Adaptive-Size Reservoir Sampling over Data Streams

Mohammed Al-Kateb Byung Suk Lee X. Sean Wang

Department of Computer Science
The University of Vermont
Burlington VT 05405, USA
{malkateb,bslee,xywang}@cs.uvm.edu

Abstract

Reservoir sampling is a well-known technique for sequential random sampling over data streams. Conventional reservoir sampling assumes a fixed-size reservoir. There are situations, however, in which it is necessary and/or advantageous to adaptively adjust the size of a reservoir in the middle of sampling due to changes in data characteristics and/or application behavior. This paper studies adaptive-size reservoir sampling over data streams considering two main factors: reservoir size and sample uniformity. First, the paper conducts a theoretical study on the effects of adjusting the size of a reservoir while sampling is in progress. The theoretical results show that such an adjustment may bring a negative impact on the probability of the sample being uniform (called uniformity confidence *herein). Second, the paper presents a novel algorithm for maintaining the reservoir sample after the reservoir size is adjusted such that the resulting uniformity confidence exceeds a given threshold. Third, the paper extends the proposed algorithm to an adaptive multi-reservoir sampling algorithm for a practical application in which samples are collected from memory-limited wireless sensor networks using a mobile sink. Finally, the paper empirically examines the adaptivity of the multi-reservoir sampling algorithm with regard to reservoir size and sample uniformity using real sensor networks data sets.*

1. Introduction

Uniform random sampling has been known for its usefulness and efficiency for generating consistent and unbiased estimates of an underlying population. In this sampling scheme, every possible sample of a given size has the same chance to be selected. Uniform random sampling has been heavily used in a wide range of application domains such as statistical analysis [17] [37], computational geometry [15] [16], graph optimization [31], knowledge discovery [34] [35], approximate query processing [3] [14] [22] [23] [45] [59], and data stream processing [6] [7] [28] [49].

When data subject to sampling come in the form of a data stream (e.g. stock price analysis [2], and sensor networks monitoring [8]), sampling should be processed sequentially. Sequential sampling has two main challenges. First, the input stream must be processed in a single pass. Second, the size of the stream is usually unknown beforehand. A technique commonly used in this scenario is *reservoir sampling* [42] [53], which selects a uniform random sample of a given size from an input stream of an unknown size. Reservoir sampling has been used in many database applications including clustering [25] [32] [33], data warehousing [10], spatial data management [46], and approximate query processing [13] [14] [19] [23] [45] [59].

The conventional reservoir sampling assumes a fixed-size reservoir (i.e., the size of a sample is fixed). There are situations, however, in which it is necessary and/or advantageous to adaptively adjust the reservoir size in the middle of sampling. Key factors characterizing such a situation include changes in sample variance and application behavior. We provide three motivating examples below.

Example 1 *In data collection over wireless sensor networks using a mobile sink [11] [12] [18] [27] [30] [47] [48] [52] [57], a mobile sink roves the area covered by the sensors and collects data from sensors in its proximity. Consider the following scenario (see Figure 1). Sensors are spatially clustered. Each cluster is associated with a sensor proxy which stores readings from sensors in the corresponding cluster and acts as a data cache point. Periodically, a mobile sink navigates the network to collect readings from the proxies. A proxy, however, has limited memory and, therefore, may store only samples of the readings. Each proxy may very well keep multiple reservoir samples, one for each sensor. An application may demand that the size of a reservoir be in proportion to the number of readings generated so far by the corresponding sensor. If the sampling rates of sensors change over time, the reservoir sizes should be adjusted dynamically as the sampling rates of sensors change.*

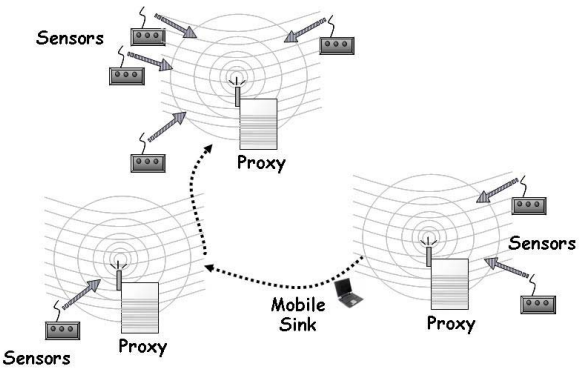

Figure 1. Data collection over a wireless sensor network using a mobile sink.

Example 2 *Reservoir sampling is commonly used in approximate query processing [13] [14] [19] [23] [45] [59] as an efficient sampling technique. For instance, Chaudhuri et al. [14] adapt reservoir sampling to sample the result of a join operation. Random sampling over joins requires information about the statistics of base relations. However, such information may not be available [14] and, therefore, it may not be possible to pre-estimate the size of an intermediate join result and accordingly pre-allocate an appropriate reservoir size. Even if available, such statistics are often inaccurate at run time [9], and the size of an intermediate join result may be much larger than estimated [9]. If, while the sampling is in progress, the reservoir size becomes too small to represent the intermediate join result adequately, then the reservoir size should be increased. Furthermore, if the total available memory for the query processor is limited [9] [44] [60], increasing the size of a reservoir would force the release of some memory elsewhere, possibly decreasing the size of another reservoir.* □

Example 3 *Periodic queries [55], a variation of continuous queries [5], is appropriate for many real-time streaming applications such as security surveillance and health monitoring. In the periodic query model, once a query is registered to the system, query instances are instantiated periodically by the system. Upon instantiation, a query instance takes a snapshot of tuples that arrived since the last instantiation of the query. Consider a common situation in which, due to the nature of data streams and their potentially high arrival rates, a technique like random sampling is used to reduce the stream rate [6]. As the query is instantiated periodically, the system may keep a reservoir sample of stream data arriving between the execution times of two consecutive query instances. If at some time point, the reservoir size has become too small to represent the stream adequately, the system should provide a way to increase the reservoir size for better representing the stream data at the execution time of the next query instance. Moreover, it is not uncommon that multiple queries run simultaneously in the system [39] [49] [51] [54]. In this case, each query may have its own reservoir sample maintained. Besides, at any point in time, one or more queries can be registered to or expired from the system. In order to adapt to this dynamic change of the query set, the system should be able to adaptively reallocate the memory among all reservoirs of the current query set.* □

Adjusting the reservoir size while the sampling is in progress does not come for free. As shown in Section 2, such an adjustment may have a negative impact on the statistical quality of the sample in terms of the probability of the sample being uniform.

