Streaming Techniques and Data Aggregation in Networks of Tiny Artifacts

Luca Becchetti[a], Ioannis Chatzigiannakis[b], Yiannis Giannakopoulos[c,*]

[a] *SAPIENZA University of Rome, Dipartimento di Informatica e Sistemistica, Via Ariosto 25, 00185 Rome, Italy*
[b] *Research Academic Computer Technology Institute (RACTI), 1 N. Kazantzaki Str., University of Patras Campus, Rion 26504, Greece*
[c] *University of Athens, Department of Informatics, Panepistimioupolis, 15784 Athens, Greece*

Abstract

In emerging pervasive scenarios, data is collected by sensing devices in streams that occur at several, distributed points of observation. The size of data typically far exceeds the storage and computational capabilities of the tiny devices that have to collect and process them. A general and challenging task is to allow (some of) the nodes of a pervasive network to collectively perform monitoring of a neighbourhood of interest by issuing continuous aggregate queries on the streams observed in its vicinity. This class of algorithms is fully decentralized and diffusive in nature: collecting all data at few central nodes of the network is unfeasible in networks of low capability devices or in the presence of massive data sets. Two main problems arise in this scenario: i) the intrinsic complexity of maintaining statistics over a data stream whose size greatly exceeds the capabilities of the device that performs the computation; ii) composing the partial outcomes computed at different points of observation into an accurate, global statistic over a neighbourhood of interest, which entails coping with several problems, last but not least the receipt of duplicate information along multiple paths of diffusion.

Streaming techniques have emerged as powerful tools to achieve the general goals described above, in the first place because they assume a computational model in which computational and storage resources are assumed to be far exceeded by the amount of data on which computation occurs. In this contribution, we review the main streaming techniques and provide a classification of the computational problems and the applications they effectively address, with an emphasis on decentralized scenarios, which are of particular interest in pervasive networks.

Keywords: data streams, aggregation, sensor networks, database management

1. Introduction

Technological advancements and socioeconomic forces have transformed the way in which we live, work and communicate with each other. In this new era, perhaps the most critical variable upon which all our developement is based, is that of efficiently managing the huge amount of information constantly being generated in various, diverse forms and locations. The predominant computational model in modern environments is distributed in nature: many remote devices, possibly different in hardware specifications, are continuously observing and generating huge amounts of data that far exceed their storing, processing and energy capabilities and are organized in dynamically evolving, pervasive network infrastructures that, as well, have limited bandwidth and serving capabilities with respect to the amount and the dissemination of the tasks we are asking them to perform.

In this survey we deal with the algorithmic issues underlying such settings, giving special emphasis on the assumption that our fundamental processing units are *tiny artifacts*, small and usually inexpensive devices with very limited storage, computational power, energy independence and, of course, reliability. In particular, we are interested in being able to efficiently extract crucial statistical information regarding our entire network, in the form of *aggregate queries*. We review important results from the areas of data streaming and database management, in section 2 describing the fundamental algorithmic techniques of traditional, centralized data streaming. Then, using these as building blocks, we cover distributed computational models in section 3.

In no way we consider this survey to be exhaustive. The area of data streaming is very wide and constantly evolving and we refer to other treatments [1–6] and excellent tutorials [7, 8] for further consideration.

1.1. Motivation

Our main motivating applications in this survey arise in the field of *sensor networks* [9]. The authors in [10–12] report the deployment of such networks in a wide range of scientific, security, industrial and business applications. Examples include climatological and environmental monitoring, traffic monitoring, smart homes, fire detection, seismic measurements, structural integrity, animal control and

*Corresponding author
Tel.:+30 2107275139, Fax: +30 2107275114
Email addresses: `becchett@dis.uniroma1.it` (Luca Becchetti), `ichatz@cti.gr` (Ioannis Chatzigiannakis), `ygiannak@di.uoa.gr` (Yiannis Giannakopoulos)

habitat monitoring. Apart from sensor networks, other motivating applications include IP routing and network traffic monitoring and analysis, managing large databases, secure and real-time financial transactions and of course, *the Web* itself [1, 2, 13, 3].

