is explained for GPS data in the next section. Note that also in the case of the PMC-MR algorithm the compression of different time-series can be applied and the type determine different error tolerance for each type of data.

4.2 Tracking GPS Data

We use the example of GPS (Global Positioning System) data to exemplify the use of the technique. Consider that we want to monitor the movement of people in real-time; a personal GPS device that they carry transmits their coordinates – latitude and longitude – at regular intervals. These coordinates range from 0.0000 to 179.9999 degrees with three digits for the decimal part and four for the fractional part (e.g, 128.4523 degrees). We use twelve bits for encoding three decimal digits (128 is encoded as 0001 0010 1000 for instance), sixteen bits for the fractional digits and four more bits for denoting the decimal

Decimal point	Meaning
1010	negative
1011	positive

Table 1. **Decimal point nibble and its meaning. The fact that the nibble is bigger than 9 identifies it as the decimal point digit.**

Difference Value	Encoding
0.0001	1010 0001
-0.0102	1011 0010 0000 0001
0.2000	1010 0000 0000 0010

Table 2. **Difference value and their respective encoding. The fractional digits are sent from the most least significant to the least significant. The decimal digits are not even sent corresponding to the GPS pattern as will be shown in the result section.**

point and the sign of the difference. This forms a four byte packet (12+16+4=32 bits) and the baseline case for our experiments.

However, movement on the earth's surface translates to small changes in terms of latitude and longitude. We find that usually these changes are in the range of 0.001 degrees. Hence, in our delta encoding approach, we send only the fraction digits after the decimal point, instead of sending the full four byte packet. We also use four bits for encoding each decimal digit (as in the baseline case), four bits for encoding the decimal point and the sign and four bits for each fractional digits as well. As we will see in the results section, the difference between two GPS coordinates is always very small yielding a good compression ratio. This encoding is implemented by the function *gps_encode* on the δ computed previously.

Table 1 shows the encoding for the decimal point nibble. Besides delimiting the boundaries between the decimal and fractional parts of the value, it also carries the sign of the difference between the pairs of GPS data. The simple fact that the nibble is bigger than 9 differentiates it from the other digits. Table 2 exemplifies the encoding of typical values for the GPS data.

The difference technique is a simple technique inspired by the motion estimation algorithm in the MPEG-1 video compression protocol. Our results, however, demonstrate the effectiveness of this technique in reducing the power usage in a sensor network.

5 System Architecture

To model a sensor network, we use a network of 50 personal hand-held devices able to communicate with each other and the fixed wire-line network over the wireless 802.11b interface. A node in this network is a Compaq iPAQ device (206 Mhz StrongARM 32-bit processor, 32 MB RAM, 16 MB ROM). Each node is equipped with an expansion pack consisting of two PCMCIA slots which were fitted with

a Cisco AiroNet wireless LAN adapter and a TeleType GPS receiver to localize the position of the node.

These devices are used to send a steady stream of information such as the GPS location of the user to a centralized server. We apply our techniques to reduce the communication bandwidth of each iPAQ device and thereby conserve its battery power.

Figure 3 shows a schematic representation of the system architecture. The QUASAR core comprises of a Java based server which runs as a daemon on one or more machines in the infrastructure network on a pre-determined port. The server listens for client connections on this port and creates a new socket connection every time an iPAQ node connects to the server. An iPAQ node connects to the server via a handshake message which establishes the identity of the device and registers it. The server responds to control messages from the client and provides data in response to queries.

Each iPAQ node sends information such as remaining battery power, network bandwidth utilization, memory utilization, GPS data and web cache data to the server at regular intervals. If a client were to deliver every single data item produced it would result in a huge drain of battery power thereby reducing the operation time of the client. In order to reduce communication bandwidth, we use online compression algorithms to send an approximate representation of the time series data to the server. In short, we approximate the data between two time instants by a constant value, ensuring that the approximate representation is of some pre-defined bounded quality. As soon as the client needs to send data to the server, it tries to open a socket connection with the server and sends a fresh value. This makes the case for a quality driven communication. We also apply delta encoding whenever the periodicity is a crucial factor and we refer to this case as a reliability driven communication.

One may have two types of updates in this scenario, e.g a value initiated update which is initiated by the client or a query initiated update which may be initiated by a consumer which wants to query the statistics of a particular iPAQ machine with a stringent quality requirement. This mode of quality cognizant data management may be particularly useful in case the producer data fluctuates very rarely since it radically minimizes the communication bandwidth of the device. An example of such a scenario would be GPS data of an individual who is stationary or moves about within a small perimeter most of the time, or the remaining battery power of a device which is connected to a power outlet for the majority of the time.