Motivated by this observation, in this paper we address adaptive-size reservoir sampling over data streams considering the following two main factors: *reservoir size* (or equivalently *sample size*) and *sample uniformity*. An appropriate sample size depends on data characteristics such as the size, mean, and variance of the population [17] [37]. Sample uniformity brings an unbiased representation of the population, and is especially desirable if it is not clear in advance how the sample will be used [20].

Specifically, we introduce the notion of *uniformity confidence* in a sampling algorithm. Uniformity confidence is the probability that a sampling algorithm generates a uniform random sample. Based on this notion, we conduct a theoretical study on the effect of adjusting the reservoir size in the middle of sampling. The study leads to the following conclusions. On one hand, if the reservoir size decreases, we can maintain the sample in the *reduced* reservoir with a 100% uniformity confidence; on the other hand, if the reservoir size increases, it is not possible to maintain the sample in the *enlarged* reservoir with a 100% uniformity confidence and, in this case, there is a tradeoff between the size of the enlarged reservoir and the uniformity confidence.

Based on the above theoretical results, we propose a novel algorithm (called *adaptive reservoir sampling*) for maintaining the reservoir sample after the reservoir size is adjusted. If the size decreases, the algorithm maintains a sample in the reduced reservoir with a 100% uniformity confidence by randomly evicting tuples from the original reservoir. If the size increases, the algorithm finds the minimum number of incoming tuples that should be considered in the input stream to refill the enlarged reservoir such that the resulting uniformity confidence exceeds a given threshold. Then, the algorithm decides probabilistically on the number of tuples to retain in the enlarged reservoir and randomly evicts the remaining number of tuples. Eventually, the algorithm fills the available room in the enlarged reservoir using the incoming tuples.

To our knowledge, there does not exist an algorithm comparable to the proposed algorithm. Thus, we evaluate the proposed algorithm by extending it for a practical application and empirically examining the adaptivity of the algo-

rithm with regard to reservoir size and sample uniformity. For this, we extend the proposed algorithm to what we call *adaptive multi-reservoir sampling* algorithm for an application in which samples are collected from memory-limited wireless sensor networks using a mobile sink (see Example 1), and conduct experiments using real sensor networks data sets. The experimental results demonstrate the adaptivity of the algorithm through two sets of experiments. The first set of experiments shows the sizes of multiple reservoirs changing adaptively to the change in the sampling rate of sensors and the second set of experiments shows the effects of these changes on the samples' uniformity.

This paper makes the following contributions:

1. Identifies and motivates the problem of adaptive-size reservoir sampling in which the reservoir size is adjusted while the sampling is in progress.
2. Introduces the notion of uniformity confidence and, based on this notion, examines theoretical results on adjusting the reservoir size in the middle of sampling.
3. Proposes the adaptive reservoir sampling algorithm for maintaining the reservoir sample after the reservoir size is adjusted.
4. Extends the proposed algorithm to the adaptive multi-reservoir sampling algorithm, and, using real sensor networks data sets, empirically demonstrates the adaptivity of the algorithm with regard to reservoir size and sample uniformity.

The rest of the paper is organized as follows. Section 2 introduces the notion of uniformity confidence and presents the adaptive reservoir sampling algorithm. Section 3 presents the adaptive multi-reservoir sampling algorithm, and Section 4 empirically demonstrates the adaptivity of the algorithm. Section 5 briefly reviews related work. Section 6 concludes this paper and suggests future work.

2. Adaptive Reservoir Sampling

In this section, we briefly review the reservoir sampling algorithm. Then, we introduce the notion of uniformity confidence (denoted as UC) and conduct a theoretical study on the effects of decreasing or increasing a reservoir size in the middle of sampling on the uniformity confidence. Finally, we propose our adaptive reservoir sampling algorithm.

2.1. Reservoir sampling

The conventional reservoir sampling [42] selects a uniform random sample of a fixed size, without replacement, from an input stream of an unknown size (see Algorithm 1). Initially, the algorithm places all tuples in the reservoir until the reservoir (of the size of r tuples) becomes full. After that, each k^{th} tuple is sampled with the probability $\frac{r}{k}$. A sampled tuple replaces a randomly selected tuple in the reservoir. It is easy to verify that the reservoir always holds a *uniform* random sample of the k tuples seen so far [42].

Algorithm 1 *conventional reservoir sampling*

Inputs: r {*reservoir size*}

1: $k = 0$
2: **for** each tuple arriving from the input stream **do**
3: $k = k + 1$
4: **if** $k \leq r$ **then**
5: add the tuple to the reservoir
6: **else**
7: sample the tuple with the probability $\frac{r}{k}$ and replace a randomly selected tuple in the reservoir with the sampled tuple
8: **end if**
9: **end for**

2.2. Uniformity confidence

A sample is a uniform random sample if it is produced using a sampling scheme in which all statistically possible samples of the same size are equally likely to be selected. In this case, we say the uniformity confidence in the sampling algorithm equals 100%. In contrast, if some statistically possible samples cannot be selected using a certain sampling algorithm, then we say the uniformity confidence in the sampling algorithm is below 100%. Thus, we define uniformity confidence as follows.

$$\frac{\text{the number of different samples of the same size possible with the algorithm}}{\text{the number of different samples of the same size possible statistically}} \times 100 \quad (1)$$

For reservoir sampling, the uniformity confidence in a reservoir sampling algorithm which produces a sample $\mathbb{S}$ of size r (denoted as $\mathbb{S}_{[r]}$) is defined as the probability that $\mathbb{S}_{[r]}$ is a uniform random sample of *all the tuples seen so far*. That is, if k tuples have been seen so far, then the uniformity confidence is 100% if and only if every statistically possible $\mathbb{S}_{[r]}$ has an equal probability to be selected from the k tuples. As we show in Sections 2.2.1 and 2.2.2 below, if the reservoir size is decreased from r to $r - \delta$ ($\delta > 0$), then there is a way to maintain the sample in the reduced reservoir such that every statistically possible $\mathbb{S}_{[r-\delta]}$ has an equal probability of being in the reduced reservoir, whereas if the reservoir size is increased from r to $r+\delta$ ($\delta > 0$), then some statistically possible $\mathbb{S}_{[r+\delta]}$'s cannot be selected.

2.2.1 Decreasing the reservoir size

Suppose the size of a reservoir is decreased from r to $r - \delta$ ($\delta > 0$) immediate after the k^{th} tuple arrives (see Figure 2). Then, the sample in the reduced reservoir can be maintained by randomly evicting δ tuples from the original reservoir.

