SPASS: Scalable and Energy-efficient Data Acquisition in Sensor Databases [*]

M. H. Ali Walid G. Aref Cristina Nita-Rotaru
Department of Computer Science, Purdue University
{mhali, aref, crisn}@cs.purdue.edu

ABSTRACT

Scalability and energy management issues are crucial for sensor network databases. In this paper, we introduce the *Sharing and PArtitioning of Stream Spectrum (SPASS)* protocol as a new approach to provide scalability with respect to the number of sensors and to manage the power consumption efficiently. The spectrum of a sensor is the range/distribution of values read by that sensor. Close-by sensors tend to give similar readings and, consequently, exhibit similar spectra. We propose to combine similar spectra into one global spectrum that is shared by all contributing sensors. Then, the global spectrum is partitioned among the sensors such that each sensor carries out the responsibility of managing a partition of the spectrum. Spectrum sharing and partitioning require continuous coordination to balance the load over the sensors. Experimental results show that the *SPASS* protocol relieves a sensor database system from the burden of data acquisition in large-scale sensor networks and reduces the per-sensor power consumption.

1. INTRODUCTION

The recent advances in large-scale sensor network technologies enabled the deployment of a huge number of sensors in the surrounding environment. Each sensor consists of a small node with sensing, computing, and communication capabilities. Due to the limited processing capabilities of sensor nodes, the sensor readings are minimally processed at the sensor network level. Then, the sensor data is transmitted through a multi-hop communication route to a centralized sensor database system for further processing. As sensor networks get larger, sensor databases are burdened with the massive amount of data that is streamed out of the sensors. Recent research focuses on sampling [10, 13, 17, 19, 20], communication [7, 15, 16], and query processing [5, 9, 11, 18, 25] techniques for sensor data. *Scalability* with respect to the size of the sensor network and energy-management issues have been major challenges in these techniques.

[*]This work was supported in part by the National Science Foundation under Grants IIS-0093116, IIS-0209120, and 0010044-CCR.

Sensors are deployed densely in the space to increase the reliability of monitoring the surrounding environment. The dense distribution of sensors achieves reliability via redundancy to decrease the likelihood of losing sensor readings. However, much redundancy results in *overhearing* the environmental measurements and, consequently, overloading the data stream management system. Things get worse if sensors indulge in a *correlated behavior* where sensors transmit the same readings successfully while other readings are not delivered by any sensor. As a result, the system receives duplicates of the same value while losing other values totally.

A crisp observation of the sensor data reveals similarities in the distribution of the sensor readings that are coming from close-by sensors. This fact is understandable because close-by sensors are exposed to the same environmental conditions. We refer to the distribution of readings from a sensor as the sensor's *spectrum* (A formal definition of the sensor's *spectrum* is given in Section 2). The similarities in the sensors' spectra bear redundant information allowing a wide room for optimizations.

1.1 Approach

Motivated by similarities in the spectra of sensors, we propose to cluster close-by sensors into groups as illustrated in Figure 1. The spectra of sensors in the same group are merged to form one global spectrum. The global spectrum is partitioned among the sensors in the group to assign each sensor the responsibility of transmitting a partition of the spectrum to the sensor database. A sensor gives its own partition the highest priority and processes other partitions based on the availability of resources.

To detect similarities in the sensors' spectra, we push some of the data stream management system functionalities to the sensor network level. In particular, we move the summary manager [1] in part from the data stream management system to the sensor network level to provide early summarization of the sensor data and to discover the associated sensor spectrum. In general, summaries at the core of the data stream management system reduce the processing cost by providing approximate answers in lieu of the exact ones. We propose to shift some of the summarization tasks to the sensor network level to provide an early approximation of the sensor data and to reduce the transmission cost as well as the processing cost. The proposed protocol (*SPASS*) saves energy by reducing the number of transmitted readings across the network. The reduction in the number of transmitted readings comes with the advantage of reducing the load over the sensor database and achieves scalability. Example applications that benefit from our work include surveillance, tracking, and environmental monitoring [12, 21, 22].

1.2 Contributions

The contributions of this paper can be summarized as follows:

1. We propose a new data acquisition protocol, the *Sharing and PArtitioning of Stream Spectrum (SPASS)* protocol, that acquires a *faithful* representation of sensor data and handles energy management and scalability issues.

2. We implement the proposed *SPASS* protocol inside *Nile* [14], a research prototype that is currently being developed at *Purdue University*. *Nile* extends relational database management systems with the data streaming functionalities.

3. We provide experimental evidence that the *SPASS* protocol represents sensor data faithfully and enhances the performance of sensor databases in terms of scalability and power consumption.

The rest of the paper is organized as follows: Section 2 gives the definition of the stream *spectrum* and its properties. Section 3 introduces the proposed *SPASS* protocol while Section 4 presents a variation of the *SPASS* protocol that is optimized for adaptivity. Experimental results that are based on a real implementation of the proposed *SPASS* protocol inside *Nile* are presented in Section 5. Section 6 highlights related work. Finally, the paper is concluded in Section 7.

2. STREAM SPECTRUM AND ITS PROPERTIES

The term *spectrum* refers to the distribution of a physical characteristic. The *spectrum* of a certain characteristic is the value ranges over which this characteristic is distributed. For example, the spectrum of the visible light is the continuous frequency ranges of its components. In the context of data streams, we define broadly the stream spectrum as the value ranges from which the stream tuples are drawn.