1.2. Aggregation

In the scenarios outlined above, single individual values are usually not of great relevance. In fact, users are more interested in the quick extraction of succinct and useful synopses about a large portion of the underlying observation set. Consider, for example, the case of a temperature sensor network. We would like to be able to continuously monitor entire infrastructure and efficiently answer queries such as "What was the average temperature over the entire terrain during the last 12 hours?", or "Are there any specific clusters that have reached dangerously high temperatures?".

As already mentioned and further discussed in following section 1.4, trying to collect all data monitored by the sensors would be unrealistic in terms of bandwidth, power consumption and communication intensity. So, the canonical approach is to compute statistical *aggregates*, such as max, min, average, quantiles, heavy hitters, etc., that can compactly summarise the distribution of the underlying data. Furthermore, since this information is to be extracted and combined across multiple locations and devices, repeatedly and in a dynamic way, *in-network aggregation* schemes [14] must be developed that efficiently merge and quickly update partial information to include new observations. Also notice that, computing aggregates instead of reporting exact observations, can leverage the effect of packet losses and, generally, network failures, which are common phenomena in wireless networks of tiny artifacts. We deal with such issues explicitly in section 3.3.

1.3. Traditional vs sensor network streaming

It is evident that there are two levels of computation and aggregation in distributed settings. At a low level, each sensor observes a stream of data and needs to efficiently extract and maintain information about it. This is essentially the problem of traditional, centralized streaming which has been extensively studied during the last two decades [15–17]. Aggregation is considered with respect to the individual values comprising the data stream, into a concise summary. This is the subject matter of section 2.

At a higher level, all remote sites should coordinate to combine these partial information computed from each device. Here, aggregation is considered with respect to this merging process of creating summaries that describe the entire infrastructure. Obviously, new challenges are imposed in such distributed settings, which we address in section 3. It should be clear that this in-network aggregation model generalizes traditional streaming, in the way that a single data stream can be seen as values distributed along a linear-chain topology [18, section 1.3]. Efficient algorithms for distributed computation, that do not make stringent assumptions about the infrastructure topology, can be readily used for classical streaming problems.

1.4. Physical restrictions and algorithmic challenges

In the setting of massive data stream computation addressed in this survey, data are observed or produced at a far higher rate than can be locally stored or, sometimes, even observed. Their delivery to aggregation or elaboration points requires an amount of in-network communication that far exceeds the power capabilities of sensing devices. Furthermore, the computational complexity of exactly evaluating the statistical aggregates of interest is unrealistic [6]. Considering further that we are interested in scenarios where these functions are performed by tiny devices with extremely limited resources, it is natural to ask for algorithmic solutions and data structures that require storage and update times that are sublinear and often (poly)logarithmic with respect to the size of the observed data, further imposing similar constraints on the amount of communication involving each device. We address these issues in more detail to sections 2.3 and 3.1.

2. Traditional Data Streams

2.1. Streaming models

In the remainder of this section, we refer to streaming models that best reflect the scenarios arising in networks of tiny artifacts. In particular, we refer to the taxonomy presented in [1]. The input is a stream $a_1, a_2, \ldots$ of items that arrive sequentially, one after the other, and that describe an underlying signal $\mathbf{A}$, which in turn can be regarded as a one-dimensional function $\mathbf{A} : [1 \cdots N] \to \Re$. Each item causes an update of $\mathbf{A}$'s state. We denote by A_t the state of the signal after seeing the t-th item in the stream. Models differ on how the a_i's describe $\mathbf{A}$.

Time Series Model. In this model, each a_i equals $\mathbf{A}[i]$ and the a_i's are presented in increasing order of i. This is a suitable model for data that are released in time series. For example, this is the case when we receive periodic updates about the temperature at a particular site, or we receive periodic updates about the amount of traffic observed at a network link. This is sometimes called "Insert Only Model" [19, Chapter 3].