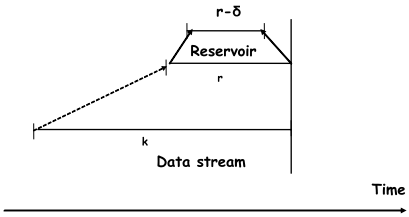

Figure 2. Decreasing the reservoir size.

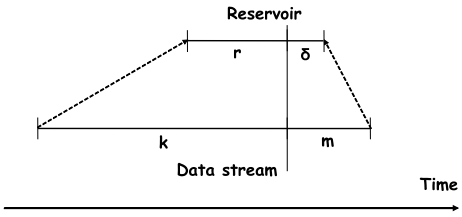

Figure 3. Increasing the reservoir size.

With this random eviction in place, there are $\binom{r}{r-\delta}$ different $\mathbb{S}_{[r-\delta]}$'s that can be selected in the reduced reservoir from the original reservoir. Note that there are $\binom{k}{r}$ different $\mathbb{S}_{[r]}$'s that can be selected in the original reservoir from the k tuples and there are $\binom{k-(r-\delta)}{r-(r-\delta)}$ duplicate $\mathbb{S}_{[r-\delta]}$'s that can be selected in the reduced reservoir from the different $\mathbb{S}_{[r]}$'s. Therefore, there are $(\binom{k}{r}\binom{r}{r-\delta})/\binom{k-(r-\delta)}{r-(r-\delta)}$ different $\mathbb{S}_{[r-\delta]}$'s that can be selected in the reduced reservoir from the k tuples. On the other hand, the number of different samples of size $r-\delta$ that should be statistically possible from sampling k tuples is $\binom{k}{r-\delta}$. Hence, the uniformity confidence is expressed as follows:

$$UC(k,r,\delta) = \frac{(\binom{k}{r}\binom{r}{r-\delta})/\binom{k-(r-\delta)}{r-(r-\delta)}}{\binom{k}{r-\delta}} \times 100 = \frac{\binom{k}{r-\delta}}{\binom{k}{r-\delta}} \times 100 \tag{2}$$

which clearly shows the the uniformity confidence is 100%.

The following theorem summarizes the uniformity confidence property of reservoir sampling in the event of decreasing the reservoir size during sampling.

Theorem 1 *If the size of a reservoir is decreased from r to $r-\delta$ ($\delta > 0$) while sampling from an input stream is in progress, it is possible to maintain the sample in the reduced reservoir with a 100% uniformity confidence.*

Proof To prove that the uniformity confidence is 100%, we only need to show that every tuple among the k tuples seen so far has an equal probability to be selected in the reduced reservoir. Following the conventional reservoir sampling, each of the k tuples has the equal probability $\frac{r}{k}$ to be selected in the original reservoir. Once the reservoir size decreases to r-δ, we can sample the tuples in the original reservoir with the equal probability $\frac{r-\delta}{r}$ to select r-δ tuples for the reduced reservoir. Thus, every tuple among the k tuples has the equal probability $\frac{r}{k} \times \frac{r-\delta}{r} = \frac{r-\delta}{k}$ to be selected in the reduced reservoir.

2.2.2 Increasing the reservoir size

Suppose the size of a reservoir is increased from r to r+δ ($\delta > 0$) immediately after the k^{th} tuple arrives (see Figure 3). Then, the reservoir has room for δ additional tuples. Clearly, there is no way to fill this room from sampling the k tuples as they have already passed by. We can only use incoming tuples to fill the room. The number of incoming tuples used to fill the enlarged reservoir is denoted as m and called the *uniformity confidence recovery tuple count*.

For the sake of better uniformity, we allow some of the r existing tuples to be evicted probabilistically and replaced by some of the incoming m tuples. In our work, we *randomly* pick the number of tuples evicted (or equivalently, the number of tuples retained). Clearly, the number of tuples that are retained, x, can be no more than r. Besides, x should not be less than $(r+\delta)-m$ if $m < r+\delta$ (because otherwise, with all m incoming tuples the enlarged reservoir cannot be refilled), and no less than 0 otherwise. Hence, we can have x tuples, where $x \in [\max\{0,(r+\delta)-m\}, r]$, from the k tuples and the other $r+\delta-x$ tuples from the m tuples. This eviction scheme allows for $\binom{k}{x}\binom{m}{r+\delta-x}$ different $\mathbb{S}_{[r+\delta]}$'s for each x in the range $[\max\{0,(r+\delta)-m\}, r]$. On the other hand, the number of different samples of size $r+\delta$ that should be statistically possible from sampling $k+m$ tuples is $\binom{k+m}{r+\delta}$. Hence, with the eviction in place, the uniformity confidence is expressed as follows:

$$UC(k,r,\delta,m) = \frac{\sum_{x=\max\{0,(r+\delta)-m\}}^{r}\binom{k}{x}\binom{m}{r+\delta-x}}{\binom{k+m}{r+\delta}} \times 100 \tag{3}$$

where $m \geq \delta$.

Examining this formula shows that the uniformity confidence increases monotonously and saturates as m increases. Figure 4 shows this pattern for one setting of k, r, and δ. Note that the uniformity confidence *never* reaches 100%, as exemplified by Figure 5 which magnifies the uniformity confidence curve of Figure 4 for $m \geq 9000$.

The following theorem summarizes the uniformity confidence property of reservoir sampling in the event of increasing the reservoir size during sampling.

Theorem 2 *If the size of a reservoir is increased from r to $r+\delta$ ($\delta > 0$) while sampling from an input stream is in progress (after seeing more than r tuples), it is not possible to maintain the sample in the enlarged reservoir with a 100% uniformity confidence.*

Proof Let x be the number of tuples that can be selected in the enlarged reservoir (of size $r+\delta$) from the k tuples seen so far in the input stream. Then, the uniformity confidence is equal to 100% if and only if x can be any value in the range of [0,$r+\delta$]. However, x cannot be more than r since we have only r tuples from the k tuples seen so far. From this we conclude that the uniformity confidence cannot reach 100%.

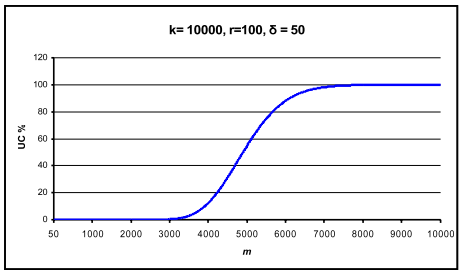

Figure 4. UC with respect to m (Equation 3).