Before we proceed to a formal definition of the *stream spectrum*, we discuss the underlying structure of sensor databases (Section 2.1) and how the stream summaries can be partially pushed to the sensor-network level to help derive the stream spectrum of each sensor (Section 2.2). We give a formal definition of the *stream spectrum* and how it can be derived from various summarization techniques in Section 2.3. By the end of this section, we highlight the benefits of merging the individual spectra of close-by sensors to form one global shared spectrum. Then, we provide mathematical bounds on the amount of savings that can be obtained by sharing the global stream spectrum (Section 2.4).

2.1 Sensor Network Support Layer

A data stream management system (*DSMS*) comes at the core of a sensor database system (Figure 1). The *DSMS* provides a pipelined query execution of continuous queries over the data streams that are generated by the sensors. We extend data stream management systems with an additional layer, the *Sensor Network Support Layer (SNSL)*, to support the functionalities of sensor networks. The main purpose of the *Sensor Network Support Layer (SNSL)* is (1) to reduce the processing load at the system's side for the sake of scalability, and (2) to reduce the power consumption at the sensors' side.

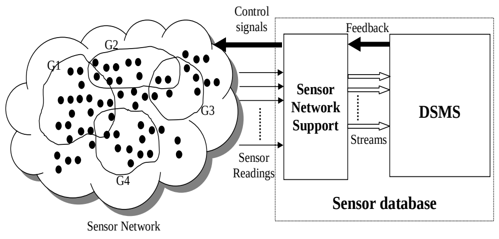

Figure 1: Sensor network support layer.

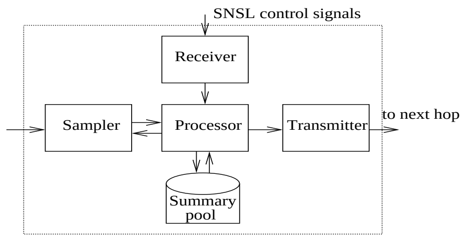

Figure 2: Basic components of a sensor.

The *SNSL* accepts feedback from the *DSMS* about the sensor data and sends control signals to the sensors to control their behavior. The *SNSL* instructs the sensors on how to sample, aggregate, and transmit their readings. As one of the *SNSL* components, we propose the *Sharing and PArtitioning of Stream Spectrum (SPASS)* protocol to provide scalable and energy-efficient acquisition of sensor readings that *faithfully* represent the sensor-network data.

2.2 Sensor-network Level Summarization

We propose two levels of summarization: sensor-network level summarization and system level summarization. Sensor-network level summarization guides the sensors to the tuples that are worth transmission while system level summarization guides query processing. In our work, sensor-network summarization is conducted at each sensor to capture the sensor spectrum. Guided by the sensor spectrum, we prioritize the stream tuples for transmission.

Each sensor has four basic components, as illustrated in Figure 2. The functionalities of these components can be summarized as follows:

1. The *sampler* has the capability of sensing the surrounding environment.

2. The *receiver* receives control signals from the *sensor network support layer (SNSL)* and forwards these signals to the processing unit.

3. The *processor* performs various tasks based on the *SNSL* signals. For example, the processor instructs the sampler on how to control its sampling rate. Data aggregation and filtration are performed at the processor to reduce data size. In addition, the processor performs the following *SPASS* functions: (a) build summaries over the incoming stream of tuples, (b) generate

the stream spectrum from the summaries, (c) share the spectrum among other sensors, and (d) prepare tuples to be sent by the transmitter.

4. The *transmitter* is responsible for the physical transmission of (a) sensor data, (b) sensor spectrum, and (c) information about the sensor's transmission rate. The information about the transmission rate helps the *SNSL* coordinate the partitioning of the sensors' shared spectrum based on the load of each sensor.

2.3 Definition of a Stream Spectrum

In this section, we provide a formal definition of the stream spectrum and give examples of how the stream spectrum can be derived from stream summaries. The stream spectrum can be defined as follows:

DEFINITION 1. *For a data stream S that consists of an infinite tuple sequence $\{x_1,\ x_2,\ x_3,\ \cdots\}$ that arrive at time instants $\{t_1,\ t_2,\ t_3,\ \cdots\}$, respectively, a stream spectrum SS, at time instant τ, is a finite set of stream representatives $R = \{r_1, r_2, \cdots, r_L\}$ such that R is obtained using a summarization function $\phi(x_{\tau-w+1}, x_{\tau-w+2}, \cdots, x_\tau) \rightarrow R$, where w is a sliding time-window. For any stream tuple x, $\exists\ r \in R$ such that $M(x) \rightarrow r$, where M is a mapping function that maps a stream tuple to one of the stream representatives.*

The stream spectrum is generated using a summarization function ϕ that captures the stream behavior over the most recent time-window w and produces a finite set of stream representatives. Notice that the stream spectrum is associated with a time instant τ because the stream spectrum may change with the slide of the summarization function window. A newly incoming tuple in the stream updates the stream summaries and is mapped to one of the stream representatives. All tuples that map to the same representative are processed and transmitted in the same way. A stream representative provides a unified processing and transmission interface for all tuples that map to that representative.

To give an example of how the stream spectrum can be extracted from stream summaries, we first consider *histograms* as our summarization technique. In histograms, the stream representatives are the histogram buckets. The summarization function ϕ updates the bucket frequencies based on the incoming tuples over the last w time-window. The tuple is mapped by the mapping function M to the bucket it falls in ($M(x) \rightarrow bucket(x)$). All the tuples in the same bucket are treated, i.e., processed and transmitted, uniformly.

