# Sampling Based Algorithms for Quantile Computation in Sensor Networks

Zengfeng Huang Lu Wang Ke Yi Yunhao Liu

Hong Kong University of Science and Technology
{huangzf, luwang, yike, liu}@cse.ust.hk

## ABSTRACT

We study the problem of computing approximate quantiles in large-scale sensor networks communication-efficiently, a problem previously studied by Greenwald and Khana [12] and Shrivastava et al. [21]. Their algorithms have a total communication cost of $O(k \log^2 n/\epsilon)$ and $O(k \log u/\epsilon)$, respectively, where $k$ is the number of nodes in the network, $n$ is the total size of the data sets held by all the nodes, $u$ is the universe size, and $\epsilon$ is the required approximation error. In this paper, we present a sampling based quantile computation algorithm with $O(\sqrt{kh}/\epsilon)$ total communication ($h$ is the height of the routing tree), which grows sublinearly with the network size except in the pathological case $h = \Theta(k)$. In our experiments on both synthetic and real data sets, this improvement translates into a 10 to 100-fold communication reduction for achieving the same accuracy in the computed quantiles. Meanwhile, the maximum individual node communication of our algorithm is no higher than that of the previous two algorithms.

## Categories and Subject Descriptors

F.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems

## General Terms

Algorithms

## Keywords

Sensor networks, quantiles

## 1. INTRODUCTION

Sensor networks are large ad-hoc networks of interconnected, battery powered, wireless sensors. They are now being widely deployed to monitor diverse physical variables, such as temperature, sound, activities of wild life and so forth [15, 17, 27]. As technologies mature, sensor networks have reached the scale of thousands of nodes [1, 2] and will get even larger in the near future. However, power consumption remains the biggest obstacle for large-scale deployment for sensor networks as the on-board battery is still the only power source for a sensor node. Since wireless transmission of data is the biggest cause of battery drain [20], in-network aggregation techniques that prevent the nodes from forwarding all the data to the base station are extremely useful for energy conservation in sensor networks. The observation is that for many monitoring tasks, we do not actually need all the measurement data collected by all the sensors, often a succinct aggregate suffices. Computing an aggregate could be much more communication-efficient than transmitting all the data to the base station, in terms of both the *total communication*, as well as the *maximum individual node communication*.

Simple aggregates such as MAX, MIN, SUM, COUNT can be computed easily and efficiently [14]: The nodes first organize them into a spanning tree rooted at the base station. Then starting from the leaves, the aggregation propagates upwards to the root. When a node receives the aggregates from its children, it computes the aggregate of these aggregates and its own data, which equals the aggregate of all the data in the node's subtree, and forwards it to its parent. Such a simple approach works due to the *decomposable* property of these aggregates: for any two disjoint sets $S_1$ and $S_2$, the aggregate of $S_1 \cup S_2$ can be computed from the individual aggregates of $S_1$ and $S_2$. Some other aggregates such as AVERAGE can also be computed this way since it is derived from two decomposable aggregates SUM and COUNT, though it is not decomposable itself. Letting $k$ be the number of nodes in the sensor network, it is clear that a decomposable aggregate can be computed with $O(k)$ total communication and $O(1)$ maximum node communication (assuming each node has $O(1)$ children).

However, these simple aggregates are not expressive enough. A *quantile summary*, which allows one to extract the $\phi$-quantile for any $0 < \phi < 1$ of the underlying data, much better characterizes the data distribution. Recall that the *$\phi$-quantile* of a set $S$ of $n$ data values from a totally ordered universe is the value $a$ with rank $r(a) = \lfloor \phi n \rfloor$ in $S$; the *rank* of any value $x$, $r(x)$, is the number of values in $S$ smaller than $x$. For ease of presentation we assume that all values in $S$ are distinct; such an assumption can be easily removed by using any consistent tie-breaker. The quantiles are also called the *order statistics* and *equi-depth histograms*, and are very useful in a variety of data analytical tasks as they offer a lot more insight into the underlying data than the simple,


*SIGMOD'11,* June 12–16, 2011, Athens, Greece.

single-valued aggregates. Because a quantile summary that returns the accurate quantiles must contain the entire data set, an $\epsilon$-approximation is usually allowed: an *$\epsilon$-approximate $\phi$-quantile* is a value with rank between $(\phi-\epsilon)n$ and $(\phi+\epsilon)n$. This additive error definition is the one that has been mostly adopted in the literature [5, 6, 10–12, 18, 21, 26], though multiplicative errors have also been considered [7, 28]. Note that since the error is additive, the $\epsilon$ should be set small, usually on the order of 0.01% to 1% [6]. With an $\epsilon$-approximation allowed, an (approximate) quantile summary could just retain the $1/\epsilon$ values that rank at $0, \epsilon n, 2\epsilon n, 3\epsilon n, \ldots$, respectively. Then for any $\phi$, we return the value with the largest rank in the summary that is no larger than $\phi n$. One can also easily argue that $\Omega(1/\epsilon)$ is the theoretical minimum size of such a summary.

## 1.1 Previous results

In 2004, two back-to-back, well cited papers by Greenwald and Khanna [12] and Shrivastava et al. [21] studied the problem of computing quantile summaries in a communication-efficient manner in sensor networks. Both of them still follow the "decomposable" approach described earlier. But unlike SUM or MAX, a quantile by itself is not decomposable (for this reason it is also called a *holistic aggregate* in the literature [5]). So the challenge was to design a decomposable summary that contains enough information so that the approximate quantiles can be extracted. Their solutions only differ in the decomposable quantile summaries used. The *GK summary* [12] has size $O(\log^2 n/\epsilon)$ where $n$ is the total number of data values in the network, while the *q-digest* [21] has size $O(\log u/\epsilon)$ where $u$ is the size of the universe from which the data values are drawn. The two extra factors $O(\log^2 n)$ and $O(\log u)$ are not strictly comparable, but the GK summary is theoretically more general as it supports unbounded universes (assuming, of course, any value in the universe takes one unit of storage). The GK summary has two additional variants with size $O(h/\epsilon)$ and $O(\log n \log(h/\epsilon)/\epsilon)$, respectively, where $h$ is the height of the routing tree. They can be better than the basic version when the routing tree is well-balanced.

Specifically, both algorithms [12, 21] start from the leaves and compute the GK summary (resp. q-digest) upwards. Each node first computes a summary of its own data, and then combines it with all the summaries it receives from its children. This produces an aggregated summary that incorporates all the data in the node's subtree, which is then forwarded to its parent. It is clear that the individual node communication is equal to the summary size, while the total communication is $k$ times that.

## 1.2 Our results

In this paper, we present a new algorithm for computing quantile summaries in sensor networks, with an expected total communication of $O(\sqrt{kh}/\epsilon)$. The computed quantile summary allows one to extract an $\epsilon$-approximate $\phi$-quantile with constant probability for any $0 < \phi < 1$. Depending on various physical situations, the height of the tree, $h$, could range from $\log k$ to $k$, but usually does not exceed $\sqrt{k}$ (which happens when the sensor nodes form a grid structure). Thus the communication is always less than that of the previous algorithms. More importantly, except in the pathological case $h = \Theta(k)$, the cost of our algorithm grows *sublinearly* in the size of the network, leading to excellent scalability.

| | total comm. | max node comm. |
|---|---|---|
| q-digest [21] | $k \log u/\epsilon$ | $\log u/\epsilon$ |
| GK algorithm [12] | $k \log^2 n/\epsilon$ | $\log^2 n/\epsilon$ |
| GK algorithm v2 | $k \log n \log(h/\epsilon)/\epsilon$ | $\log n \log(h/\epsilon)/\epsilon$ |
| GK algorithm v3 | $kh/\epsilon$ | $h/\epsilon$ |
| new | $\sqrt{kh}/\epsilon$ | $\log(k/h)/\epsilon$ |

**Table 1: The asymptotic communication costs of the algorithms, where $n$ is the total number of data values, $u$ is the universe size, $k$ is the network size, $h$ is the height of the routing tree, and $\epsilon$ is the error parameter.**

Meanwhile, the maximum individual node communication of our algorithm is $O(\log(k/h)/\epsilon)$, which is also strictly better than any of the previous algorithms, as $\log k < h < k < n < u$. A comparison of the communication costs of these algorithms is given in Table 1.

Note that for relatively large $\epsilon$, say a constant, $O(\sqrt{kh}/\epsilon)$ could be much smaller than $k$, which means that we can compute the quantiles without contacting all nodes. This may seem surprising, as even computing or approximating a simple aggregate, like SUM or COUNT, needs $\Theta(k)$ communication. This "surprise" comes with the assumption that our algorithm knows the values of $n$ and $k$; otherwise, we need $O(k)$ communication to compute them first. We separated the cost of computing $n$ and $k$ from that of the quantile problem itself for the following reasons: (1) It simplifies the bound and makes the core problem of quantile computation stand out; otherwise we should always include an additive $O(k)$ term. (2) Applications usually use quantiles to track the distribution of the underlying data, so will compute quantile summaries periodically. The values of $n$ and $k$ from period to period are unlikely to change much, while our algorithm actually only needs to know $n$ and $k$ within a constant factor. So we can usually repeatedly execute it without refreshing $n$ and $k$. Anyways, in practice we typically have $k \ll \sqrt{kh}/\epsilon$, and the $O(k)$ term can be neglected.