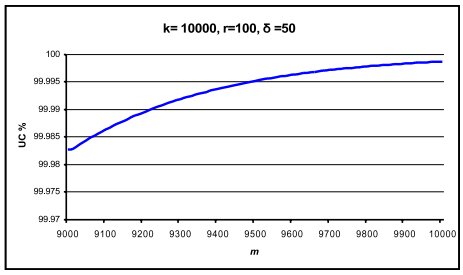

Figure 5. Figure 4 magnified for $m \geq 9000$.

2.3. Adaptive reservoir sampling algorithm

Algorithm 2 *Adaptive reservoir sampling*

Inputs: r {*reservoir size*}
k {*number of tuples seen so far*}
ζ {*uniformity confidence threshold*}

1: **while** true **do**
2: **while** reservoir size does not change **do**
3: conventional reservoir sampling (Algorithm 1).
4: **end while**
5: **if** reservoir size is decreased by δ **then**
6: randomly evicts δ tuples from the reservoir.
7: **else**
8: {*i.e., reservoir size is increased by* δ}
9: Find the minimum value of m (using Equation 3 with the current values of k, r, δ) that causes the UC to exceed ζ.
10: flip a biased coin to decide on the number, x, of tuples to retain among r tuples already in the reservoir (Equation 4).
11: randomly evict $r - x$ tuples from the reservoir.
12: select $r+\delta-x$ tuples from the incoming m tuples using conventional reservoir sampling (Algorithm 1).
13: **end if**
14: **end while**

Based on the above discussion, our adaptive reservoir sampling algorithm works as shown in Algorithm 2. As long as the size of the reservoir does not change, it uses the conventional reservoir sampling to sample the input stream (Line 3). If the reservoir size decreases by δ, the algorithm evicts δ tuples from the reservoir randomly (Line 6). After that, the algorithm continues sampling using the conventional reservoir sampling (Line 3). On the other hand, if the reservoir size increases by δ, the algorithm computes the minimum value of m (using Equation 3) that causes the UC to exceed a given threshold (ζ) (Line 9). Then, the algorithm flips a biased coin to decide on the number of tuples (x) to retain among the r tuples already in the reservoir (Line 10). The probability of choosing the value x, where $\max\{0, (r+\delta)-m\} \leq x \leq r$, is defined as:

$$p(x) = \frac{\binom{k}{x}\binom{m}{r+\delta-x}}{\binom{k+m}{r+\delta}} \quad (4)$$

After that, the algorithm randomly evicts $r - x$ tuples from the reservoir (Line 11) and refills the remaining reservoir space with $r + \delta - x$ tuples from the arriving m tuples using the conventional reservoir sampling (Line 12). Eventually, the algorithm continues sampling the input stream using the conventional reservoir sampling (Line 3) as if the sample in the enlarged reservoir were a uninform random sample of the $k + m$ tuples.

3. Application to Adaptive Multi-reservoir Sampling

In this section, we extend the adaptive reservoir sampling algorithm for a practical application of multi-reservoir sampling in which samples are collected from memory-limited wireless sensor networks using a mobile sink. Applications of data collection over wireless sensor networks using a mobile sink have recently received a significant research attention [11] [12] [18] [27] [30] [47] [48] [52] [57]. These applications take advantage of the mobility to improve the process of data gathering. A mobile sink roves the network and collects data from sensors in its proximity, thereby reducing the in-network communications and increasing the lifetime of the network.

We consider the application scenario described in Example 1, assuming that the processing power of each proxy is sufficient to carry the required computations. For this scenario, we propose an adaptive multi-reservoir sampling algorithm that is based one the following key ideas. First, the objective of the algorithm is to *adaptively* adjust the memory allocation in each proxy so that the size of each reservoir is allocated in proportion to the number of readings (i.e., tuples) generated so far by the corresponding sensor. Second, the algorithm adjusts the memory allocation only if the relative change in the size of at least one reservoir is above a given memory adjustment threshold and the resulting uniformity confidence for all reservoirs exceeds a given uniformity confidence threshold.

3.1. Problem formulation

The objective is to allocate the memory of size M to the reservoirs (R_1,R_2,...,R_v) of v input streams (S_1,S_2,...,S_v) so that at the current time point t, the size $r_i(t)$ of each reservoir R_i is proportional to the total number of tuples, $k_i(t)$, seen so far from S_i. The rationale behind this objective is explained below. (Table 1 summarizes the notations used in the problem formulation.)

Table 1. Notations used in the problem formulation.

Symbol	Description
v	number of streams (i.e., number of reservoirs)
S_i	stream i
R_i	the reservoir allocated to S_i
M	total available memory for v reservoirs
t	current time point
$r_i(t)$	computed size of R_i at t
$r_i^M(t)$	computed size of R_i at t with limited memory M
$r_i(t_u)$	size of R_i adjusted at time point t_u ($t_u < t$)
$\delta_i(t)$	change in the size of R_i at t
$k_i(t)$	number of tuples seen up to t from S_i
$m_i(t)$	number of tuples to be seen from S_i, starting from t, to fill an enlarged reservoir R_i
$\lambda_i(t)$	the average stream rate of S_i
$\top$	time period left until the next data collection time
ζ	uniformity confidence threshold
φ	memory adjustment threshold ($0 \leq \varphi \leq 1$)

Three criteria are typically used to determine a statistically appropriate sample size for a given population. These criteria are the *confidence interval*, the *confidence level*, and the *degree of variability* in the population [43]. Confidence interval is the range in which the true value of the population is estimated to be. Confidence level is the probability value associated with a confidence interval. Degree of variability in the population is the degree in which the attributes being measured are distributed throughout the population. A more heterogeneous population requires larger sample to achieve a given confidence interval. Based on these criteria, Yamane [58] provides the following simplified formula to calculate a statistically appropriate sample size, assuming 95% confidence level and 50% degree of variability (note that 50% indicates the maximum variability in a population):

$$n = \frac{N}{1 + N\, e^2} \tag{5}$$

where n is the sample size, N is the population size, and e is $1-$confidence interval.