Maintaining the *topk* list of a stream is another summarization technique. In the *topk* approach, the stream is summarized using its k most frequent elements. The summarization function ϕ updates the *topk* list based on the tuple count over the last w time-window. The stream representatives (R) are the most frequent k elements themselves plus the *NULL* value. The mapping function M maps an element to itself ($M(x) \rightarrow x$) if its frequent or ignores it otherwise ($M(x) \rightarrow NULL$).

Similarly, the stream spectrum can be extracted from other forms of stream summaries by defining their corresponding ϕ and M functions. The proposed *SPASS* protocol is general and can accommodate various summarization techniques. However, the mathematical analysis and the experimental evaluation assume that the sensor data follows the *Zipfian* distribution and is summarized using the *topk* list approach as presented in [8].

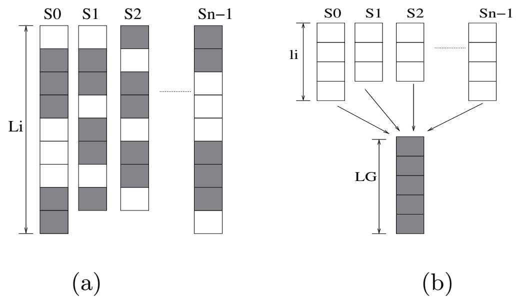

Figure 3: Individual sensor spectra versus a global sensor spectrum.

2.4 Global Stream Spectrum

In real life, close-by sensors are exposed to similar environmental conditions. Hence, close-by sensors produce readings that share similar distributions over almost identical value ranges. The closer a sensor to its neighbors, the more correlated its spectrum to the neighbors' spectra. The *correlation coefficient*, ρ_{ij}, assesses how much the readings of a sensor (S_i) vary in response to the variation in the readings of another sensor (S_j). Equation 1 gives the correlation coefficient between the readings of two sensors, S_i and S_j, where μ and σ are the mean and the standard deviation of the stream tuples, respectively.

$$\rho_{ij} = \frac{E[(X_i - \mu_i)(X_j - \mu_j)]}{\sigma_i \sigma_j} \tag{1}$$

A *correlation matrix* is a two-dimensional matrix that records the *correlation coefficient* between every two sensors, S_i and S_j. By observing the correlation matrix of real sensor data, we find out that the correlation matrix contains *ones* along the diagonal because a sensor is fully correlated with itself. Also, the correlation coefficient between two sensors gets higher as they get closer to each other. The high correlation among a group of close-by sensors over time is our target area of optimization.

Figure 3a shows the spectra of a group of n sensors. Each sensor has a spectrum of length L_i. A gray box represents a value that appears in the spectra of *all* sensors. Figure 3b suggests to maintain only one global spectrum of length L_G that is shared by all sensors. The global spectrum accommodates all the items that appear in all sensors. Each sensor sees a spectrum of length l_i that accommodates its private non-shared spectrum elements plus the shared global spectrum.

Combining the common parts of the spectra of various sensors into one global spectrum reduces the overhead of processing the same value at different sites. The *stream compression ratio* (SCR) is the amount of savings achieved by merging the sensors' common spectra into one global spectrum. The (SCR) is given by Equation 2. Equation 2 calculates the ratio of the global spectrum size plus the sizes of private spectra to the summation of the sizes of the individual non-shared spectra, then subtracts this value from 1 to yield the compression ratio.

$$SCR = 1 - \frac{\sum_{i=1}^{n} l_i + L_G}{\sum_{i=1}^{n} L_i} \tag{2}$$

The *stream compression ratio* (SCR) is of great significance because it denotes the amount of processing overhead that can be distributed and balanced among sensors. In the remainder of this section, we give a mathematical bound on the SCR that can be achieved for sensor data that follows the *Zipfian* distribution. We derive the sensors' spectra from summaries that are based on the most frequent item list (the *topk* list) as discussed in [8]. In this case, the shared spectrum will contain the k most frequent elements that are common to all sensors (Equation 3).

$$L_G = k \tag{3}$$

For simplicity, assume that (1) sensors are close enough to each other such that they have a common k most frequent item list, and that (2) sensors maintain spectra that are of the same length (i.e., $L_i = L_j \ \forall i, j$, and consequently, $l_i = l_j$ because $l_i = L_i - L_G$). The SCR in Equation 2 reduces to Equation 4.

$$SCR = 1 - \frac{nl + L_G}{nL} \tag{4}$$

The technique in [8] captures the k most frequent items using a list of length L such that the most frequent item that is *not* captured in the list (element number $L + 1$) occurs with a frequency that is less than $(1 - \epsilon)$ of the frequency of the k most frequent element (Equation 5).

$$f_{L+1} < (1 - \epsilon) f_k \tag{5}$$

To capture the k most frequent element, we have to maintain a list of length $L = O(k)$. In particular L is given by Equation 6, where z is the *Zipfian* distribution parameter.

$$L = \frac{k}{(1 - \epsilon^{\frac{1}{z}})} \tag{6}$$

The non-shared part of the stream spectrum at each sensor, l, is given by subtracting the shared spectrum length from original spectrum length as in Equation 7.

$$l = L - L_G = \frac{k}{(1 - \epsilon^{\frac{1}{z}})} - k \tag{7}$$

Substitute for both L (Equation 6) and l (Equation 7) in Equation 4, we obtain Equation 8 that expresses the *stream compression ratio* in terms of the number of sensors (n), the summarization technique parameter (ϵ), and the *Zipfian* distribution parameter (z).

$$SCR = \frac{n - 1}{n} (1 - \epsilon)^{\frac{1}{z}} \tag{8}$$

Notice that the SCR that is obtained in practice can be less than the value of Equation 8 because sensors may not share all the *topk* elements during the life time of the experiment.