Note that for any algorithm using decomposable summaries, the total communication is always at least $\Omega(k/\epsilon)$, as any such summary must have size $\Omega(1/\epsilon)$. To break this barrier, we deviate from the decomposable framework: The message a node sends to its parent does not necessarily contain a quantile summary of the data in its subtree. Nevertheless, we will make sure that the base station in the end will have a valid quantile summary for the entire data set. In particular, our algorithm uses larger messages (but of size at most $O(\log(k/h)/\epsilon)$) for nodes near the base station, while nodes far away send small or even no messages. This is in contrast with the previous approaches where all nodes use essentially the same message size (ignoring polylog factors).

The improvement of our algorithm for the maximum individual node communication is not as drastic as that for the total communication. In fact, it is not difficult to show an $\Omega(1/\epsilon)$ lower bound on the maximum node communication for any quantile algorithm, decomposable or not, and all the algorithms already come close to this theoretical limit within polylog factors, with our log factor being the smallest. Some may say that the maximum node cost is more important [12], but we would argue the other way round, for the following reasons: 1) The network will not be disconnected due to a single node exhausting its battery. A sensor network usually has enough redundancy so that rerouting is possible for

a permissible number of node failures. 2) We can install larger batteries for sensor nodes that are expected to have larger power consumptions, e.g. those near the base station, or simply deploy more sensors there to increase redundancy. When we have good provisioning, the total communication cost, rather than the maximum individual node cost, will become the determining factor for the longevity of the network. In fact, most previous papers on data aggregation indeed used total communication as the primary measure of energy efficiency, e.g. [4, 16, 22]. The previous work on the quantile problem [12, 21] did not emphasize it as much because for those algorithms, the total communication is just $k$ times the (fixed) message size.

Our algorithm is based on sampling. However, simply taking a random sample of the data and computing the quantiles on the sample is not accurate enough, as it is well known that to achieve an $\epsilon$-error with constant probability, a random sample of size $\Theta(1/\epsilon^2)$ needs to be drawn [23]. This results in $O(h/\epsilon^2)$ total communication and $O(1/\epsilon^2)$ maximum node communication. To improve accuracy (or equivalently, to reduce size), we augment the random sample with additional information about the data, together with several new ideas that we briefly outline in Section 1.4 and develop in stages in later sections. In fact, the total communication of our algorithm is $O(\min\{\sqrt{kh}/\epsilon, h/\epsilon^2\})$, i.e., it is always no worse than simple random sampling, but we will avoid carrying along with the "min" throughout the paper to simplify exposition.

Since our algorithm is based on random sampling, it provides a probabilistic guarantee, that any $\phi$-quantile can be extracted within error $\epsilon$ with a constant probability which can be made arbitrarily close to 1. While the GK algorithm and the q-digest provide a worst-case $\epsilon$-error guarantee. However, we would be happy with a probabilistic guarantee, since transmission errors and link failures are common in sensor networks, a theoretical worst-case guarantee becomes a probabilistic one in practice anyway. Nevertheless, to ensure a fair comparison, in the experiments (Section 5) we did not simply set the required $\epsilon$ and compare the communication costs. Instead, we measure the *actual* average and maximum error of 99 quantiles for $\phi = 1\%, 2\%, \ldots, 99\%$, and compare all the algorithms in terms of the communication-error trade-off curve.

## 1.3 Related work

Data aggregation in sensor networks has been a topic of intensive studies for the past years. Below we only review the most relevant work; please refer to the surveys [9, 24] for more comprehensive results in this area.

Data aggregation techniques can be broadly classified into two categories: *tree-based* approaches and *multi-path* approaches. In a tree-based approach, the nodes organize themselves into a routing tree. Typically the tree is built from the base station in a breadth-first manner: All nodes within communication range with the base station become level-one nodes. Then a node $u$ that can reach a level-one node $v$ becomes a level-two node, with $v$ being the parent of $u$, and so on so forth. When a link fails, the child will try to find a new parent, but at any time, a node has only one parent and the aggregation is performed along a tree. Most data aggregation techniques, including all the quantile algorithms, are tree-based. In a *multi-path* approach, a node has multiple parents and will broadcast its message to all of them. Such an approach is more robust against link failures, but also requires more communication, as now the algorithm has to be designed to be insensitive to the duplication of messages. With a multi-path approach, even a simple aggregate like COUNT or SUM cannot be computed exactly. To obtain an $\epsilon$-approximation of COUNT or SUM, each node will need to send a message of size $O(1/\epsilon^2)$ [4, 19]. Currently there are no multi-path algorithms for the quantile problem. There is also a hybrid approach [16] that combines the benefits of tree-based and multi-path approaches, where it uses tree aggregation when the link failure rate is low to gain better communication efficiency, while adopts a multi-path strategy when the failure rate is high.

The closely related *heavy hitters* problem has also received a lot of attention, where the goal is to compute a summary of a multiset of size $n$ from which we can estimate the frequency of any item up to an additive error of $\epsilon n$. It is well known [6] that this problem can be reduced to the $\epsilon'$-approximate quantile problem for some $\epsilon' = \Theta(\epsilon)$, by simply using some tie breaker to convert the multiset into a set (say, padding different lower-order bits), and then asking quantile queries with $\phi = \epsilon', 2\epsilon', \ldots$. Thus the previous quantile algorithms [12, 21], as well as our new algorithm, also solve the heavy hitters problem. But of interest is whether the heavy hitters problem can be solved more efficiently, as in the case of the streaming model, where the heavy hitters problem can be solved in $O(1/\epsilon)$ space [6] while the best quantile algorithm needs $O(\log n/\epsilon)$ space [11]. Manjhi et al. [16] proposed a deterministic algorithm for computing the heavy hitters with total communication $O(k/\epsilon)$, but the bound only holds for a class of "nice" routing trees that they define. It is still an open question whether there are better deterministic heavy hitter algorithms for arbitrary routing trees. For randomized algorithms, one can use the *count-min* sketch [8] in the decomposable framework, leading to total communication $O(k/\epsilon)$ for any routing tree. Our algorithm improves this to $O(\sqrt{kh}/\epsilon)$.

Quantile summaries, as fundamental statistics and a useful data analytical tool, have been extensively studied in several other settings. Munro and Paterson [18] studied how to compute an exact quantile with limited memory and multiple passes, and also showed that approximation is necessary if only one pass is allowed. The best one-pass (i.e., streaming) algorithm for computing approximate quantile summaries is due to Greenwald and Khana [11], whose algorithm uses $O(\log n/\epsilon)$ space. Gilbert et al. [10] and Cormode and Muthukrishnan [8] studied how to maintain a quantile summary under both insertions and deletions using small space. Finally, Cormode et al. [5] and Yi and Zhang [26] studied how to track the quantile summary over distributed data sets as they evolve.

## 1.4 Roadmap

We develop our algorithm in three stages. In Section 2 we first present the algorithm in the *flat model*, in which all nodes are directly connected to the base station. This algorithm has $O(\sqrt{k}/\epsilon)$ total communication and $O(1/\epsilon)$ maximum individual node communication. Simply running the algorithm on a spanning tree of height $h$ would result in $O(h\sqrt{k}/\epsilon)$ total communication; even worse, the maximum individual node communication could be as high as $O(\sqrt{k}/\epsilon)$ since all the traffic might have to go through a single node. In Section 3 we develop techniques that combine

the messages a node receives before it forwards its message to its parent. This results in an $O(\log k/\epsilon)$ maximum message size. Finally in Section 4, we use a tree partitioning technique to improve the total communication cost to the claimed $O(\sqrt{kh}/\epsilon)$ bound.

## 2. THE FLAT MODEL

In this section we first describe our algorithm in the flat model, in which each node is directly connected to the base station. Let the set of data values at node $i$ be $S_i, i = 1, \ldots, k$, and let $n$ be the total number of data values.

**The algorithm.** The algorithm is very simple. Each node first independently samples each of its data values with some probability $p$ (to be determined later). For each sampled value $a$, it computes its *local rank* $r(a, i)$ at node $i$, i.e., the rank of $a$ in set $S_i$. Then it simply sends all the sampled values and their local ranks to the base station.

We first show how a value-to-rank query can be answered at the base station from the information it receives, namely, given any value $x$, we need to estimate $r(x)$, the rank of $x$ in $\bigcup_i S_i$. After this, quantile (rank-to-value) queries can be easily answered. Let $pred(x, i)$ be the predecessor of $x$ in the sampled values sent from node $i$, namely, the largest value no larger than $x$; note that $pred(x, i)$ may not exist. We show below that

$$\hat{r}(x,i) = \begin{cases} r(pred(x,i),i) + 1/p, & \text{if } pred(x,i) \text{ exists;} \\ 0, & \text{else} \end{cases}$$

is an unbiased estimator of $r(x, i)$, the local rank of $x$ in $S_i$. Then, we can estimate the global rank of $x$ as

$$\hat{r}(x) = \sum_i \hat{r}(x,i).$$

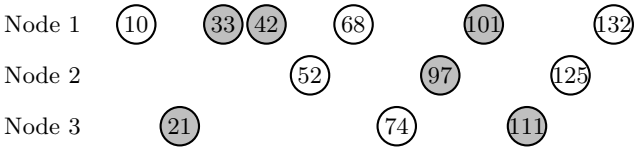


**Figure 1: There are 3 nodes, each of which holds a set of integers, and the shaded integers are sampled (the sample rate here is 1/2).**

See Figure 1 for an example. Suppose we want to query for the rank of 80. In this example, $\hat{r}(80, 1) = 2+2 = 4$ since in the sample from Node 1, the local rank of 80's predecessor, 42, is 2 and the sample probability is 1/2. Similarly, $\hat{r}(80, 2) = 0$ and $\hat{r}(80, 3) = 0 + 2 = 2$, so the estimated global rank of 80 is $\hat{r}(8) = 4 + 0 + 2 = 6$, whereas its actual global rank is 7.