Adapting this formula to our problem, we compute the size $r_i(t)$ of R_i as:

$$r_i(t) = \frac{k_i(t)}{1 + k_i(t)\, e^2} \tag{6}$$

subject to the following limit on the total memory M:

$$\sum_{i=1}^{v} r_i(t) \leq M \tag{7}$$

We assume that M may not be large enough for all reservoirs. In this case, we use the heuristic of allocating the memory to each reservoir R_i in proportion to the value of $r_i(t)$ computed using Equation 6. That is:

$$r_i^M(t) = \lfloor M(\frac{r_i(t)}{\sum_{i=1}^{v} r_i(t)}) \rfloor \tag{8}$$

Algorithm 3 *Adaptive multi-reservoir sampling*

Inputs: ζ, φ, M, $\top$, $\{r_1(t_u), r_2(t_u), ..., r_v(t_u)\}$, $\{k_1(t), k_2(t), ..., k_v(t)\}$, $\{\lambda_1(t), \lambda_2(t), ..., \lambda_v(t)\}$

1: **while** true **do**
2: **while** there is no tuples arriving from any stream **do**
3: {do nothing.}
4: **end while**
{one or more tuples arrived from some streams}
5: compute $r_i(t)$ (Equation 6) for the streams from which tuples arrived.
6: **for** each $R_i \in \{R_1, R_2, ..., R_v\}$ **do**
7: compute $r_i^M(t)$ (Equation 8).
8: compute $\delta_i(t) = r_i^M(t) - r_i(t_u)$.
9: **end for**
10: **if** Equation 11 holds for any $R_i \in \{R_1, R_2, ..., R_v\}$ **then**
11: $L_{reduced}$ = set of all R_i whose $\delta_i(t) < 0$
12: $L_{enlarged}$ = set of all R_i whose $\delta_i(t) > 0$
13: compute $m_i(t)$ (Equation 9) for all $R_i \in L_{enlarged}$.
14: $L'_{enlarged}$ = set of all $R_i \in L_{enlarged}$ whose $UC_i(k_i(t), r_i(t_u), \delta_i(t), m_i(t)) \leq \zeta$
15: **if** $L'_{enlarged}$ is empty **then**
16: **for** each $R_i \in (L_{reduced} \cup L_{enlarged})$ **do**
17: **if** $R_i \in L_{reduced}$ **then**
18: randomly evicts $\delta_i(t)$ tuples from R_i.
19: **else**
20: flip a biased coin to decide on the number of tuples, x, to retain in R_i (using Equation 4 with $k_i(t)$,$r_i(t_u)$,$\delta_i(t)$,$m_i(t)$ substituting k,r,δ,m, respectively).
21: randomly evict $r_i(t_u) - x$ tuples from R_i.
22: select $r_i(t_u) + \delta_i(t) - x$ tuples from the incoming $m_i(t)$ tuples using Algorithm 1.
23: **end if**
24: $r_i(t_u) = r_i^M(t)$
25: **end for**
26: **end if**
27: **end if**
28: **end while**

At the current time point t, this computed reservoir size $r_i^M(t)$ may be different from the reservoir size $r_i(t_u)$ adjusted at time point t_u ($t_u < t$). Let $\delta_i(t)$ denotes the difference. As concluded in Section 2, if $\delta_i(t) < 0$, the uniformity confidence is 100%. In contrast, if $\delta_i(t) > 0$, the uniformity confidence is below 100%; in this case, as in Algorithm 2, we maintain the sample in an enlarged reservoir R_i using incoming tuples from the input stream. In our problem formulation, the number of incoming tuples $m_i(t)$ used to fill an enlarged reservoir R_i is computed as a product of the average stream rate, $\lambda_i(t)$, and the time period, $\top$, left to the next data collection time as follows:

$$m_i(t) = \lambda_i(t) \times \top \tag{9}$$

For an enlarged reservoir R_i, the uniformity confidence expressed in Equation 3 is refined here as follows:

$$UC_i(k_i(t), r_i(t_u), \delta_i(t), m_i(t)) =$$

$$\frac{\sum_{x=\max\{0,(r_i(t_u)+\delta_i(t))-m_i(t)\}}^{r_i(t_u)} \binom{k_i(t)}{x}\binom{m_i(t)}{r_i(t_u)+\delta_i(t)-x}}{\binom{k_i(t)+m_i(t)}{r_i(t_u)+\delta_i(t)}} \times 100 \quad (10)$$

where $m_i(t) \geq \delta_i(t)$.

To control the frequency of memory allocation adjustment, we consider the adjustment only if the relative change in the computed size (Equation 8) exceeds a given threshold (denoted as φ) for some R_i, that is, the adjustment is considered if Equation 11 holds for some $i \in \{1, 2, ...v\}$.

$$\frac{\left|r_i^M(t) - r_i(t_u)\right|}{r_i(t_u)} > \varphi \quad (11)$$

where $0 \leq \varphi \leq 1$.

3.2. Adaptive multi-reservoir sampling algorithm

Based on the problem formulation in Section 3.1, the algorithm works as follows (see Algorithm 3). As long as there is no tuples arriving from any stream, the algorithm stays idle (Lines 2-4). Upon the arrival of a new tuple from any stream, it computes $r_i(t)$ for those streams from which tuples arrived (Line 5) and computes $r_i^M(t)$ and $\delta_i(t)$ for all streams (Lines 6-9). Then, it checks the relative change in the size of each reservoir (Line 10). If the relative change in the size of *any* reservoirs is larger than the memory adjustment threshold φ (using Equation 11), the algorithm computes $m_i(t)$ for all of the enlarged reservoirs (Lines 11 and 13). Then, it checks if the uniformity confidence computed using Equation 10 exceeds the given threshold for every enlarged reservoir (Lines 14-15). If so, for each of the adjusted reservoirs, it applies the corresponding steps of the adaptive reservoir sampling algorithm (see Algorithm 2), and updates $r_i(t_u)$ to the current $r_i^M(t)$ (Lines 16-25).

4. Performance Evaluations

The purpose of the evaluations is to empirically examine the adaptivity of the the multi-reservoir sampling algorithm with regard to reservoir size and sample uniformity. We have conducted two sets of experiments. The objective of the first set of experiments is to observe how the reservoirs sizes change as data arrive. The objective of the second set of experiments is to observe the uniformity of the reservoir samples as the reservoirs sizes change.

4.1. Setup

4.1.1 Data setup

We use a real data set collected from sensors deployed in the Intel Berkeley Research lab between February 28^{th} and April 5^{th}, 2004 [1]. Sensors mounted with weather boards collected timestamped topology information, along with humidity, temperature, light and voltage values once every 31 seconds. Collection of data was done using the TinyDB in-network query processing system, built on the TinyOS platform.