3. THE SPASS PROTOCOL

The *SPASS* protocol coordinates the sharing and the partitioning of the global spectrum among a cluster of close-by sensors. Sensors may be clustered by the nature of the problem. For example, sensors that read the temperature of the same room are exposed to similar conditions and are considered one cluster. Otherwise, clustering techniques that aim at minimizing the power consumption due to packet routing among sensors are deployed to cluster the sensors, e.g., [2, 4]. In our work, we are interested in sharing the spectrum of sensors that fall within the same cluster. This makes the clustering problem orthogonal to our work. Any clustering technique can be applied as a preprocessing phase to our protocol. For the sake of implementation, we use the *HEED* clustering technique as presented in [26].

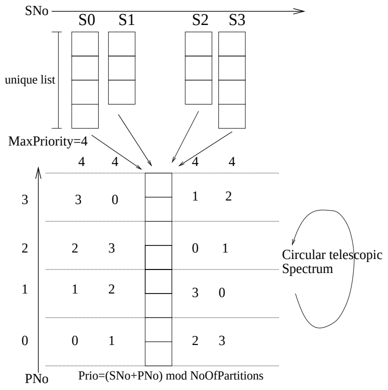

Figure 4: An example circular telescopic spectrum for four sensors.

Clustering techniques select a dedicated sensor among the sensors that fall in the same cluster to be the cluster head. All communication messages that go out of a sensor are forwarded to the cluster head as their next hop. The cluster head is responsible for the coordination among the sensors in its cluster as well as routing their messages to the centralized sensor database.

The goal of the *SPASS* protocol is to maintain a global spectrum over each cluster of sensors as shown in Figure 4. Each sensor maintains a small list of its unique items that are not shared among other cluster members. This portion of the spectrum is given the highest priority, *MaxPriority*, because no other sensors are expected to transmit these items. The shared spectrum is divided into n partitions of equal sizes (i.e., shares) where n is the number of sensors in the cluster. Each sensor takes the responsibility of transmitting one share. The sensor gives its own share the next highest priority, *MaxPriority-1*, and reduces the priority of other sensors' shares by one in a circular fashion. The priority of partition PNo at sensor SNo is given by Equation 9.

$$P = (SNo + PNo) \quad mod \quad n \tag{9}$$

We refer to the global spectrum as presented in Figure 4 by the term *circular telescopic spectrum* because each sensor processes its own share and extends the processing *telescopically* to other partitions based on the availability of time. The *scope* of a sensor is the average depth of items being transmitted by that sensor. The depth of an item is the difference between the item's index in the shared spectrum vector and the index of the beginning of the sensor's share, given the circular direction of movement. The *scope* parameter of a sensor provides information about how much of the global spectrum is covered by that sensor.

The *SPASS* protocol is divided into two major components: (1) the *summarize and transmit* procedure, which is placed at all sensor nodes to build their individual spectrum and to control the transmission of their sensor readings, and (2) the *share and partition* procedure, which is placed at the cluster head to form the shared global spectrum of the cluster and to coordinate the partitioning of the global spectrum among sensors.

Figure 5 summarizes the processing at each sensor node. A sensor either (1) generates a sensor reading or (2) receives a global spectrum from the cluster head. Upon receiving a new reading, the sensor uses this reading to update the

```
procedure Summarize and Transmit
Input: (1) a stream of sensor readings $x_1$, $x_2$, $x_3$, $\cdots$ and (2) A global spectrum (GS)
Output: continuously maintain (1) the sensor individual spectrum and (2) the priority transmission queue.
Description:
Upon receiving a sensor reading $x_i$
    1. UpdateSummaries($x_i$).
    2. UpdateSpectrum().
    3. if $|NewSpectrum - OldSpectrum| > \alpha$ then
       transmit the spectrum to the cluster head.
    4. P=GetPrio($x_i$)
    5. Transmit($x_i$,P)
Upon receiving the global spectrum
        update the circular teslescopic spectrum
```

Figure 5: The SPASS protocol at each sensor node.

summaries (Step 1) and update the local individual spectrum based on the change in the summaries (Step 2). If the distance between the new spectrum and the spectrum at the cluster head exceeds α, a fresh copy of the spectrum needs to be transmitted to the cluster head (Step 3). The sensor probes the spectrum to find the partition where the sensor reading falls and, consequently, retrieves its transmission priority (Step 4). The sensor places the reading in the transmission priority queue to compete for the transmission bandwidth based on the priority (Step 5). Upon receiving a new version of the global spectrum, the sensor updates the transmission priorities based on the notion of the circular telescopic spectrum.

Figure 6 describes the role of the cluster head in sharing and partitioning the global spectrum. The cluster head receives from each sensor either (1) a sensor reading or (2) a sensor's local spectrum. When the cluster head receives a sensor reading, it forwards the reading to the next hop on its way to the sensor database. When the cluster head receives a sensor's spectrum, it merges this spectrum with the spectra of other sensors in the cluster to compute the global spectrum (GS) (Step 1). Merging the spectrum is simply to find the common items in *all* sensors' spectra. Then, the common items are partitioned into n partitions of equal sizes (Step 2). Finally, the cluster head updates each sensor with a copy of the shared item list (Step 3).