**Analysis.** The key to showing that $\hat{r}(x)$ estimates the global rank of $x$ accurately is the following lemma.

LEMMA 1. *For any $x$, $\hat{r}(x, i)$ is an unbiased estimator for $r(x, i)$, with variance* $\mathrm{Var}[\hat{r}(x,i)] \le \dfrac{1-(1-p)^{r(x,i)}}{p^2}$.

PROOF. Consider the random variable

$$X = \begin{cases} r(x,i) - r(pred(x,i),i), & \text{if } pred(x,i) \text{ exists;} \\ r(x,i) + 1/p, & \text{else.} \end{cases}$$

Note that $\hat{r}(x, i) = r(x, i) - X + 1/p$. So we just need to show that $\mathbf{E}[X] = 1/p$ and bound $\mathrm{Var}[X]$.

Starting from $x$ and walking to smaller values, we observe that $X$ represents the number of values we see until the first sampled value $pred(x, i)$ in set $S_i$, when it exists. When all the $r(x, i)$ values smaller than $x$ are not sampled, $X$ is set to $r(x, i) + 1/p$. Therefore, we have (detailed derivations are omitted; $r(x, i)$ is shorthanded as $r$)

$$\mathbf{E}[X] = \sum_{\ell=1}^{r} \ell p(1-p)^{\ell-1} + (1-p)^r(r+1/p) = 1/p.$$

The variance is

$$\begin{aligned} \mathrm{Var}[X] &= \mathbf{E}[X^2] - \mathbf{E}[X]^2 \\ &= \sum_{\ell=1}^{r} \ell^2 p(1-p)^{\ell-1} + (1-p)^r(r+1/p)^2 - 1/p^2 \\ &= \frac{(1-(1-p)^r)(1-p)}{p^2} \le \frac{1-(1-p)^{r(x,i)}}{p^2}. \end{aligned}$$

□

Since the global rank of $x$ is the sum of the local ranks, $\hat{r}(x)$ is an unbiased estimator of $r(x)$ with variance $\sum_{i=1}^{k} \mathrm{Var}[\hat{r}(x,i)]$. We bound $\mathrm{Var}[\hat{r}(x,i)]$ in two ways. First it is clear that $\mathrm{Var}[\hat{r}(x,i)] \le 1/p^2$, and hence $\sum_i \mathrm{Var}[\hat{r}(x,i)] \le k/p^2$. Thus, by setting $p = \Theta(\sqrt{k}/\epsilon n)$, the variance becomes $O((\epsilon n)^2)$. By Chebyshev's inequality, this means that $\hat{r}(x)$ approximates $r(x)$ within an additive error of $\epsilon n$ with constant probability. This constant probability can be made arbitrarily close to 1 by enlarging $p$ by appropriate constant factors. In this case the total communication is $O(pn) = O(\sqrt{k}/\epsilon)$.

Alternatively we can bound $\mathrm{Var}[\hat{r}(x,i)]$ as

$$\mathrm{Var}[\hat{r}(x,i)] \le \frac{1-(1-pr(x,i))}{p^2} = \frac{r(x,i)}{p},$$

so

$$\mathrm{Var}[\hat{r}(x)] = \sum_i \mathrm{Var}[\hat{r}(x,i)] \le \frac{1}{p}\sum_i r(x,i) \le \frac{n}{p}.$$

This means that when $p$ is set to $\Theta(1/\epsilon^2 n)$, $\mathrm{Var}[\hat{r}(x)]$ is also $O((\epsilon n)^2)$. In this case the total communication is $O(1/\epsilon^2)$, namely the same as simple random sampling. Therefore, the algorithm has a total communication of $O(\min\{\sqrt{k}/\epsilon, 1/\epsilon^2\})$. In the rest of the paper, we will only consider the interesting and typical case when $\sqrt{k}/\epsilon < 1/\epsilon^2$, i.e., $k < 1/\epsilon^2$, to avoid carrying the "min" around.

**Reducing individual node communication.** The algorithm described above has the desired total communication $O(\sqrt{k}/\epsilon)$, but all this traffic could be from one node, if it dominates the entire data set. Below we show how to limit the individual node communication to $O(1/\epsilon)$. We classify the nodes into those with more than $n/\sqrt{k}$ data values and those with at most that. If a node $i$ has $|S_i| \le n/\sqrt{k}$ values, it still uses the previously determined sample probability $p_i = p$; if $|S_i| > n/\sqrt{k}$, it will use a smaller sample probability $p_i = 1/\epsilon|S_i|$. The estimator correspondingly becomes

$$\hat{r}(x,i) = \begin{cases} r(pred(x,i),i) + 1/p_i, & \text{if } pred(x,i) \text{ exists;} \\ 0, & \text{else.} \end{cases}$$

It is clear that now a node samples at most $O(1/\epsilon)$ values in expectation. To see that the estimator is still accurate,

by Lemma 1, $\hat{r}(x)$ is still an unbiased estimator of $r(x)$ with variance

$$\begin{aligned}\mathrm{Var}[\hat{r}(x)] &\leq \sum_{i=1}^{k} 1/p_i^2 = \sum_{i\,:\,|S_i|>\frac{n}{\sqrt{k}}} (\epsilon n_i)^2 + \sum_{i\,:\,|S_i|\leq\frac{n}{\sqrt{k}}} \frac{(\epsilon n)^2}{k} \\ &\leq \left(\sum_{\{i\,:\,|S_i|>\frac{n}{\sqrt{k}}\}} (\epsilon n_i)\right)^2 + (\epsilon n)^2 \leq 2(\epsilon n)^2.\end{aligned}$$

Thus $\hat{r}(x)$ is still an $\epsilon$-approximation of $r(x)$ with constant probability.

**Quantile queries.** We have shown how to answer value-to-rank queries using the summary structure at the base station. Quantile (rank-to-value) queries can also be answered easily, as follows. For each sampled data value $a$ received by the base station from node $i$, we first estimate its global rank $\hat{r}(a)$ as

$$\hat{r}(a) = r(a,i) + \sum_{j\neq i} \hat{r}(a,j).$$

Now, given a required rank $r$, we simply return the sampled value $x$ that has the closest estimated rank $\hat{r}(x)$ to $r$. Below we argue that its true rank, $r(x)$, is away from $r$ by at most $\epsilon n$ with constant probability. Assume that $r$ is between the estimated ranks of two consecutive values $x$ and $y$ in the sample, i.e., $\hat{r}(x) \leq r \leq \hat{r}(y)$. Consider the following three events:

$$\begin{aligned}&1) \quad \hat{r}(x) - \frac{1}{2}\epsilon n \leq r(x) \leq \hat{r}(x) + \frac{1}{2}\epsilon n.\\ &2) \quad \hat{r}(y) - \frac{1}{2}\epsilon n \leq r(y) \leq \hat{r}(y) + \frac{1}{2}\epsilon n.\\ &3) \quad r(y) - r(x) \leq \epsilon n.\end{aligned}$$

When all three events happen, one can verify that $x$ or $y$, whoever has the closest estimated rank to $r$, must have its true rank within $\epsilon n$ to $r$.

By appropriately adjusting the constants, we can make sure that events 1) and 2) each happen with probability 8/9 (say). Since the sample probability is at least $1/\epsilon|S_i| \geq 1/\epsilon n$, the number of missed values between $x$ and $y$ is no more than $\epsilon n$ with constant probability. Again this constant can be boosted to 8/9. Then by a union bound, all three events happen together with probability at least 2/3.

THEOREM 1. *Our algorithm in the flat model has $O(\sqrt{k}/\epsilon)$ total communication and $O(1/\epsilon)$ maximum individual node communication, and answers an $\epsilon$-approximate quantile query with constant probability.*

# 3. THE TREE MODEL

There are two challenges in extending the flat model algorithm to a general routing tree. First, if each node simply sends its message through its ancestors in the routing tree to the base station without any data reduction, an intermediate node might see too much traffic going through. This could result in an $O(\sqrt{k}/\epsilon)$ maximum individual node communication. Second, in terms of total communication, simply running the flat model algorithm in the tree model would result in $O(h\sqrt{k}/\epsilon)$ communication as a message needs $O(h)$ hops to reach the base station. This section will resolve the first issue while Section 4 the second.

## 3.1 Basic ideas

From Theorem 1 we know that each node's own message has size at most $O(1/\epsilon)$. Problems arise when a node has too many descendants whose messages need to be forwarded. Our idea is to merge these messages in a systematic way so as to reduce their size.