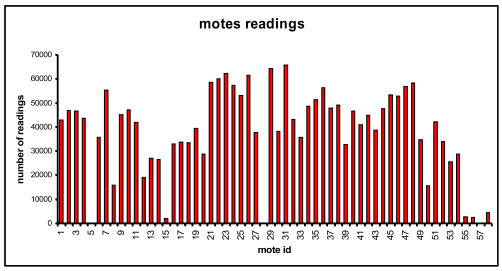

Figure 6. Total number of readings from each mote between February 28^{th} and April 5^{th}.

The resulting data file includes a log of about 2.3 million readings collected from these sensors. The schema of records is ⟨date: yyyy-mm-dd, time: hh:mm:ss.xxx, epoch: int, moteid: int, temperature: real, humidity: real, light:real, voltage:real⟩. In this schema, epoch is a monotonically increasing sequence number unique for each mote. Moteid range from 1 to 58. Data from three motes (of ID=5, ID=28, and ID=57) have incomplete readings, and thus discarded. This leaves readings from 55 motes used in the experiments. (Figure 6 reports the total number of readings from each mote.) Temperature is in degrees Celsius. Humidity is temperature-corrected relative humidity, ranging from 0 to 100%. Light is in Lux.Voltage is expressed in volts, ranging from 2.0 to 3.0 volts.

4.1.2 Algorithm setup

In Algorithm 3, we set the uniformity confidence threshold ζ to 0.90. We believe this value is adequately large to constrain the frequency of adjusting the memory allocation. To check the effect of the total available memory size on the frequency of change in reservoir sizes, we range the value of M from 1000 (tuples) to 5000 (tuples) and range the memory adjustment threshold φ from 0.1 to 0.5. Readings acquired for the whole day of February 28^{th} are used in the experiments. We assume data collection is done every 1 hour and, accordingly, report results on the change in reservoir size and sample uniformity every hour.

4.2. Change in reservoir size

Figure 7 shows the changes in the sizes of the 55 reservoirs. For better visibility, Figure 8 shows the changes for 5 selected reservoirs. In the beginning (i.e., by the end of the 2^{nd} hour), the total available memory is enough to store all reading from all motes and, therefore, the reservoir sizes increase linearly. Then, the reservoir sizes start fluctuating. The fluctuations are smooth and small in the first stage

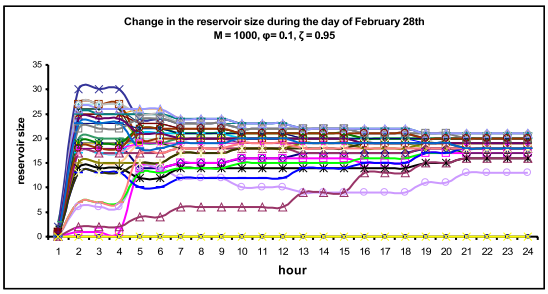

Figure 7. Change in sizes of reservoirs.

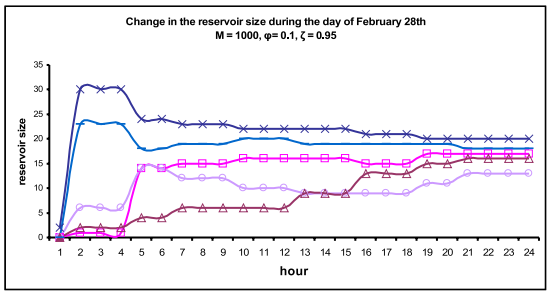

Figure 8. Changes in sizes for selected reservoirs (motes IDs: 2, 15, 31, 49, and 54).

(from the 2^{nd} to the 4^{th} hour), larger in the second stage (from the 4^{th} to the 21^{st} hour), and eventually diminish in the last stage (after the 21^{st} hour). This pattern of changes is attributed to the characteristics of data sets used in the experiments. In the first stage, there is no tangible difference between the numbers of readings acquired by different motes. Therefore, reservoir sizes stay almost constant. In the second stage, the differences start increasing and, therefore, the changes in reservoir sizes become more frequent and more tangible. The saturations in reservoir sizes in the last stage indicate that the number of readings acquired by each mote is large enough and, therefore, does not cause a change in the computed reservoir size (see Equation 6).

With a larger value of the memory adjustment threshold φ ($= 0.5$), Figure 9 shows a similar pattern except that the changes in reservoir sizes happen less frequently, and saturate earlier. The reason for these observations can be easily seen from Equations 6 and 11. Results obtained for varying other parameters (M and ζ) show similar patterns, and are omitted due to space constraint.

4.3. Sample uniformity

We use χ^2 statistics [24] [37] as a metric of the sample uniformity. Higher χ^2 indicates lower uniformity and vice versa. For each value v in a domain D, χ^2 statistics measures the relative difference between the observed number of tuples ($o(v)$) and the expected number of tuples ($e(v)$) that contain the value v. That is:

$$\chi^2 = \sum_{\forall v \in D} \frac{(e(v) - o(v))^2}{e(v)} \tag{12}$$

In our experiments, we measure χ^2 statistics for the humidity attribute. For this, we round the original real value of humidity to return the closest int to that original value.

Figure 10 shows the changes in size and the resulting sample uniformity for one selected reservoir. It shows that when the reservoir size increases, the sample uniformity degrades (i.e., decreases) and then starts recovering (i.e., increasing). The degree of uniformity degradation and recovery varies due to randomness in the data sets used in experiments.

5. Related Work

This work builds on two main research: *adaptive sampling* and *sampling over data streams*. In this section, we briefly review related work to each of these research.

Adaptive sampling: In *statistics* literature [50], adaptive sampling refers to a design technique for statistical experiments, where data acquired from experiments are used to adjust the experiment as it is being run. In the *database* community, adaptive sampling has been mainly used in query size estimation. Lipton et al. give an adaptive sampling algorithm for query size estimation, where the number of samples taken depends on the information obtained from each sample [36]. Lynch in [38] introduce detailed theoretical results on using adaptive sampling to compute an approximate size of the query. The study conducts an analysis of the average run time of different adaptive sampling algorithms. In [59], Yi-Leh Wu et al. propose a query estimation technique, based on random sampling, that can adapt to user query patterns to estimate the size of selection or range queries over databases. Recently Gemulla et al. [20] proposed an algorithm for resizing a bounded-size sample assuming that tuples are archived in the system for possible further accesses. In *sensor networks* research, adaptive sampling techniques provide adaptive control mechanisms to change the sampling rate at each sensor in an adaption to changes in the streaming-data characteristics [26] [41] [56].