Cash Register Model. In this case, the a_t's are increments to $\mathbf{A}[j]$'s. Following the notation of [1], we have $a_t = (j, I_t)$, $I_i \geq 0$, meaning that the t-th item causes the following update to the underlying signal: $\mathbf{A}_t[j] = \mathbf{A}_{t-1}[j] + I_t$, Note that in this case, multiple a_i's could increment a given $\mathbf{A}[j]$ over time. This model reflects many scenarios of practical interest, such as keeping track of the traffic volumes directed towards different IP destination addresses as packets traverse a router. It is a special case of the more general Turnstile model [1], in which updates can be negative (also called "Accumulative Model" in [19, Chap. 3]). We do not consider the more general Turnstile model in

this survey, since it is of lesser interest for the applications in networks of tiny artifacts we consider.
Further modelling issues. Some scenarios of interest pose constraints that are not taken into account by the models we have introduced (and not even by the Turnstile model). For example, assume that, considered the generic item $a_t = (i, I_t)$, i uniquely describes an event (e.g., specifying geographical coordinates and a time-stamp). Assume further that the same item a_t may be reported several times (e.g., because events are delivered to a computing sink node from different sensor nodes connected in a network in which multiple data paths from nodes to sink are present) but $\mathbf{A}$ has to be updated only once per event.

Another issue occurs when we want to compute an aggregate over multiple data streams. Think, for example, of computing the average temperature over the streams of readings performed by a number of sensor nodes covering an area. One natural intuition is that this is equivalent to performing the computation over a suitably defined aggregate stream. The problem is that the correct way to derive the aggregate stream is problem dependent. In particular, the arrival order of the items observed at different streams might affect the value of the aggregate we want to estimate. In the remainder of this survey we restrict to cases of interest in practice, in which arrival order is irrelevant, i.e., the aggregate is *order-insensitive.* In this case, considered any time t, $\mathbf{A}_t$ only depends on the items that were observed until time t and not on the order of observation.

Windowed streaming. It is natural to imagine that the recent past in a data stream is more significant than distant past. A typical case is the following: A sensor continuously monitors a temperature, maintaining its maximum over the past hour. It should be clear that with limited memory, it is not possible for the sensor to know at every time step the maximum temperature observed in the past hour. There are currently two main approaches to model this aspect. The first is combinatorial and assumes a sliding window of size W, so that at time t, the aggregate we are interested in should be computed only over the last W updates, i.e, $a_{t-W+1}, \dots, a_t$. The impact of items outside the current window should be discounted as the window slides over time. The difficulty of course is that we cannot store the entire window, but only $o(W)$, or typically only $o(polylog(W))$ bits are allowed. This pretty natural model was proposed in [20]. Its main drawback is that the correct window size should actually depend on the current rate of observation and thus vary over time. One way to address this issue is based on the use of an Exponential Weighted Moving Average to describe the falling importance of items as they age.

2.2. Statistical aggregates

We first consider the case in which we have a stream of data and a single point of observation. With respect to this case, we define a number of statistics of interest for a network of tiny artifacts. We emphasize that the collection we propose is far from exhaustive. The interested reader can refer to [1, 19] for more comprehensive overviews.