The unit of our *merge* operation is a sample $s$ taken from a *ground set* $D(s)$. Let $n(s)$ denote the size of the ground set $D(s)$, and we store $n(s)$ together with $s$. We say $s$ is a *small* sample if $n(s) < n/\sqrt{k}$ and a *large* sample if $n(s) \geq n/\sqrt{k}$. Initially, each such sample $s$ is generated by a node $i$ from its own data set $D(s) = S_i$. Recall from the previous section that the initial samples have the following properties:

(P1) The ground sets of the samples are disjoint, and their union is the entire data set.

(P2) Each value in $D(s)$ has been sampled to $s$ with some equal probability $p(s)$; in particular, $p(s) = \sqrt{k}/\epsilon n$ if $s$ is a small sample and $p(s) = 1/\epsilon n(s)$ if it is a large sample.

(P3) Each sampled value $a$ in $s$ is associated with $r(a, D(s))$, the local rank of $a$ in $D(s)$.

Recall that an immediate consequence of (P2) is that each sample has size at most $O(1/\epsilon)$.

For a large sample $s$, we define its class number as

$$c(s) = \lfloor \log(n(s)\sqrt{k}/n) \rfloor.$$

It is clear that $0 \leq c(s) \leq \log\sqrt{k}$. When a node has received a number of samples from its children, together with one of its own, it will first check if the ground sets of all the small samples have a combined size of at least $n/\sqrt{k}$. If so it will merge all the small samples into a large sample. Next it will repeatedly merge two large samples of the same class into one in the next class, until no two large samples are in the same class. As a result, there will be at most one large sample per class left. During the merge operation, we will also ensure that the three properties above are maintained. As a result, since all the small samples, if there are any, have their combined ground set smaller than $n/\sqrt{k}$, the total size of all the small samples is $O(1/\epsilon)$, and because each large sample has size at most $O(1/\epsilon)$, the node will eventually send out a message of size $O(\log k/\epsilon)$.

When merging two samples $s_1$ and $s_2$ and producing a merged sample on the ground set $D(s_1) \cup D(s_2)$, properties (P1) and (P2) are relatively easy to maintain, by appropriately subsampling the values in $s_1$ and $s_2$ based on $p(s_1)$ and $p(s_2)$. However, (P3) is difficult to guarantee. In fact, because we only have a sample $s_2$ from $D(s_2)$, for any value $a$ subsampled to the merged sample from $s_1$, we cannot really compute its exact rank in $D(s_2)$, and vice versa. So we will instead estimate its rank in $D(s_2)$ based on $s_2$. Thus, we will relax property 3) to the following:

(P3′) Each sampled value $a$ in $s$ is associated with $\hat{r}(a, D(s))$, which is an unbiased estimator of $r(a, D(s))$, the local rank of $a$ in $D(s)$.

Now we need to be careful since the errors in these estimated ranks will propagate as more merges are performed, and we need to make sure that when the samples reach the base station, the accumulated error should not exceed $\epsilon n$. Below we first present the relatively simple merging algorithm, and defer the more complicated analysis to later.

## 3.2 The merging algorithm

As described above, if all the small samples have their combined ground set smaller than $n/\sqrt{k}$, we will not do any merges since their total sample size is $O(1/\epsilon)$. Otherwise, we use the following MERGE-SMALL operation to merge them into a large sample.

MERGE-SMALL: Let $s_1, s_2, \ldots, s_m$ be all the small samples, such that $\sum_i n(s) \geq n/\sqrt{k}$. Let $s$ be the merged sample, and let $n(s) = \sum_i n(s_i)$ be the size of the combined ground set. Note that $s$ will be a large sample, so its sample probability should be $p(s) = 1/\epsilon n(s)$. To form the sample, we can simply subsample each data value in all the $s_i$'s with probability $\frac{p(s)}{p(s_i)} = \frac{n}{n(s)\sqrt{k}}$. For each $a$ thus sampled, if it is from $s_i$, we estimate its local rank in the combined ground set as

$$\hat{r}(a, D(s)) = r(a, i) + \sum_{j \neq i} \hat{r}(a, D(s_j)),$$

where

$$\hat{r}(a, D(s_j)) = \begin{cases} r(pred(a, s_j), D(s_j)) + 1/p(s_j), & \\ & \text{if } pred(a, s_j) \text{ exists;} \\ 0, & \text{else.} \end{cases}$$

As before, here $pred(a, s_j)$ denotes the predecessor of $a$ in the sample $s_j$.

It is easy to see that MERGE-SMALL maintains properties (P1), (P2), and (P3′) since by Lemma 1, $\hat{r}(a, D(s_j))$ is an unbiased estimator of $r(a, D(s_j))$.

After executing MERGE-SMALL, we will repeatedly execute MERGE-LARGE to merge the large samples. Unlike MERGE-SMALL, we apply MERGE-LARGE only on pairs of large samples of the same class, one pair at a time, progressively from the low classes to high. More precisely, starting from class $c = 0$, as long as there are at least two large samples in this class, we merge them with MERGE-LARGE, to form a sample in class $c + 1$. When there is one or no sample left in class $c$, we move on to class $c + 1$. This idea is similar to that in [12], but because of our way of sampling and the fact that we deviate from the decomposable framework, the total size of our merged samples is smaller than that of [12] by a logarithmic factor.

MERGE-LARGE: Let $s_1$ and $s_2$ be two large samples to be merged. Let $s$ be the merged sample, and set $n(s) = n(s_1) + n(s_2)$. As $s$ has a sample probability $p(s) = 1/\epsilon n(s)$, we subsample each data value in $s_1$ with probability $p(s)/p(s_1) = n(s_1)/n(s)$ and subsample each data value in $s_2$ with probability $p(s)/p(s_2) = n(s_2)/n(s)$. For each subsampled value $a$, if it is from $s_1$, we estimate its rank in $D(s)$ as

$$\hat{r}(a, D(s)) = \hat{r}(a, D(s_1)) + \hat{r}(a, D(s_2)),$$

where $\hat{r}(a, D(s_1))$ is the rank (either exact or approximate) that $a$ carries from $s_1$, and $\hat{r}(a, D(s_2))$ is computed from $s_2$ similarly as before

$$\hat{r}(a, D(s_2)) = \begin{cases} \hat{r}(pred(a, s_2), D(s_2)) + 1/p(s_2), & \\ & \text{if } pred(a, s_2) \text{ exists;} \\ 0, & \text{else,} \end{cases}$$

except that $\hat{r}(pred(a, s_2), D(S_2))$ could now also be an estimate rather than the exact rank of $pred(a, s_2)$ in $D(S_2)$. The case where $a$ is from $s_2$ is handled symmetrically.

Let us see an example of merging two messages. Suppose $n/\sqrt{k} = 100$, and the two messages contain summaries $\{s_1, s_2, s_3\}$ and $\{t_1, t_2, t_3\}$, with ground set sizes 80, 400, 800, and 60, 400, 3200 respectively. So $s_1$ and $t_1$ are small samples, and we merge them into $s_1'$ using MERGE-SMALL. The resulting merged sample has class number 0. We then find that $s_2$ and $t_2$ are both in class 2, and merge them together into a new summary with class number 3. It is further merged with $s_3$, getting a summary $s_2'$ with class number 4. Now we are done, since the summaries left are $\{s_1', s_2', t_3\}$, all of which have different class numbers.

When all the samples have been sent to the base station, we can use exactly the same query algorithms as in the flat model to answer value-to-rank and rank-to-value queries using these samples, just that now the local ranks for the sampled values are estimates of the actual local ranks in the respective ground sets.

## 3.3 Error analysis

It easily follows from the merging algorithm that properties (P1), (P2), and (P3′) are all maintained, but it remains to show that when the base station has received all the samples, an $\epsilon$-approximate quantile query can still be answered with constant probability.

LEMMA 2. *For any large sample $s$ resulted from* MERGE-SMALL*, the estimated local rank $\hat{r}(a, D(s))$ has variance at most $m(\epsilon n)^2/k$, where $m$ is the number of merged small samples.*

PROOF. Because each small sample is an initial sample with sample probability $p = \sqrt{k}/\epsilon n$, the lemma directly follows from Lemma 1. □

LEMMA 3. *Let $s$ be any large sample of class $c(s)$. For any data value $a \in s$, its estimated rank $\hat{r}(a, D(s))$ has variance at most $m(\epsilon n)^2/k + (\epsilon 2^{c(s)+1} n)^2/k$, where $m$ is the number of small samples whose ground sets are included in $D(s)$.*

PROOF. We will prove by induction on $c(s)$. The base case $c(s) = 0$ is easy to verify: For any large sample of class 0, it is either an initial sample or one produced from MERGE-SMALL. The variance of $\hat{r}(a, D(s))$ is 0 in first case, and at most $m(\epsilon n)^2/k$ in the second case by Lemma 2.

Now we assume the lemma holds for all large samples of class $i$ and proceed to prove it for class $i + 1$. A large sample $s$ with $c(s) = i + 1$ could be produced from MERGE-SMALL, or merged from two large samples $s_1$ and $s_2$ with $c(s_1) = c(s_2) = i$. In the first case, again by Lemma 2, the variance is at most $m(\epsilon n)^2/k$; in the second case, by the induction hypothesis, we have

$$\mathrm{Var}[\hat{r}(a, D(s_j))] \leq m_j(\epsilon n)^2/k + (\epsilon 2^{i+1} n)^2/k,$$

for any $a \in s_j$, where $m_j$ is the number of small samples whose ground sets are included in $D(s_j)$, $j = 1, 2$.