Sampling over Data Streams: In addition to reservoir sampling [42] [53], there are several special-purpose algorithms for sampling over data streams, such as *heavy hitters* [40] and *distinct counts* [21]. Heavy hitters find those elements in a data stream that appear for at least a certain fraction of all tuples, and distinct counts estimate the number of distinct values for a given target attribute over an input stream. In [7], Babcock et al. present memory-efficient algorithms for the problem of maintaining a uniform random sample of a specified size from a moving window over a data stream. In [29], Johnson et al. abstract the process of sampling from a stream and design a generic sampling operator which can be specialized to implement a wide variety of stream sampling algorithms. With the objective of

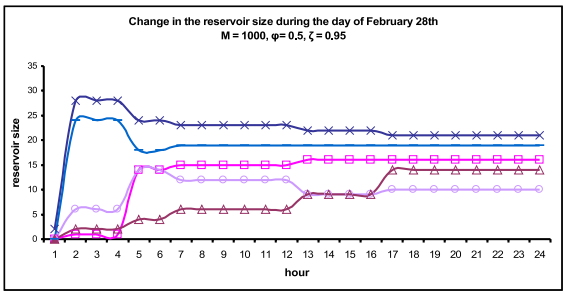

Figure 9. Changes in sizes for selected reservoirs (motes IDs: 2, 15, 31, 49, and 54).

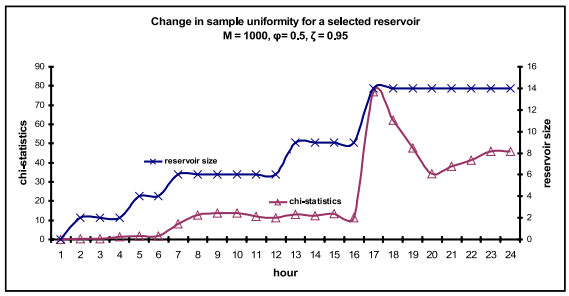

Figure 10. Changes in the sample uniformity for a selected reservoir (mote ID: 49).

minimizing answers inaccuracy in sliding-window aggregation queries where processing power is limited, Babcock et al. [6] propose a load shedding technique which includes random sampling operators in a query plan. Srivastava et al. introduce a random sampling algorithm to stream out a uniform random sample over stream join with limited memory [49]. In [4], we present a progressive reservoir join-sampling algorithm for sampling over memory-limited stream joins. The algorithm increases the reservoir size during the join-sampling process by releasing memory from the join buffer and allocating it to the reservoir.

6. Conclusion and Future Work

In this paper, we addressed the problem of adaptive-size reservoir sampling over data streams. We introduced the notion of uniformity confidence and conducted a theoretical study on the effects of adjusting the reservoir size in the middle of sampling. The study concludes that if the reservoir size decreases, we can maintain the sample in the *reduced* reservoir with a 100% confidence uniformity. In contrast, if the reservoir size increases, it is not possible to maintain the sample in the *enlarged* reservoir with a 100% uniformity confidence. We proposed an algorithm for maintaining the reservoir sample after the reservoir size is adjusted such that the resulting uniformity confidence exceeds a given threshold. We extended the proposed algorithm to an adaptive multi-reservoir sampling algorithm, and empirically examined the adaptivity of the multi-reservoir sampling algorithm with regard to reservoir size and sample uniformity using real sensor networks data sets.

Several issues are still open for future work. Alternative algorithms for adaptive sampling may be developed and compared, and more real-world applications may be investigated for practical usefulness of the algorithms.

Acknowledgments

This research is based upon work supported by the National Science Foundation under Grant No. IIS-0415023. We would also like to thank Peter Bodik (UC Berkeley), Wei Hong (Arched Rock Corporation), Carlos Guestrin (Carnegie Mellon University), Sam Madden (MIT), Mark Paskin (Stanford University), and Romain Thibaux (UC Berkeley) for graciously granting the permission to use their sensor data sets in our experiments.

References

[1] Intel lab data. http://berkeley.intel-research.net/labdata/.

[2] Traderbot home page. http://www.traderbot.com.

[3] S. Acharya, P. B. Gibbons, V. Poosala, and S. Ramaswamy. Join synopses for approximate query answering. In *SIGMOD '99*, pages 275–286.

[4] M. Al-Kateb, B. S. Lee, and X. S. Wang. Reservoir sampling over memory-limited stream joins. In *SSDBM '07*.

[5] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *PODS '02*, pages 1–16.

[6] B. Babcock, M. Datar, and R. Motwani. Load shedding for aggregation queries over data streams. In *ICDE '04*, pages 350–361.

[7] B. Babcock, M. Datar, and R. Motwani. Sampling from a moving window over streaming data. In *SODA '02*, pages 633–634.

[8] P. Bonnet, J. Gehrke, and P. Seshadri. Towards sensor database systems. In *MDM '01*, pages 3–14.

[9] L. Bouganim, O. Kapitskaia, and P. Valduriez. Memory-adaptive scheduling for large query execution. In *CIKM '98*, pages 105–115.

[10] P. G. Brown and P. J. Haas. Techniques for warehousing of sample data. In *ICDE '06*, pages 6–17.

[11] M. M. Cecilia Mascolo and B. Pasztor. Data collection in delay tolerant mobile sensor networks using SCAR. In *SenSys '06*.

[12] I. Chatzigiannakis, A. Kinalis, and S. Nikoletseas. Sink mobility protocols for data collection in wireless sensor networks. In *MobiWac '06*, pages 52–59.

[13] S. Chaudhuri, G. Das, and V. Narasayya. A robust, optimization-based approach for approximate answering of aggregate queries. In *SIGMOD '01*, pages 295–306.

[14] S. Chaudhuri, R. Motwani, and V. Narasayya. On random sampling over joins. In *SIGMOD '99*, pages 263–274.

[15] K. Clarkson. A probabilistic algorithm for the post office problem. In *STOC '85*, pages 175–184.

[16] K. L. Clarkson. Applications of random sampling in computational geometry, ii. In *SCG '88*, pages 1–11.

[17] W. G. Cochran. *Sampling Techniques*. John Wiley, third edition edition, 1977.