Statistics of interest in networks of tiny artifacts. The reference scenario of this subsection is the one in which we want to issue continuous queries over the values of a stream observed at a single point of observation, e.g., a sensor node. we will restrict to queries that are realistic and of interest in networks of artifacts with limited computing, communication and storage capacities. In particular, we shall consider sum, mean, frequency moments [16, 15, 21], quantiles, frequent items, heavy hitters and histograms [22, 1]. These aggregates are defined below.
Sum, mean and max. These queries are of interest both in the time series and the cash register models. In the basic case of a single stream, the sum can be obviously maintained in $O(1)$ space with $O(1)$ cost per update both in the time series and cash register models using a single counter. Maintaining mean is also straightforward in the time series model: at time t the value of the mean is exactly $\sum_{i=1}^{t} \mathbf{A}[i]/t = \sum_{i=1}^{t} I_i/t$. Maintaining the mean in the cash register model in polylogarithmic space poses some challenge. In this case, the (exact) value of the mean is given by $\sum_i \mathbf{A}[i]/|\{j : \mathbf{A}[j] \neq 0\}|$. This requires maintaining, at any time t the number of *non-zero* components in $\mathbf{A}$ at time t. While this is straightforward with $\Omega(N)$ space, it is far less obvious if we are constrained to sublinear space. This problem is considered in Subsection 2.4.
Frequency moments. For $k \geq 0$, the k-th frequency moment is defined as $F_k = \sum_i \mathbf{A}[i]^k$. The values of k of interest in applications are $k = 0, 1, 2$. For $k = 0$ we are tracking the number of distinct items observed so far. For $k = 1$ we are keeping track of the sum of updates. F_2 is important as a measure of variance in data streams and it can be interpreted as the Self-Join size.
Quantiles. The response to the (ϕ, ϵ)-quantiles query is the set $\{j_k\}$ of indices, with $k = 0, \dots, 1/\phi$,[1] such that $(k\phi - \epsilon)\|\mathbf{A}\|_1 \leq \sum_{i \leq j_k} \mathbf{A}[i] \leq (k\phi + \epsilon)\|\mathbf{A}\|_1$ for every k.
Top-k items and heavy hitters. Problems of interest in monitoring applications include the following: i) maintain the top-k (i.e., the k largest items) of $\mathbf{A}$; ii) maintain the ϕ-heavy hitters, i.e., the set $\mathcal{I}$ of indices, such that $\mathbf{A}[i] \geq \phi\|\mathbf{A}\|_1$, for every $i \in \mathcal{I}$. In the time series model, the former problem can be easily solved using k counters, whereas the latter can be solved using $O(1/\phi)$ counters, after observing that there cannot be more than $1/\phi$ heavy hitters. In the cash register model, information theoretic arguments show that even the problem of maintaining $\max_i \mathbf{A}[i]$ requires $\Omega(N)$ space [15] and the same holds for the ϕ-heavy hitter problem [1, Subsection 5.1.2, Thm. 9]. In the remainder, we consider the (ϕ, ϵ)-heavy hitter problem, defined as follows: Return all i's such that

[1] For the sake of simplicity, we assume that $1/\phi$ is an integer. If this is not the case, the last value of k is $\lfloor 1/\phi \rfloor$.

$\mathbf{A}[i] \geq \phi \|A\|_1$ and no i such that $\mathbf{A}[i] \leq (\phi - \epsilon)\|A\|_1$, for some specified $\epsilon < \phi$.

Histograms. A histogram [23] is a partition of $[0, N)$ into intervals $[b_0, b_1) \cup \cdots [b_B - 1, b_B)$, where $b_0 = 0$ and $b_B = N$, called buckets, together with a collection $\{h_j\}_{j=0}^{B-1}$ of B heights, one for each interval. For every $S \subseteq \{0, N\}$, denote by χ_S the N-dimensional index vector that is 1 on components corresponding to elements in S. The histogram is an approximation to $\mathbf{A}$ given by $\mathbf{R} = \sum_{j=0}^{B-1} h_j \chi_{[b_j, b_{j+1})}$. This means that the value of each component $\mathbf{A}[i]$ is approximated by h_j, where j is the index of the unique bucket containing i. The problem is choosing $B - 1$ (boundary, height) pairs (b_j, h_j), such that $\|\mathbf{A} - \mathbf{R}\|^2$ is minimized. Let $\mathbf{R}_{opt}$ the optimal choice for $\mathbf{R}$. The problem cannot be solved exactly in a streaming model [1, 23], hence the (B, ϵ)-histogram problem is naturally defined as finding a vector $\mathbf{R}$ as defined above, such that with probability at least $1 - \delta$, $\|\mathbf{A} - \mathbf{R}\|^2 \leq (1 + \epsilon)\|\mathbf{A} - \mathbf{R}_{opt}\|^2$.

Estimating statistics in a connected world. Some emerging application, such as sensor network based monitoring and network-wide IP traffic analysis present many technical challenges. They need distributed monitoring and continuous tracking of events. In these new scenarios, the focus is more on aggregate queries, computed over readings collected at different points rather than at a single point of observation. This is for example the case in large wireless sensor networks where, as observed in [24], aggregation queries often have greater importance than individual sensor readings. A similar scenario is likely to emerge in networks of tiny artifacts, where we are likely to perform continuous monitoring functions and their individual readings have to be aggregated and/or collected at one or more points of elaboration and analysis.