Consider a data value $a \in s$. W.l.o.g., assume that it is subsampled from $s_1$. The rank of $a$ in $D(s)$ is estimated as

$$\hat{r}(a, D(s)) = \hat{r}(a, D(s_1)) + \hat{r}(a, D(s_2)).$$

The variance of $\hat{r}(a, D(s_1))$, by the induction hypothesis, is at most

$$m_1(\epsilon n)^2/k + (\epsilon 2^{i+1} n)^2/k. \quad (1)$$

The variance of $\hat{r}(a, D(s_2))$, if the local ranks in $s_2$ were accurate, by Lemma 1 is at most

$$1/p(s_2)^2 = (\epsilon n(s_2))^2 \leq (\epsilon 2^{i+1} n)^2/k. \quad (2)$$

Since now we only have an estimate for the rank of $pred(a, s_2)$ with variance

$$m_2(\epsilon n)^2/k + (\epsilon 2^{i+1} n)^2/k, \quad (3)$$

by the *law of total variance*, $\mathrm{Var}[\hat{r}(a, s)]$ is the sum of (1), (2) and (3). Thus

$$\begin{aligned} \mathrm{Var}[\hat{r}(a,s)] &\leq (m_1 + m_2)(\epsilon n)^2/k + 3(\epsilon 2^{i+1} n)^2/k \\ &\leq m(\epsilon n)^2/k + (\epsilon 2^{i+2} n)^2/k, \end{aligned}$$

which completes the induction. □

THEOREM 2. *Our algorithm in the tree model has $O(h\sqrt{k}/\epsilon)$ total communication and $O(\log k/\epsilon)$ maximum individual node communication, and answers an $\epsilon$-approximate quantile query with constant probability.*

PROOF. The communication bounds follow directly from the algorithm description, so we only prove correctness here. Below we only focus on a value-to-rank query, i.e., estimating the rank $r(x)$ of any given value $x$ within error $\epsilon n$; after that, a quantile query can be answered in the same way as in Section 2.

Recall that we use the same algorithm as in the flat model to estimate $r(x)$ from a number of small samples and at most one large sample per class. The total variance from all the small samples is at most $O((\epsilon n)^2)$ according to the analysis in Section 2, since they are the initial samples without merging.

Let $s_0, s_1, \ldots, s_{\log \sqrt{k}}$ be the large samples for each of the classes. The estimated local ranks of $x$ in these samples have two sources of variance: the variance due to the sampling, which is $1/p(s_i)^2$ as in Lemma 1, and the variance of the estimated local rank $\hat{r}(pred(x, s), D(s))$, which can be bounded by Lemma 3. The total variance from the first source is at most

$$\sum_{i=0}^{\log \sqrt{k}} \frac{1}{p(s_i)^2} \leq \sum_{i=0}^{\log \sqrt{k}} (\epsilon 2^{i+1} n)^2/k = O((\epsilon n)^2).$$

The total variance from the second source, by Lemma 3, is at most

$$\begin{aligned} &\sum_{i=0}^{\log \sqrt{k}} (m_i(\epsilon n)^2/k + (\epsilon 2^{i+1} n)^2/k) \\ \leq\ & k(\epsilon n)^2/k + \sum_{i=0}^{\log \sqrt{k}} (\epsilon 2^{i+1} n)^2/k = O((\epsilon n)^2). \end{aligned}$$

Again, the constant in the big-Oh can be made arbitrarily small by enlarging the sample probabilities by constant factors. This means that we can estimate $r(x)$ within $\epsilon n$ error with a constant probability. □

# 4. TREE PARTITIONING

In this section we describe our final improvement of the algorithm, reducing the total communication by another $O(\sqrt{h})$ factor to $O(\sqrt{kh}/\epsilon)$, which is sublinear in $k$ for all $h = o(k)$. The idea is to partition the routing tree into $t$ connected components, each of which contains $O(k/t)$ nodes. Then we conceptually shrink each component into a "super node". These $t$ super nodes form a tree of size $t$ but whose height still could be $h$. Now, if we apply our algorithm of Section 3 on these super nodes, the total communication would be $O(h\sqrt{t}/\epsilon)$. This seems to suggest a $t$ that is as small as possible. However, since a super node is not really one node, but $O(k/t)$ nodes that are connected. To produce an initial sample for a super node and compute the local ranks for the sampled values within the super node, we have to send messages among the $O(k/t)$ nodes. It turns out preparing the initial samples now takes communication $O(k/\epsilon\sqrt{t})$. Thus, setting $t = k/h$ balances these two terms and yields the desired $O(\sqrt{kh}/\epsilon)$ bound. We next elaborate on this idea in the rest of this section.

## 4.1 Tree partitioning

We first partition the routing tree into $O(t) = O(k/h)$ components, each of which has $O(h)$ nodes. To ensure that each component is connected, we may introduce a few *virtual nodes*, by cloning the actual nodes. A virtual node has no data. It sits inside the actual node but logically operates on its own. Note that the tree partitioning phase depends only on the topology of the routing tree, so we can separate it from the actual quantile algorithm and only run it when the tree topology changes.

Below we present a distributed algorithm that does the partitioning in $O(k)$ total communication. We assume that each node is aware of its subtree size; if not this information can be obtained easily using a bottom-up computation with $O(k)$ communication. Each node $u$ maintains a weight $w(u)$ which is initially set to $u$'s subtree size; during the algorithm $w(u)$ will represent the number of unpartitioned nodes in $u$'s subtree. The partitioning algorithm starts by calling PARTITION(root of the tree), and proceeds recursively, as outlined in Algorithm 1.

**Algorithm 1**: PARTITION($u$)

```
1  foreach child v of u do
2      if w(v) ≥ h then PARTITION(v);
3  set w(u) := Σ_v w(v) + 1 for all children v of u;
4  while w(u) > h do
5      if u has children v_1, ..., v_l s.t. h/2 ≤ Σ_i w(v_i) ≤ h
       then
6          put all the unpartitioned nodes in the subtrees
           of v_1, ..., v_l into one component;
7          mark all these nodes as "partitioned";
8          if l ≥ 2, create a virtual node at u as the root of
           this component;
9          set w(u) := w(u) − w(v_1) − ··· − w(v_l);
10         set w(v_i) := 0 for i = 1, ..., l;
```

Note that when PARTITION(root) returns, the root might still have at most $h$ unpartitioned nodes below. Then we simply allocate these nodes into the last component. An example of the partitioning obtained by this algorithm is shown in Figure 2.

LEMMA 4. *The partitioning algorithm partitions an arbitrary routing tree into $O(k/h)$ connected components, each of size $O(h)$. At most one virtual node is created for each component.*

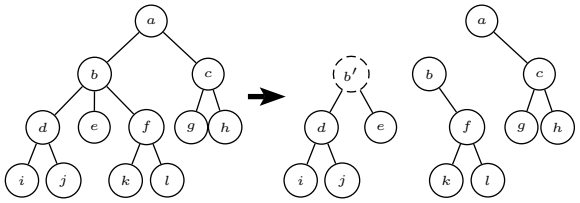


**Figure 2: An example of the tree partition algorithm when $k = 12$ and $h = 4$. The whole tree is partitioned into 3 components. Node $b'$ is a virtual node added by the algorithm, and in real life its role can be played by node $b$.**

PROOF. It is clear that the algorithm only produces components of size between $h/2$ and $h$, and at most one component (the last one) of size between 1 and $h + 1$. But we still need to argue that all nodes must have been partitioned eventually. To do so, we show that when PARTITION($u$) finishes, $u$ has at most $h$ unpartitioned nodes in its subtree.

We prove it by induction. When $u$ is a leaf, PARTITION($u$) does nothing and the claim is certainly correct. Now consider an internal node $u$. By the induction hypothesis when lines 1–2 are done, each of $u$'s children $v$ has at most $h$ unpartitioned nodes below, namely, $w(v) \leq h$. Thus, as long as the sum of their weights is at least $h$, line 5 will always evaluates to true. In fact, we can first check if there is any $v$ with $h/2 \leq w(v) \leq h$. If there is one that already satisfies the condition. Otherwise all of them have $w(v) < h/2$. We can then simply collect them one by one until the sum falls between $h/2$ and $h$. Therefore, we can continue producing components of size between $h/2$ and $h$ until $w(u) \leq h$. This finishes the induction and hence the proof. □

## 4.2 Quantile algorithm on the partitioned tree

On the partitioned tree, we run our tree model algorithm of Section 3 by treating each component as a super node. Recall that there are only $t = O(k/h)$ super nodes. Let $S_i$ be the set of data values in the $i$-th super node. The previous analysis suggests a sampling rate of $p = \sqrt{t}/\epsilon n$ for a super node with less than $n/\sqrt{t}$ data values, and $p = 1/\epsilon|S_i|$ if $|S_i| \geq n/\sqrt{t}$. After the sample is drawn, we also needed to compute the local ranks of the sampled values in $S_i$.