4. THE SPASS+: AN ADAPTIVE VERSION OF THE PROTOCOL

The *SPASS+* promotes adaptivity by balancing the load among sensors in the same cluster based on their relative loads. In this section, we define the *sensor load* and develop an adaptive technique to partition the global spectrum dynamically among sensors. Each sensor is assigned a share of the spectrum that is inversely proportional to the load over that sensor. We define the sensor load as follows:

Definition 2. *The sensor load is defined to be the total time required to transmit all items that are queued in the sensor's buffer.*

Two major parameters formulate the sensor load: (1) the throughput, which refers to the achieved transmission rate

```
procedure Share and Partition
Input: Given a cluster of n sensors $S_0$, $S_2$, $\cdots$, $S_{n-1}$. Each sensor generates: (1) a stream of readings and (2) an individual sensor spectrum $SS_i$.
Output: (1) the global spectrum of the cluster and (2) the share of each individual sensor.
Description:
Upon receiving a sensor reading
        forward the sensor reading to next hop
Upon receiving a sensor spectrum $SS_i$
    1. MergeSpectrum(GS, $SS_i$)
    2. PartitionSpectrum(GS)
    3. for i=0 to $n-1$ SendtoSensor($S_i$, GS)
```

Figure 6: The SPASS protocol at the cluster head.

in terms of the number of transmitted messages per second and (2) the queue length, which specifies how many items are still queuing in the buffer waiting for transmission. The sensor load is computed as follows:

$$Load = \frac{Queuelength}{throughput} \tag{10}$$

Let Ld_i be the load at sensor S_i. The share of sensor S_i in the global spectrum is calculated at the cluster head using Equation 11. Notice that the more the sensor is loaded, the smaller the share it gets. To achieve this adaptive behavior, each sensor is required to report its load periodically to the cluster head. The cluster head keeps track of the load over each sensor in its cluster and continuously repartitions the spectrum among sensors to balance the load within the cluster.

$$Share_i = \frac{\frac{1}{Ld_i}}{\sum_{j=0}^{n-1} \frac{1}{Ld_j}} \times SpectrumLength \tag{11}$$

SPASS+ requires two major modifications over the *SPASS* protocol. In the *Summarize and Transmit* procedure (Figure 5), each sensor transmits information about its current load periodically to the cluster head. In the *Share and Partition* procedure (Figure 6), the *Partition Spectrum* function (Step 2) is modified to divide the spectrum into non-equal partitions. The length of each partition is given by Equation 11.

5. EXPERIMENTS

In this section, we perform an experimental study to explore the performance of the proposed *SPASS* protocol. Two sets of experiments are conducted. The first set of experiments (Section 5.1) addresses the performance of the *SPASS* protocol in terms of scalability and power consumption. The second set of experiments (Section 5.2) is concerned with the *SPASS* internal parameters, e.g., the *correlation* and the *spectrum compression ratio*, with respect to different cluster sizes. We study three protocols:

1. *SIMPLE*, where each sensor simply transmits, based on the allowed bandwidth, a uniform sample of its own readings to the sensor database.
2. *SPASS*, where the *SPASS* protocol is deployed to manage the transmission of data as described in Section 3.
3. *SPASS+*, where the *SPASS* protocol is optimized for adaptivity as described in Section 4.

Our major measure of performance is *Hist-MSE* (Equation 12), which represents the *mean square error* between the global histogram of all the generated sensor data (at the sensor side) and the global histogram of the transmitted sensor data (at the system side) after they are normalized by the data set size. A global histogram includes the streams coming from all sensors to give a global view of the whole sensor network. We do not care about the individual histogram of each sensor. Instead, we care about the collaboration of sensors to transmit a faithful view of the entire sensor field being investigated.

Let H_1 be the histogram of the original data and let H_2 be the histogram of the transmitted data. Each histogram is an equi-width histogram of n intervals (n is set to 100). H_1 is divided into $H_{11}, H_{12}, \cdots, H_{1n}$ and H_2 is divided into $H_{21}, H_{22}, \cdots, H_{2n}$. Let N_1 be the size of the original data set and let N_2 be the size of the transmitted data set ($N1 \geq N2$). *Hist-MSE* is defined as follows:

$$\textit{Hist-MSE} = \frac{\sum_{i=1}^{n} (\frac{h_{1i}}{N_1} - \frac{h_{2i}}{N_2})^2}{n} \qquad (12)$$

Unless mentioned otherwise, we maintain 1000 sensors uniformly distributed in the space. Each sensor generates a stream of 10,000 tuples where the tuple values follow the Zipfian distribution. The sensors are grouped into clusters, each cluster is of size 5. The interarrival time of sensor data follows an exponential distribution with an average of one second. To model the scarcity of resources, the sensor database is capable of processing a bandwidth that is up to 200 tuples per second. The bandwidth is shared fairly among the 1000 sensors, which means that each sensor is granted to transmit a bandwidth that is up to 0.2 samples every second (i.e., the allowed bandwidth carries around 20% of the sensor readings). In other words, instead of reading a value per sensor every second, we read a value per cluster. All the experiments in this section are based on a real implementation of the *SPASS* protocol inside the *Nile* data stream management system [14]. The *Nile* engine executes on a machine with Intel Pentium IV, CPU 2.4GHZ with 512MB RAM running Windows XP.

5.1 Scalability and Power Consumption

In this section, we study the performance of the proposed *SPASS* protocol with respect to the *Hist-MSE* measure of performance under various conditions of the sensor network. We provide an experimental evidence that the *SPASS* protocol: (1) reduces the power consumption of the sensors, (2) is scalable in terms of the stream rates, and (3) is scalable in terms of the size of the sensor network.