5.1 Application of Poor Man's Compression

The iPAQ device monitors the range (minimum and maximum) of its input. While this range is less than or equal to $2\varepsilon_{capt}$, it updates the endpoints of this range if needed. When the range exceeds $2\varepsilon_{capt}$ at time n, then the segment ending at $n-1$ with a constant value equal to the average of the maximum and minimum values within this time segment i.e $((min+max)/2, n-1)$ is sent to the server. The algorithm then tries to compress the next set of samples starting at time n. This ensures that the archived values stay within ε_{capt} of the actual values at the producer.

Since the mean error produced by PMC-MR, which has been described above, may sometimes be large if the distribution of the values is skewed, a slight modification may be made to use the mean of the points in each time segment as the constant of the segment. Thus, values are sampled until the mean of the points seen so far is more than ε_{capt} away from the minimum or the maximum of the observed points. We observe that the PMC algorithms produce a sequence of compressed segments. These can be forwarded immediately or aggregated into packets for network transmission depending on the buffer size allocated in the iPAQ device.

6 Experimental Results

We briefly describe the experimental results of our compression algorithm on GPS data collected from an iPAQ device. For the purpose of this paper, we present results for the GPS data of only one user carrying an iPAQ device in our mobile testbed. We found that this data is representative of the data collected from all the other users. The track followed in the experiment has been showed in Figure 4. Each GPS receiver sends a steady stream of data comprising of the longitude, latitude indicating the current position of the user and the corresponding timestamp. Instead of sending each reading of the GPS receiver(the exact number of samples over the period of time was 1800), we compressed the time series using our PMC-MR algorithm as described in Section 3, on the longitude and the latitude data generated by the iPAQ GPS. The compression ratio achieved using varying level of tolerance or precision at the server have been shown in Figures 4. In the figures, K is the number of time series segments sent by the iPAQ device, while n is the number of actual samples(uncompressed data). Thus the lower the value of K/n, the higher is our compression ratio. As is evident from figure 4, we obtain a high degree of compression at reasonable spatial tolerance levels. This may result in significant energy savings at the iPAQ device since it drastically reduces the number of messages sent to the server.

Figure 5 shows the results of sending data throught the IPAQ using neither compressing nor delta encoding on the data (Baseline case), only compression (with $\varepsilon_{capt} = 0.025$ yielding a compression rate of 0.14), and only delta encoding. The graphics show the size of the GPS info sent over time. For the Baseline case, the messages are sent in a periodic basis. 32 bits are used for the longitude and 32 bits for the latitude for a total of 1800 messages. In the compression case, since the assumption that the periodicity of the messages is not relevant, only 412 messages are transmitted and each message can have either 32 or 64 bits, depending on whether either latitude or longitude are sent or both are sent. This will depend on when the compression algorithm "decides" to send the data. If the time coincides for both latitude and longitude both are sent (64 bits). Otherwise each one of them will be sent separately (32 bits). And finally in the delta encoding technique also 1800 messages are sent but the size of each message varies from 8 bits to 24 bits yielding a good compression ratio and still meeting the requirement of periodicity of the messages. For the sake of comparison, Figure 6 shows the curves for the number of bits accumulated over time for the three schemes.

We use the same energy model as the one used by Bhardwaj *et al.* [1]. In this model the energy consumption is a function of the number of messages sent, the size of each message and the distance of the wireless node relative to the base station when the transmission takes place. The model is given by the formula $E_{tx} = N_{tx}(\alpha_{11} + \alpha_2 d(n_1, n_2)^n)L$ where E_{tx} is the transmission energy for the sensor. N_{tx} is the number of transmitted messages, $d(n_1, n_2)$ is the distance between the transmitting node and the receiving node, n is the path loss index, L is the packet size and the αs are positive constants. For the sake of simplicity, we ignore the distance parameter setting it fixed to 100. The other parameters are for typical values as described in [14, 1]. These are $\alpha_{11} = 20nJ/bit, \alpha_2 = 10pJ/bit/m^2$ for $n = 2$. Table 3 shows a summary of the number of packet transmitted, the average size of each packet in bits and the energy normalized to the baseline case. We can see that for Delta encoding scheme provides a good tradeoff between saving energy and keeping the periodicity of the messages. For an application that needs the periodicity as a "I'm alive" signal the Delta encoding scheme fits perfectly even though it gives a energy saving 22% worse than the compression algorithm. However, compared to the baseline case, the Delta Encoding scheme still gives 28% energy savings.