However, now $S_i$ does not reside on one single node, but distributed among $O(h)$ nodes in the component, so we have to pay communication to compute the local ranks. Specifically, each node in a super node first samples its own data, and then the sampled values to the root of the component. The root of the component, after receiving all the samples, broadcasts them to everyone in the component. Now every node in the components has a copy of the sample drawn from the whole component, and thus can compute their ranks within its own data set. Finally we aggregate these local ranks in a bottom-up fashion to the root of the component, which is actually the same as performing multiple SUM aggregations within the component, one per sampled value.

After the root of each component has prepared its initial sample and the associated local ranks, we can simply run our previous tree model algorithm on these initial samples. More precisely, starting from these component roots, we send the samples hop by hop to the base station. As before, when an intermediate node has received multiple samples, it tries to merge them before propagating them upwards.

THEOREM 3. *Our quantile algorithm, when running on a partitioned tree, has $O(\sqrt{kh}/\epsilon)$ total communication and $O(\log(k/h)/\epsilon)$ maximum individual node communication.*

PROOF. Since we sample the data values with probability at most $\sqrt{t}/\epsilon n$, where $t = O(k/h)$, the total sample size is $O(\sqrt{k/h}/\epsilon)$ (in expectation). In the first phase of the algorithm, all the sampled data values are sent to the component roots, then broadcast to all nodes in the component, and finally aggregated back to the component roots to compute their local ranks. Thus each sampled data value could travel to all the $O(h)$ nodes in a component in the worst case. This results in $O(\sqrt{k/h}/\epsilon \cdot h) = O(\sqrt{kh}/\epsilon)$ communication in total.

In the second phase, the component roots send their initial samples to the base station. Even if we do not do any merging of the samples, the cost would be at most $O(\sqrt{k/h}/\epsilon \cdot h) = O(\sqrt{kh}/\epsilon)$, since each sample takes at most $h$ hops to reach the base station. Thus the total communication cost is $O(\sqrt{kh}/\epsilon)$.

In terms of maximum individual node communication, we still use the same algorithm in Section 3, except that a sample is said to be large if its ground set is of size at least $n/\sqrt{t}$. This results in $O(\log \sqrt{t}) = O(\log(k/h))$ classes. So the maximum individual node communication is $O(\log(k/h)/\epsilon)$ (expected) according to the analysis in Section 3. □

**Remarks.** The partitioning approach well fits the case where the sensor network already uses a clustered structure, as in LEACH [13] and COUGAR [25]. In this case a cluster naturally corresponds to a connected component. Note that we set the size of the component to $O(h)$ and the number of component to $O(k/h)$ just for optimizing the communication cost. The correctness of the algorithm and its probabilistic guarantees on the returned quantiles do not depend on these parameters. When the cluster sizes deviate from $O(h)$, the total communication cost might be affected slightly but not the quality of the computed quantile summary.

However, when the sensor network does not deploy a clustered structure, the algorithm on the partitioned tree might introduce additional overhead since it is no longer a one-round algorithm as in [12, 14, 21]: It needs 3 rounds of communication within each component plus one final round from the component roots to the base station.

# 5. EXPERIMENTS

In this section we evaluate our algorithm experimentally, comparing with the two previous algorithms: the q-digest [21] and the GK algorithm v2 [12] (See Table 1). We denote our Sampling Based algorithms as SB-1 (for the one-round version in Section 3) and SB-p (for the improved version based on tree partitioning in Section 4).

## 5.1 Experiment setup

We built a simulator which simulates a sensor network and implemented all four algorithms on top of the same platform. The network topology is generated in the same way as in [21], i.e., sensors are distributed over a certain area uniformly at random. Sensors are assumed to have a fixed radio range, and two sensors may communicate with each other if and only if they are within range of each other. The root of the network is chosen from the sensors randomly, after that a routing tree is generated by a breadth-first search

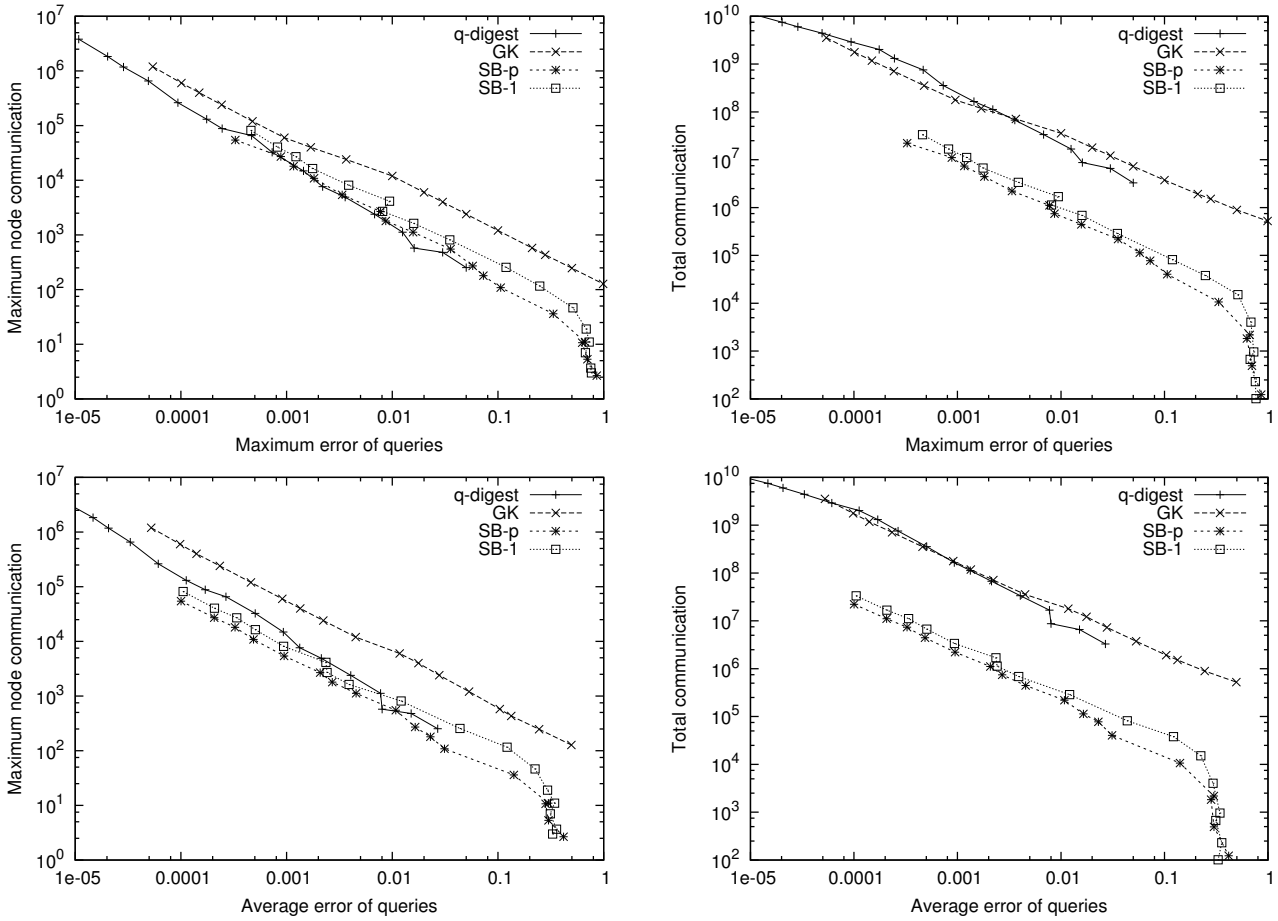


**Figure 3: Error-Communication trade-offs on the synthetic data set with $k = 16384$.**

starting from the root. Since our main goal is to improve the scalability of the algorithms in terms of network size, the experiments are done on relatively large networks, with $k = 1024$, 2048, 4096, 8192, 16384 nodes, respectively.

We used both synthetic and real data to test the performance of the algorithms. For the synthetic data sets, we first generated a total of $n = 1$ billion values from a Gaussian distribution with mean 0 and variance 1, and then scaled them to the range $[0, 2^{32} - 1]$ and round them to integers, since q-digest cannot handle floating-point numbers, though the other algorithms can. Next, we deployed the sensors over a unit square area, and randomly distributed these $n$ integers to the sensors. For the real data set, we used a terrain data for the Neuse River Basin, available at [3]. This data set contains roughly 0.5 billion LIDAR points which measure the elevation of the terrain. In this case we randomly deploy the sensors on the terrain, and assume that each sensor collects the elevation data nearby. We adjusted the radius of the area from which a sensor collects data such that the total size of data set, i.e., $n$, is around 1 billion. Note that this data set is highly correlated, since close sensors will observe similar elevations.

To perform a quantile computation, we set some $\epsilon$ and run all four algorithms. However, as mentioned earlier this may not be a fair comparison, since our algorithms give an $\epsilon$ error with a constant probability, while the previous algorithms guarantee a worst-case error of $\epsilon$. The actually observed errors for both q-digest and the GK algorithm could be much smaller than the required $\epsilon$. So, we measured the *actual* errors and plotted the error-communication trade-off, as $\epsilon$ is varied from 1 to 0.0001. Specifically, for each $\epsilon$ we measured both the actual maximum and the average error of 99 quantiles for $\phi = 1\%, 2\%, \ldots, 99\%$. Recall that the error in an estimated $\phi$-quantile is defined as $|\phi - \phi'|$, if the returned quantile is actually a $\phi'$-quantile of the data set.