The sensor's bandwidth denotes the maximum number of transmitted messages per unit time. The energy consumption of a sensor decreases with the decrease in its utilized bandwidth. In Figure 7, we vary the total system bandwidth (which is shared among the 1000 sensors) from 100 to 1000 tuples per second and compare the performances of the *SPASS* protocol, its *SPASS+* variation, and the *SIMPLE* protocol. For the same bandwidth, the *SPASS* protocol gives a better representation of the sensors' readings as indicated by the *Hist-MSE* measure of performance. The optimized *SPASS+* protocol gives a better *Hist-MSE* over the *SPASS* protocol because of its capability to balance the load among sensors dynamically. The *SPASS+* protocol reduces the *Hist-MSE* by up to 70% over the *SIMPLE* protocol and by up to 55% over the *SPASS* protocol (at 100 samples per second bandwidth). Notice that as we increase the bandwidth to 1000 samples per second, each sensor transmits its readings completely to the sensor database and *Hist-MSE* drops to zero.

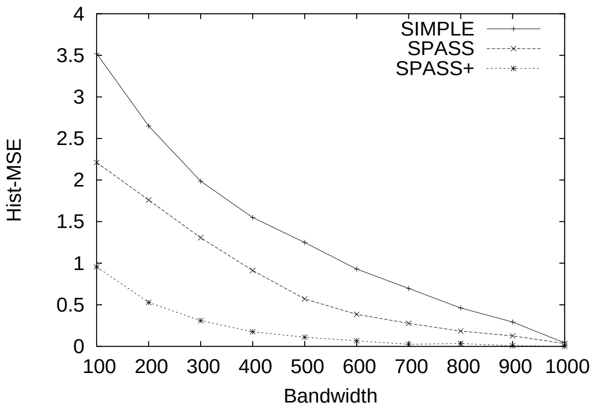

Figure 7: The effect of the allowed bandwidth.

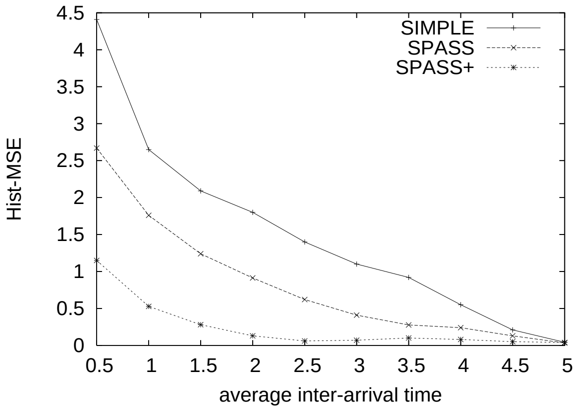

Figure 8: The effect of the stream average interarrival time.

In Figure 8, we control the stream rate by varying its average tuple interarrival time and measure *Hist-MSE* of various protocols. As we increase the average interarrival time, the system load decreases and *Hist-MSE* drops till it reaches zero. Increasing the stream's average interarrival time has a similar effect to increasing the bandwidth because an increased bandwidth implies more system resources while increasing the interarrival time implies less system load.

Figure 9 illustrates the performance of the *SPASS* protocol under various sensor-network sizes. The size of the sensor network is expressed in terms of the number of sensors in the sensor network. We vary the number of sensors in the sensor network from 200 to 2000 sensors. In terms of *Hist-MSE*, the *SPASS* protocol gives a better performance over the *SIMPLE* protocol while its *SPASS+* variation is still capable of providing further reduction in the *Hist-MSE*. As the number of sensors increases, the performance gain of the *SPASS* protocol and its *SPASS+* variation becomes more significant. The *SPASS+* protocol reduces the *Hist-MSE* by up to 65% over the *SIMPLE* protocol and by up to 35% over the *SPASS* protocol (at network of size 2000 sensors).

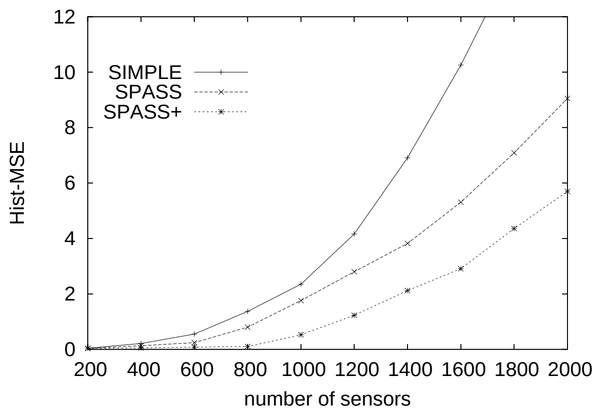

Figure 9: The effect of the number of sensors.

5.2 Cluster Size

In this section, we study the effect of the cluster size on two internal parameters of the protocol, the average *correlation coefficient* between sensor pairs (ρ) and the *spectrum compression ratio* (SCR). The *correlation coefficient* assesses the similarities in the sensors' spectra while the *spectrum compression ratio* estimates how much saving can be obtained by combining their spectra into one global spectrum. We vary the cluster size from 1, which means no clustering is in effect, to 10 sensors per cluster. The *correlation coefficient* (ρ) and the *spectrum compression ratio* (SCR) are calculated offline based on the average of (ρ) and (SCR) in a one-minute sliding window over the sensor data.