Algorithm	Messages	Size (bits)	Normal. Energy
Baseline	1800	64	1
Delta Encoding	1800	11.8	0.7284
Compression	412	40.9	0.5740

Table 3. **Summary of results applying the three algorithms. The number of messages sent, the average size of each message in bits and the energy normalize to baseline are shown.**

7 Conclusion and Future Work

We have addressed the need for reducing communication power in a sensor network by reducing the amount of data communication. We presented two techniques – a compression technique and a delta encoding technique – one of which sends data less frequently by approximating data over a period of time and the other sends fewer data packets during each communication. The compression technique is guided by a user-defined quality measure and is useful for monitoring sensor data in applications where data changes within a pre-defined range need not be reported to the server. The delta encoding technique is useful when regular updates are required to inform the server that the node is alive. Results from experiments performed using a mobile PDA-based sensor network demonstrate that we can reduce the data traffic by 86% (for $\varepsilon_{capt} = 0.025$) and 80% for the compression and delta encoding techniques respectively and the energy consumed in communication by 43% and 22% compared to the baseline case. In ongoing work, we are developing decision mechanisms that can switch between different data encoding schemes depending on energy availability.

References

[1] M. Bhardwaj and A. Chandrakasan. Bounding the lifetime of sensor networks via optimal role assignments. In *Proceedings of INFOCOM 2002*, pages 1587–1596, June 2002.

[2] A. Cerpa, J. Elson, D. Estrin, L. Girod, M. Hamilton, and J. Zhao. Habitat monitoring: Application driver for wireless communications technology. In *ACM SIGCOMM Workshop on Data Communications in Latin America and the Caribbean*, April 2001.

[3] Y. Chen, G. Dong, J. Han, B. W. Wah, and J. Wang. Multi-dimensional regression analysis of time-series data streams. In *VLDB*, 2002.

[4] C.Olston, B. Loo, and J.Widom. Adaptive precision setting for cached approximate values. In *ACM SIGMOD*, 2001.

[5] P. Deolasee, A. Katkar, A. Panchbudhe, K. Ramamritham, and P. Shenoy. Adaptive push-pull: disseminating dynamic web data. In *International World Wide Web conference on World Wide Web*, 2001.

[6] D. Estrin, D. Culler, K. Pister, and G. Sukhatme. Connecting the physical world with pervasive networks. In *Pervasive Computing*, 2002,.

[7] P. Juang, H. Oki, Y. Wang, M. Martonosi, L.-S. Peh, and D. Rubenstein. Energy-efficient computing for wildlife tracking: Design tradeoffs and early experiences with zebranet. In *ASPLOS*, 2002.

[8] E. J. Keogh, K. Chakrabarti, S. Mehrotra, and M. J. Pazzani. Locally adaptive dimensionality reduction for indexing large time series databases. In *ACM SIGMOD Conference*, 2001.

[9] D. Korn and K.-P. Vo. Engineering a differencing and compression data format. In *In Proceedings of the Usenix Annual Technical Conference*, pages 219–228, June 2002.

[10] I. Lazaridis and S. Mehrotra. Capturing sensor-generated time series with quality guarantees. In *To appear at IEEE International Conference on Data Engineering*, 2003.

[11] J. C. Mogul, F. Douglis, A. Feldmann, and B. Krishnamurthy. Potential benefits of delta encoding and data compression for HTTP. In *SIGCOMM*, pages 181–194, 1997.

[12] G. Pottie and W. Kaiser. Wireless integrated sensor networks. In *Communications of the ACM*, 2000.

[13] V. Raghunathan, C. Schurgers, S. Park, and M. B. Srivastava. Energy-aware wireless microsensor networks. In *IEEE Signal Processing Magazine*, March 2002.

[14] A. Sinha and A. Chandrakasan. Dynamic power management in wireless sensor networks. *IEEE Design and Test of Computers*, 18(2), March 2001.

[15] M. Srivastava, R. Muntz, , and M. Potkonjak. Smart kindergarten: Sensor-based wireless networks for smart developmental problem-solving environments. In *The Seventh Annual International Conference on Mobile Computing and Networking*, 2001.

Procedure PMC-MR

INPUT:

Time series $S = \langle s[1],\ s[2],\ \ldots \rangle$, tolerance $\varepsilon_{capt} > 0$.

OUTPUT:

Compressed time series PCA(S) within-ε_{capt} of S.