We measured the communication cost in terms of the number of bytes transmitted. We measured both the total communication cost, as well as the maximum individual node communication cost. Thus combined with the maximum error or the average error, we in total have four combinations for the error-communication trade-off.

## 5.2 Error-communication trade-offs

The four error-communication trade-offs on the largest network $k = 16384$ with the synthetic data set are shown in Figure 3. Note that we use log scale on both axes. As we decrease $\epsilon$, naturally all algorithms get more accurate but more expensive. Our first observation from the experiments is that the actual errors for q-digest and GK are indeed smaller than the $\epsilon$ we set. Especially for q-digest, the average error of the 99 quantiles is only $\epsilon/20$; even the maximum error is only about $\epsilon/10$. This explains why the curves for q-digest are all shifted to the left, and shows that the worst-case error does not occur, at least on this data

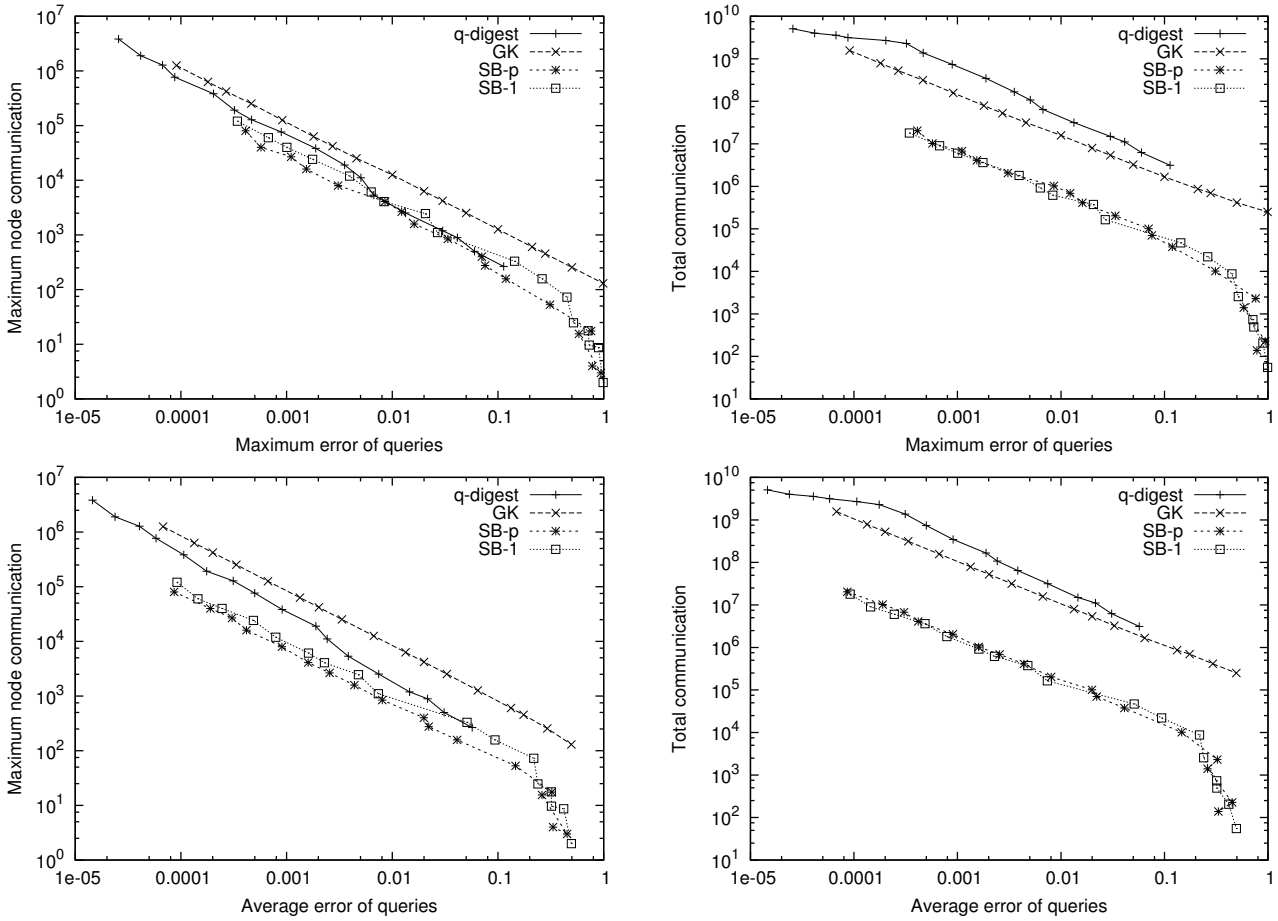


**Figure 4: Error-Communication trade-offs on the terrain data set with $k = 16384$.**

set. GK behaves roughly as required, with the average error at $\epsilon/2$ and the maximum error very close to $\epsilon$. On the other hand, the average error for our algorithms is roughly equal to $\epsilon$, exactly because we designed our algorithms to achieve a standard deviation of $\epsilon$; while the maximum error is roughly $2\epsilon$.

Nevertheless, although the actual errors of q-digest and GK are smaller than the set $\epsilon$, they are still 100 times larger than the actual errors of our algorithms, compared at the same total communication cost (see the two plots on the right in Figure 3). Alternatively speaking, to achieve the same actual (average or maximum) error, our algorithms are roughly 100 times more communication-efficient than both q-digest and GK. The gap is slightly smaller when considering the maximum error. On the other hand, the advantage of our algorithm in terms of the maximum node communication is much less impressive, and they are roughly 10 times better than GK, and almost the same as that of q-digest (the two plots on the left in Figure 3). Unfortunately, we believe that the maximum node communication cannot be further improved significantly, if possible at all, since the maximum communication always occurs at a node near the base station, which inevitably has to summarize a large amount of data.

Comparing q-digest and GK, the results suggest that q-digest is better in terms of maximum node communication while they behave almost the same way when considering the total communication. This is because q-digest almost always uses messages of a fixed size, while GK could still use messages smaller than the stated $O(\log n \log(h/\epsilon)/\epsilon)$ bound.

Another interesting observation from the results is that the error-communication curves of our algorithms take a sharp turn on the large $\epsilon$'s close to 1. This is because when $\epsilon$ is large, our algorithms essentially degenerate into simple random sampling, which has communication cost proportional to $1/\epsilon^2$. Recall that our algorithms' cost is actually $O(\min\{\sqrt{kh}/\epsilon, h/\epsilon^2\})$. Thus, on a log-log plot, it becomes two line segments with different slopes.

Finally, we do not see a significant difference between SB-1 and SB-p, with SB-p performing slightly better (within a factor of 2). Thus we would recommend the simpler SB-1 for sensor networks that do not have a clustered structure. If the network is naturally clustered, then SB-p offers better performance for free.

The experimental results on the terrain data set are shown in Figure 4, which do not show major differences from those on the random data set. The only observable difference that q-digest seems to perform slightly worse than it does on the random data set. On the terrain data set, q-digest is worse than GK in terms of total communication, but still better in terms of maximum node communication. On the other hand, there is no perceptible change in the behaviors of our algorithms. This is expected, as the analysis of our algorithms does not depend on the data characteristics.

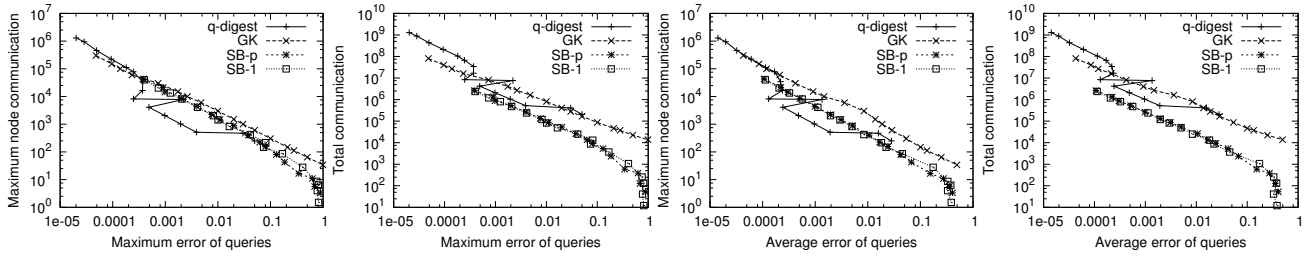


**Figure 5: Error-Communication trade-offs on the synthetic data set with $k = 1024$.**

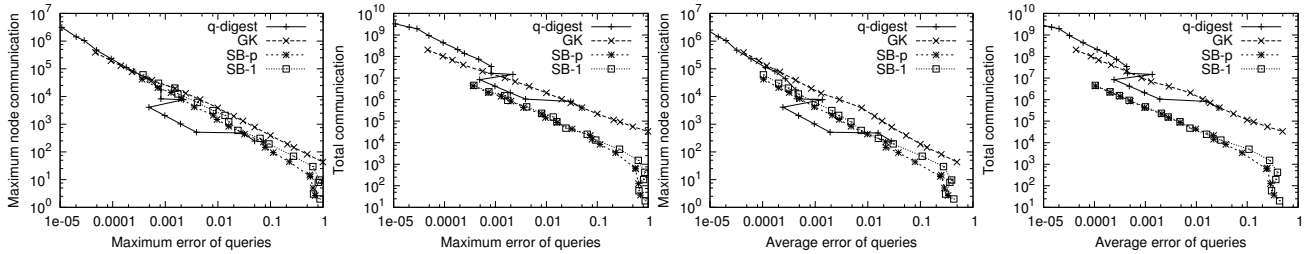


**Figure 6: Error-Communication trade-offs on the synthetic data set with $k = 2048$.**

## 5.3 Varying network size

The experimental results for the other network sizes (using synthetic data) are shown in Figure 5–8. The general conclusion is the same as before, that is, our algorithms perform much better than the previous two algorithms in terms of total communication costs. With our algorithms, the total communication cost (for achieving the same *actual* error) can be reduced to 1/10 – 1/100 of that of previous algorithms, as $k$ goes from 1024 to 16384. On the other hand, all algorithms have roughly the same maximum node communication. This is in line with our theoretical analysis.