Figure 10 gives the effect of the cluster size on the *correlation coefficient* and the *spectrum compression ratio* parameters. The maximum value of the *correlation coefficient* is one, which corresponds to full correlation among sensors. A cluster of size one is fully correlated because one sensor is 100% correlated with itself. As we increase the cluster size, fewer items tend to be shared among *all* streams and, consequently, the *correlation coefficient* decreases.

The *spectrum compression ratio* (SCR) represents the ratio of the reduction in size of the global sensor spectrum relative to the summation of the sizes of individual spectra. With the increase in the cluster size, the global spectrum benefits from the shared items among individual spectra. The size of the global spectrum gets reduced relative to the summation of the sizes of individual spectra. The size reduction in the global spectrum affects SCR positively. However, as we keep increasing the cluster size, the number of shared items decreases, the size of the global spectrum increases, and the SCR is affected negatively by the increase in the global spectrum size. The best SCR equals to 0.645 and is achieved for a cluster of size 5 (Figure 10).

Hist-MSE is measured for various cluster sizes in Figure 11. The *SPASS* protocol and its *SPASS+* variation give the same *Hist-MSE* as the *SIMPLE* protocol for a cluster of size 1 (no clustering). *Hist-MSE* decreases till we reach a cluster of size 5, then increases again with the increase in the cluster size. This behavior is accounted for by the behavior of *SCR*. For large cluster sizes, the benefit of sharing the sensors' spectrum is ruined by the overhead of maintaining a large global spectrum. A good tuning of the cluster size is the one that has the best *SCR*.

Cluster size	ρ	SCR
1	1	0
2	0.851	0.285
3	0.68	0.42
4	0.58	0.54
5	0.483	*(0.645)*
6	0.382	0.62
7	0.325	0.546
8	0.275	0.428
9	0.24	0.24
10	0.218	0.078

Figure 10: The effect of cluster size on the internal paremeters of SPASS.

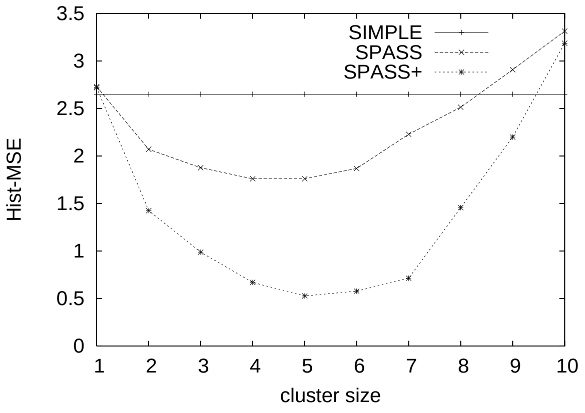

Figure 11: The effect of cluster size on the Hist-MSE.

6. RELATED WORK

Sensors are battery-equipped devices that are capable of sampling, processing, and transmitting readings from the surrounding environment.Various techniques have been proposed to save the sensor battery life at the *sampling*, *processing*, and *transmission* phases. In the remainder of this section, we give a brief overview of these techniques.

Research has been conducted to reduce the sampling rate, i.e., *sampling power*, of the sensors. Statistical models have been utilized recently in [10] to provide estimates of the sensors' readings and to assess the uncertainty of these estimates. A fresh sample is acquired from the sensors that exhibit high levels of uncertainty to refine their estimates. The *SPASS* protocol addresses a similar problem but without the requirement of a statistical model.

A framework to support an acquisitional query language is proposed in [19]. The proposed acquisitional query processor decides which sensors to query and how often to sample from each sensor. The work in [17] suggests the use of quality-aware samplers to regulate the data rate at various levels of the system. Some work, e.g., [3], extends the traditional reservoir sampling [23] to fit in the streaming environment.

In-network processing is carried over at the sensor nodes to reduce the size of the data transmitted to the sensor database [25]. Data aggregation, e.g., max, min, and average, collapses a set of readings to one representative. Notice that our work aims at transmitting the actual sensor readings without performing any data aggregation. The work in [9] proposes approximate in-network aggregation using sketches. Sensors are clustered into groups and one member of the group, the *cluster head*, is responsible for collecting and managing the data of its cluster members. The cluster head is elected based on energy requirements as in [26].

The transmission energy is conserved by techniques that configure the network topology dynamically [6, 24]. In these techniques, sensor nodes exchange messages among each other to acquire knowledge about their locations. Nodes are self-organized based on the acquired location information to reduce the communication cost. The routing decisions among nodes optimize power consumption [7, 15, 16].

7. CONCLUSIONS AND DIRECTIONS FOR FUTURE RESEARCH

In this paper, we addressed the scalability and energy management issues in sensor networks. We introduced the *Sharing and PArtitioning of Stream Spectrum (SPASS)* protocol as a part of the *Sensor Network Support Layer (SNSL)*. The *SNSL* extends the functionalities of data stream management systems to support sensor networks.

We defined the spectrum of a sensor to be the distribution of values that are read by that sensor. Close-by sensors generate similar spectra because they are exposed to similar environmental conditions. In *SPASS*, we proposed to group close-by sensors and to combine their spectra into one global spectrum that is shared among all sensors in the group. The global spectrum is partitioned among sensors such that each sensor transmits the data of its partition with a higher priority. Spectrum partitioning is continuously coordinated to balance the load over the sensor network in the *SPASS+* adaptive version of the protocol. According to the *Histogram Mean Square Error (Hist-MSE)* measure of performance, *SPASS+* achieves up to 70% improvement over the *SIMPLE* protocol for the same level of power consumption.