[18] A. S. D. Jea and M. Srivastava. Multiple controlled mobile elements (data mules) for data collection in sensor networks. In *DCOSS '05*, pages pages 244–257.

[19] V. Ganti, M.-L. Lee, and R. Ramakrishnan. ICICLES: Self-tuning samples for approximate query answering. In *VLDB '02*, pages 176–187.

[20] R. Gemulla, W. Lehner, and P. J. Haas. A dip in the reservoir: Maintaining sample synopses of evolving datasets. In *VLDB '06*, pages 595–606.

[21] P. B. Gibbons. Distinct sampling for highly-accurate answers to distinct values queries and event reports. In *VLDB '01*, pages 541–550.

[22] P. B. Gibbons and Y. Matias. New sampling-based summary statistics for improving approximate query answers. In *SIGMOD '98*, pages 331–342.

[23] P. B. Gibbons, Y. Matias, and V. Poosala. Fast incremental maintenance of approximate histograms. In *VLDB '97*, pages 466–475.

[24] N. M. Greenwood, P.E. *A Guide to Chi-Squared Testing*. John Wiley and sons, New York., 1996.

[25] S. Guha, R. Rastogi, and K. Shim. CURE: an efficient clustering algorithm for large databases. In *SIGMOD '98*, pages 73–84.

[26] A. Jain and E. Y. Chang. Adaptive sampling for sensor networks. In *DMSN '04*, pages 10–16.

[27] S. Jain, R. C. Shah, W. Brunette, G. Borriello, and S. Roy. Exploiting mobility for energy efficient data collection in wireless sensor networks. *Mob. Netw. Appl.*, 11(3):327–339, 2006.

[28] C. Jermaine, A. Pol, and S. Arumugam. Online maintenance of very large random samples. In *SIGMOD '04*, pages 299–310.

[29] T. Johnson, S. Muthukrishnan, and I. Rozenbaum. Sampling algorithms in a stream operator. In *SIGMOD '05*, pages 1–12.

[30] 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-X '02*, pages 96–107.

[31] D. R. Karger. *Random sampling in graph optimization problems*. PhD thesis, Stanford, CA, USA, 1995.

[32] K. Kerdprasop, N. Kerdprasop, and P. Sattayatham. Density-biased clustering based on reservoir sampling. In *DEXA Workshops '05*, pages 1122–1126.

[33] K. Kerdprasop, N. Kerdprasop, and J. Sun. Density biased reservoir sampling for clustering. In *AIA '05*, pages 95–100.

[34] J. Kivinen and H. Mannila. The power of sampling in knowledge discovery. In *PODS '94*, pages 77–85.

[35] S. D. Lee, D. W. Cheung, and B. Kao. Is sampling useful in data mining? a case in the maintenance of discovered association rules. *Data Min. Knowl. Discov.*, 2(3):233–262, 1998.

[36] R. J. Lipton and J. F. Naughton. Query size estimation by adaptive sampling. In *PODS'95*, pages 18–25.

[37] S. L. Lohr, editor. *Sampling: Design and Analysis*. Duxbury Press, Pacific Grove, 1999.

[38] J. F. Lynch. Analysis and application of adaptive sampling. In *PODS '00*, pages 260–267.

[39] S. Madden, M. Shah, J. M. Hellerstein, and V. Raman. Continuously adaptive continuous queries over streams. In *SIGMOD '02*, pages 49–60.

[40] G. S. Manku and R. Motwani. Approximate frequency counts over data streams. In *VLDB '02*, pages 346–357.

[41] A. D. Marbini and L. E. Sacks. Adaptive sampling mechanisms in sensor networks. In *LCS '03*.

[42] A. McLeod and D. Bellhouse. A convenient algorithm for drawing a simple random sample. *Applied Statistics*, 32:182184, 1983.

[43] R. D. M. Miaoulis, George. *An Introduction to Sampling*. Dubuque, Iowa: Kendall/Hunt Publishing Company, 1976.

[44] B. Nag and D. J. DeWitt. Memory allocation strategies for complex decision support queries. In *CIKM '98*, pages 116–123.

[45] F. Olken. *Random Sampling from Databases*. PhD thesis, Mailstop 50B-3238, 1 Cyclotron Road, Berkeley, California 94720, U.S.A., 1993.

[46] F. Olken and D. Rotem. Sampling from spatial databases. In *ICDE '03*, pages 199–208.

[47] R. Shah, S. Roy, S. Jain, and W. Brunette. DATA MULEs: Modeling a three-tier architecture for sparse sensor networks. In *IEEE SNPA Workshop '03*, 2003.

[48] A. A. Somasundara, A. Ramamoorthy, and M. B. Srivastava. Mobile element scheduling for efficient data collection in wireless sensor networks with dynamic deadlines. In *RTSS '04*, pages 296–305.

[49] U. Srivastava and J. Widom. Memory-limited execution of windowed stream joins. In *VLDB '04*, pages 324–335.

[50] S. K. Thompson and G. A. Seber. *Adaptive Sampling*. John Wiley and Sons, Inc., 1996.

[51] W. H. Tok and S. Bressan. Efficient and adaptive processing of multiple continuous queries. In *EDBT '02*, pages 215–232. Springer-Verlag.

[52] I. Vasilescu, K. Kotay, D. Rus, M. Dunbabin, and P. Corke. Data collection, storage, and retrieval with an underwater sensor network. In *SenSys '05*, pages 154–165.

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

[54] Y. Watanabe and H. Kitagawa. Adaptive query optimization method for multiple continuous queries. In *ICDEW '05*, pages 1242–1245.

[55] Y. Wei, S. H. Son, and J. A. Stankovic. RTSTREAM: Real-time query processing for data streams. In *ISORC '06*, pages 141–150.

[56] R. Willett, A. Martin, and R. Nowak. Backcasting: adaptive sampling for sensor networks. In *IPSN '04*, pages 124–133.

[57] Y. L. Y. Tirta, Z. Li and S. Bagchi. Efficient collection of sensor data in remote fields using mobile collectors. In *ICCCN 2004*.

[58] T. Yamane. *Statistics, An Introductory Analysis*. 2nd Ed., New York: Harper and Row., 1967.

[59] D. Yi-Leh Wu, Agrawal and A. El Abbadi. Query estimation by adaptive sampling. In *ICDE '02*, pages 639–648.

[60] P. S. Yu and D. W. Cornell. Buffer management based on return on consumption in a multi-query environment. *The VLDB Journal*, 2(1):1–38, 1993.