(1) PCA(S) $\leftarrow \langle\ \rangle$;
(2) n $\leftarrow$ 1;
(3) m $\leftarrow$ s[n];
(4) M $\leftarrow$ s[n];
(5) **while** S.hasMoreSamples()
(6) **if** $\max\{M,\ s[n]\} - \min\{m,\ s[n]\} > 2\varepsilon_{capt}$
(7) **append** $(\frac{M+m}{2},\ n-1)$ **to** PCA(S);
(8) m $\leftarrow$ s[n];
(9) M $\leftarrow$ s[n];
(10) **else**
(11) m $\leftarrow \min\{m,\ s[n]\}$;
(12) M $\leftarrow \max\{M,\ s[n]\}$;
(13) **end**;
(11) n $\leftarrow$ n+1;
(12) **end**;
(13) **append** $(\frac{M+m}{2},\ n-1)$ **to** PCA(S);

Figure 1. **Poor Man's Compression-MidRange (PMC-MR) Algorithm**

Procedure DELTA-ENCODING

INPUT: Current and Previous Data Values: P and Q

OUTPUT: Delta encoded packet δ_{enc}

(1) **for** $i \leftarrow 1$ to N $\{$

(2) $\quad \delta \leftarrow P[i] - Q[i]$

(3) $\quad$ **switch** $Type(P[i])$ $\{$