In future work, we plan to compare the *SPASS* protocol with in-network aggregation and model-based data acquisition techniques. Also, we plan to explore the behavior of the *SPASS+* protocol under various load imbalance conditions. The deployment and testing of the *SPASS* protocol in real-life applications will be our next goal.

8. REFERENCES

[1] M. H. Ali, W. G. Aref, and M. Eltabakh. Scalability via summaries: Stream query processing using promising tuples. Technical Report CSD-05-005, Department of Computer Science, Purdue University, March 2005.

[2] A. Amis, R. Prakash, T. Vuong, and D. Huynh. Max-min d-cluster formation in wireless ad hoc networks. In *Proc. of INFOCOM*, March 2000.

[3] B. Babcoc, M. Datar, and R. Motwani. Sampling from a moving window over streaming data. In *Proc. of the Annual ACM-SIAM Symp. on Discrete Algorithms*, pages 633–634, Jan. 2002.

[4] S. Basagni. Distributed clustering for ad hoc networks. In *Proc. of the Intl. Symposium on Parallel Architectures, Algorithms and Networks (ISPAN)*, 1999.

[5] P. Bonnet, J. E. Gehrke, and P. Seshadri. Towards sensor database systems. In *Proc. of the Intl. Conf. on Mobile Data Management*, pages 3–14, Jan. 2001.

[6] A. Cerpa and D. Estrin. Ascent: Adaptive self-configuring sensor networks topologies. In *Proc. of INFOCOM*, 2002.

[7] J.-H. Chang and L. Tassiulas. Energy conserving routing in wireless ad-hoc networks. In *Proc. of INFOCOM*, pages 22–31, March 2000.

[8] M. Charikar, K. Chen, and M. Farach-Colton. Finding frequent items in data streams. In *Proc. of the Intl. Colloquium on Automata, Languages and Programming*, pages 693–703, July 2002.

[9] J. Considine, F. Li, G. Kollios, and J. W. Byers. Approximate aggregation techniques for sensor databases. In *Proc. of ICDE*, pages 449–460, April 2004.

[10] A. Deshpande, C. Guestrin, S. Madden, J. M. Hellerstein, and W. Hong. Model-driven data acquisition in sensor networks. In *Proc. of VLDB*, pages 588–599, August 2004.

[11] A. Deshpande, S. Nath, P. B. Gibbons, and S. Seshan. Cache-and-query for wide area sensor databases. In *Proc. of ACM SIGMOD*, pages 503–514, June 2003.

[12] D. Estrin, L. Girod, G. Pottie, and M. Srivastava. Instrumenting the world with wireless sensor networks. In *Proc. of Intl. Conf. on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 2675–2678, May 2001.

[13] D. Ganesan, S. Ratnasamy, H. Wang, and D. Estrin. Coping with irregular spatio-temporal sampling in sensor networks. *Computer Communication Review*, 34(1):125–130, 2004.

[14] M. A. Hammad, M. F. Mokbel, M. H. Ali, W. G. Aref, A. C. Catlin, A. K. Elmagarmid, M. Eltabakh, M. G. Elfeky, T. Ghanem, R. Gwadera, I. F. Ilyas, M. Marzouk, and X. Xiong. Nile: A query processing engine for data streams. In *Proc. of ICDE*, page 851, April 2004.

[15] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: a scalable and robust communication paradigm for sensor networks. In *Proc. of MOBICOM*, pages 56–67, 2000.

[16] J. Kulik, W. R. Heinzelman, and H. Balakrishnan. Negotiation-based protocols for disseminating information in wireless sensor networks. *ACM Wireless Networks*, 8(2-3):169–185, 2002.

[17] I. Lazaridis, Q. Han, X. Yu, S. Mehrotra, N. Venkatasubramanian, D. V. Kalashnikov, and W. Yang. Quasar: quality aware sensing architecture. *SIGMOD Record*, 33(1):26–31, 2004.

[18] S. Madden and M. Franklin. Fjording the stream: An architecture for queries over streaming sensor data. In *Proc. of ICDE*, pages 555–566, Feb. 2002.

[19] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. The design of an acquisitional query processor for sensor networks. In *Proc. of ACM SIGMOD*, pages 491–502, 2003.

[20] J.-Y. Pan, S. Seshan, and C. Faloutsos. Fastcars: Fast, correlation-aware sampling for network data mining. In *Proceddings of GLOBECOM*, pages 2167 – 2171, 2002.

[21] S. Srinivasan, H. Latchman, J. Shea, T. Wong, and J. McNair. Airborne traffic surveillance systems: video surveillance of highway traffic. In *the 2nd ACM Intl. workshop on Video surveillance & sensor networks*, pages 131–135, 2004.

[22] R. Szewczyk, E. Osterweil, J. Polastre, M. Hamilton, A. Mainwaring, and D. Estrin. Habitat monitoring with sensor networks. *Comm. of ACM*, 47(6):34–40, 2004.

[23] J. S. Vitter. Random sampling with a reservoir. *ACM Trans. Math. Softw.*, 11(1):37–57, 1985.

[24] Y. Xu, J. S. Heidemann, and D. Estrin. Geography-informed energy conservation for ad hoc routing. In *Proc. of MOBICOM*, pages 70–84, July 2001.

[25] Y. Yao and J. Gehrke. Query processing in sensor networks. In *Proc. of CIDR*, pages 233–244, Jan. 2003.

[26] O. Younis and S. Fahmy. Heed: A hybrid, energy-efficient, distributed clustering approach for ad hoc sensor networks. *IEEE Trans. Mobile Computing*, 3(4):366–379, 2004.