From the experiments we also observe that q-digest in some cases exhibits a "zigzag" behavior: the actual error sometimes suddenly goes down even when the set $\epsilon$ gets larger. But this strange behavior starts to fade out as the network size gets larger. We do not have a good explanation of this phenomenon, but it could relate to our earlier observation that for q-digest, the actual error could be much smaller than $\epsilon$. It may happen so that in some cases, the actual error gets really small. On the other hand, both GK and our algorithms are quite stable across all the experiments we have done.

Finally, we observe that our algorithms also run much faster than the previous algorithms. This is not surprising, since all algorithms' running times are proportional (modulo polylog factors) to the total size of all the messages they handle. Thus a small communication cost also implies a faster running time.

# 6. CONCLUSION

In this paper we have designed new sampling based algorithms for quantile computation in sensor networks, improving the previous algorithms by one to two orders of magnitude in terms of the total communication cost. The key is to deviate from the standard decomposable framework, which has been followed by all previous data aggregation algorithms for sensor networks. Our result has demonstrated that, although the decomposable framework works well for simple aggregates, it may not be the best solution for complex summaries like quantiles. We believe our ideas may also lead to more communication-efficient algorithms for computing other complex data summaries in sensor networks, such as histograms and wavelets.

# 7. REFERENCES


[1] http://cast.cse.ohio-state.edu/exscal/.

[2] http://www.greenorbs.org/.

[3] http://www.ncfloodmaps.com.

[4] J. Considine, F. Li, G. Kollios, and J. Byers. Approximate aggregation techniques for sensor databases. In *Proc. IEEE International Conference on Data Engineering*, 2004.

[5] G. Cormode, M. Garofalakis, S. Muthukrishnan, and R. Rastogi. Holistic aggregates in a networked world: Distributed tracking of approximate quantiles. In *Proc. ACM SIGMOD International Conference on Management of Data*, 2005.

[6] G. Cormode and M. Hadjieleftheriou. Finding frequent items in data streams. *Proceedings of the VLDB Endowment*, 1(2):1530–1541, 2008.

[7] G. Cormode, F. Korn, S. Muthukrishnan, and D. Srivastava. Space- and time-efficient deterministic algorithms for biased quantiles over data streams. In *Proc. ACM Symposium on Principles of Database Systems*, 2006.

[8] G. Cormode and S. Muthukrishnan. An improved data stream summary: The count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.

[9] E. Fasolo, M. Rossi, J. Widmer, and M. Zorzi. In-network aggregation techniques for wireless sensor networks: a survey. *IEEE Wireless Communications*, 14(2):70–87, 2007.

[10] A. C. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. J. Strauss. How to summarize the universe: Dynamic maintenance of quantiles. In *Proc.*

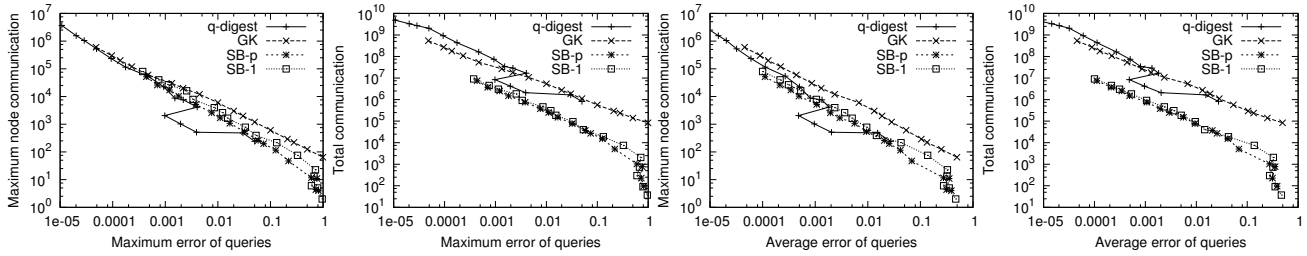


**Figure 7: Error-Communication trade-offs on the synthetic data set with $k = 4096$.**

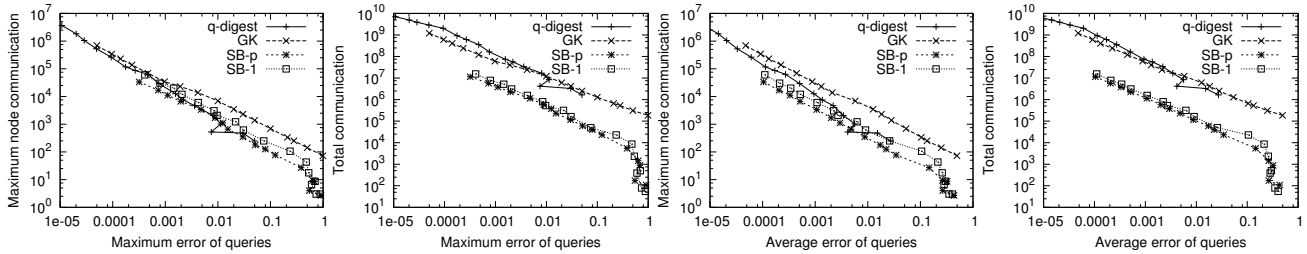


**Figure 8: Error-Communication trade-offs on the synthetic data set with $k = 8192$.**

*International Conference on Very Large Data Bases*, 2002.

[11] M. Greenwald and S. Khanna. Space-efficient online computation of quantile summaries. In *Proc. ACM SIGMOD International Conference on Management of Data*, pages 58–66, 2001.

[12] M. Greenwald and S. Khanna. Power conserving computation of order-statistics over sensor networks. In *Proc. ACM Symposium on Principles of Database Systems*, 2004.

[13] W. B. Heinzelman, A. P. Chandrakasan, and H. Balakrishnan. An application-specific protocol architecture for wireless microsensor networks. *IEEE Transactions on Wireless Communications*, 1(4):660–670, 2002.

[14] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. TAG: a tiny aggregation service for ad-hoc sensor networks. In *Proc. Symposium on Operating Systems Design and Implementation*, 2002.

[15] A. Mainwaring, D. Culler, J. Polastre, R. Szewczyk, and J. Anderson. Wireless sensor networks for habitat monitoring. In *ACM WSNA*, 2002.

[16] A. Manjhi, S. Nath, and P. B. Gibbons. Tributaries and deltas: efficient and robust aggregation in sensor network streams. In *Proc. ACM SIGMOD International Conference on Management of Data*, 2005.

[17] L. Mo, Y. He, Y. Liu, J. Zhao, S. Tang, X.-Y. Li, and G. Dai. Canopy closure estimates with GreenOrbs: Sustainable sensing in the forest. In *Proc. ACM SenSys*, 2009.

[18] J. I. Munro and M. S. Paterson. Selection and sorting with limited storage. *Theoretical Computer Science*, 12:315–323, 1980.

[19] S. Nath, P. B. Gibbons, S. Seshan, and Z. R. Anderson. Synopsis diffusion for robust aggregation in sensor networks. In *Proc. ACM SenSys*, 2004.

[20] J. Polastre, R. Szewczyk, and D. Culler. Telos: Enabling ultra-low power wireless research. In *ACM/IEEE IPSN*, 2005.

[21] N. Shrivastava, C. Buragohain, D. Agrawal, and S. Suri. Medians and beyond: New aggregation techniques for sensor networks. In *Proc. ACM SenSys*, 2004.

[22] A. Silberstein, K. Munagala, and J. Yang. Energy-efficient monitoring of extreme values in sensor networks. In *Proc. ACM SIGMOD International Conference on Management of Data*, 2006.

[23] V. N. Vapnik and A. Y. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and its Applications*, 16:264–280, 1971.

[24] J. Yang, K. Munagala, and A. Silberstein. Data aggregation in sensor networks. In *Encyclopedia of Database Systems*. 2009.

[25] Y. Yao and J. Gehrke. Query processing in sensor networks. In *Proc. Conference on Innovative Data Systems Research*, 2003.

[26] K. Yi and Q. Zhang. Optimal tracking of distributed heavy hitters and quantiles. In *Proc. ACM Symposium on Principles of Database Systems*, 2009.

[27] P. Zhang, C. M. Sadler, S. A. Lyon, and M. Martonosi. Hardware design experiences in ZebraNet. In *Proc. ACM SenSys*, 2004.

[28] Y. Zhang, X. Lin, J. Xu, F. Korn, and W. Wang. Space-efficient relative error order sketch over data streams. In *Proc. IEEE International Conference on Data Engineering*, 2006.