(4) $\quad\quad$ **case** *GPS*: $\delta' \leftarrow gps\_encode(delta)$

(5) $\quad\quad$ **case** *VITALS*: $\delta' \leftarrow vitals\_encode(delta)$

(7) $\quad\quad \ldots$

(8) $\quad \}$

(9) $\quad \delta_{enc} \leftarrow \delta'$

(10) $\}$

Figure 2. **Delta Encoding Algorithm**

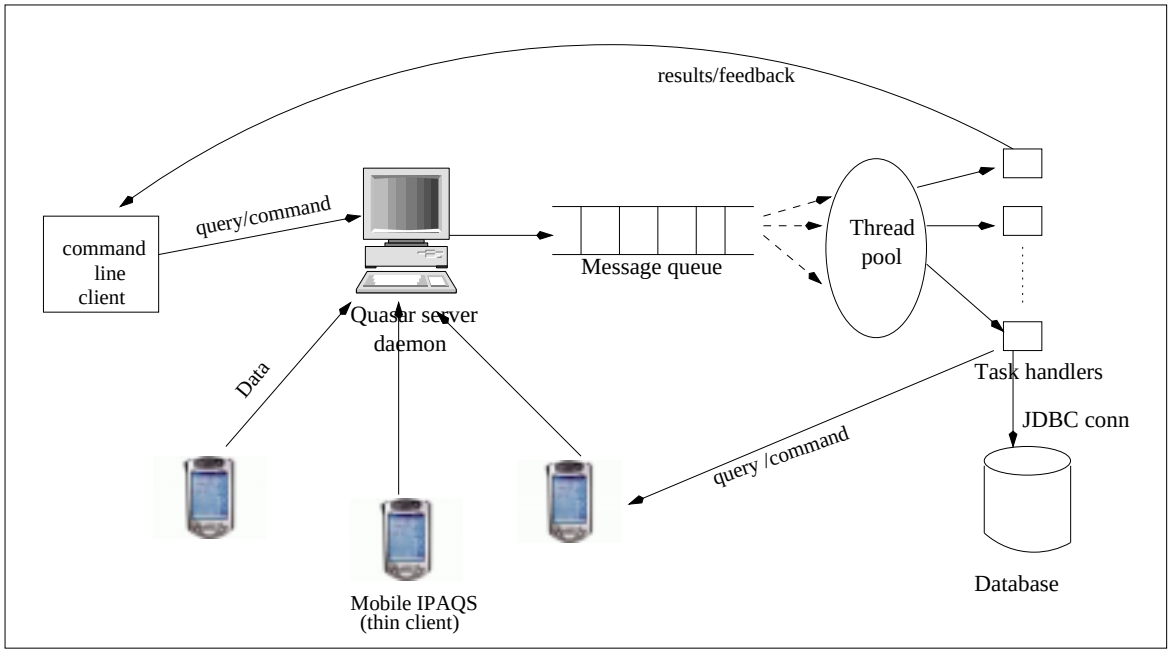

Figure 3. **System Architecture of our iPAQ-based Sensor Network**

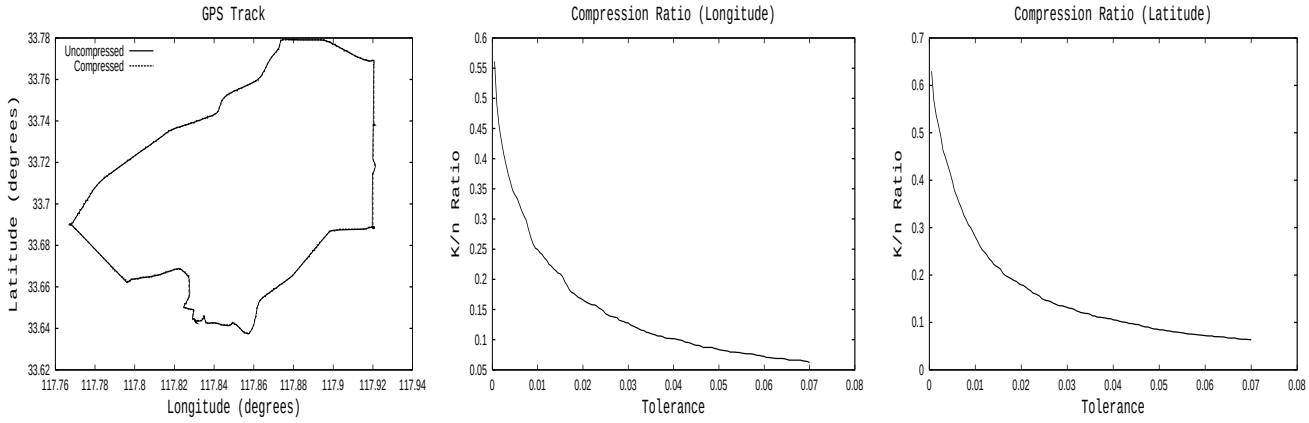

Figure 4. **Compression experiments on GPS data. For the compression ratio graphics** $\varepsilon_{capt} = 0.025$

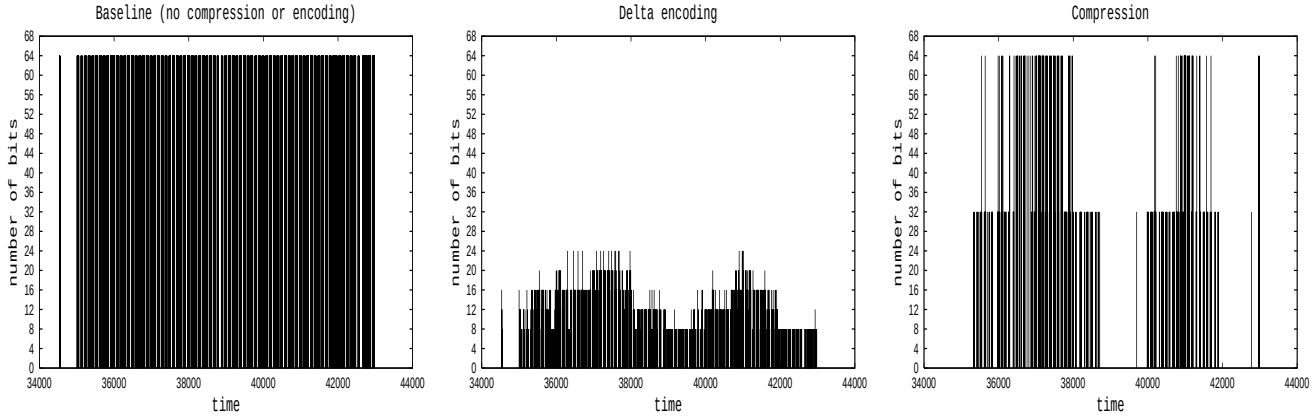

Figure 5. **Baseline, Compression and Delta Encoding results for GPS data**

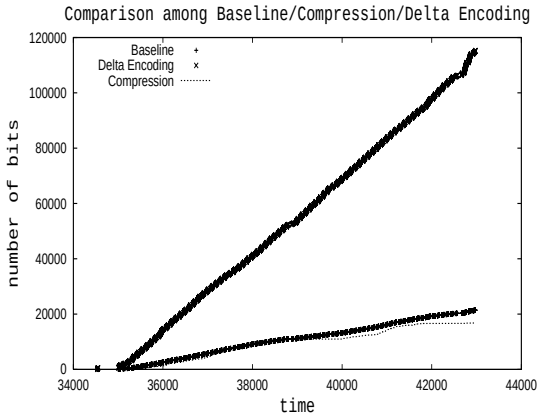

Figure 6. **Comparison on the accumulated number of bits sent over time**