This general scenario can be abstracted assuming the presence of k streams $\mathcal{S}_1, \ldots, \mathcal{S}_k$ at multiple points of observation, whose readings collectively affect a common underlying signal $\mathbf{A}$, as defined in Subsection 2.1. The i-th item $a_i(h)$ observed in the h-th stream is thus a pair (j, I) that causes the update $\mathbf{A}[j]$ by I. If we consider order-insensitive statistics, we can naturally view each $\mathcal{S}_h$ as a set and define the *union* stream $\mathcal{S}$ as $\mathcal{S} = \cup_{h=1}^{k} \mathcal{S}_h$. Given this general framework, many issues arise, depending on the communication model in the underlying network.

In particular, the underlying communication infrastructure affects the way in which queries are diffused in the network and results are aggregated. Two general communication paradigms have to be considered, since they significantly affect the nature of the problems we want to solve. The first assumes the presence of a routing infrastructure, so that queries and data packets are routed towards their destinations along single paths. This is for example the case in an IP network or a in a sensor network in which readings are delivered to the sink along the paths of a tree rooted at the sink itself [25]. The second communication paradigm is flooding-based and is for example of interest in networks of tiny artifacts connected over a wireless network. Here, the presence of multiple data paths from a source to a destination can cause the presence of duplicates, i.e., the same item or piece of data might be received multiple times. This, in turn, can affect accuracy, depending on the characteristics of the aggregate of interest and the technique used to estimate it.

Queries. In this survey we restrict to *order-insensitive* queries, i.e., aggregates whose result does not depend on the order in which items are observed. Such queries form the vast majority in practice. Queries can be further be *one-shot* or *continuous* [26]. In the former case, we want to retrieve answers to the query on demand. In the latter case, we want to track the value of an aggregate continuously, as the stream evolves. This is for example the case when we want to implement a distributed system of triggers to detect the presence of anomalies. Another distinction is between algebraic and holistic queries [27]: for the former, the aggregate can be expressed by an algebraic function of a suitable number of arguments (e.g., max or sum), whereas this does not hold for holistic queries (e.g., top-k or quantiles). A further distinction is between duplicate-sensitive and duplicate-resilient queries [28]: the former compute aggregates whose value may depend on the presence of duplicates, whereas this does not hold for the latter. For example, assume k sensor nodes report events to a sink. Assume each event is a pair (i, a), with i an identifier of the event (e.g., geographical coordinates and time-stamp) and a a value associated to the event (e.g., a temperature value). It is clear that, in estimating the aggregate, the contribution of each item should be considered only once. What happens if the same item can be reported multiple times to the sink (e.g., due to the presence of multiple data paths)? This depends on the aggregate we wish to compute. Thus, if we want to compute the maximum value ever observed then it is irrelevant whether or not the same event is reported twice or more times (possibly aggregated with other data), since max is duplicate-resilient. Conversely, multiple reports of the same event might negatively affect the accuracy for duplicate-sensitive aggregates, such as sum or mean.

2.3. Measuring performance

In general terms, the performance of a streaming algorithm is measured along two main axes: i) the amount of resources needed to perform the computation; ii) the accuracy in the estimation of the statistics of interest.

Space and time resources. This aspect has been described in general terms in Section 1.4. Here we make this notion more precise. There are different performance measures: i) Processing time per item a_t in the stream; ii) Space used to store the data structure summarizing the state of A_t at time t (storage); iii) Time needed to compute functions on A (query time).

Typically, the objective is having a data structure of size $o(\max\{N, t\})$ at any time t, specifically $O(polylog(N, t))$. We would also like to obey similar constraints for the update and query times [29]. Two comments are in order. First, in general we want to have no dependence from t for obvious reasons. Second, it should be noted that N describes the size of the universe, i.e., the maximum possible number of $\mathbf{A}$'s components that might differ from 0. To stick to an example, assume we want to keep track of the set of concurrent IP flows traversing a router. In this case, N is obviously in the order of 2^{64}, i.e., the number of potential IP (source, destination) pairs. This is the reason why enforcing a polylogarithmic constraint in N is perfectly reasonable (see [1, Section 4.1] for a thorough discussion).

Accuracy of the estimation. The general purpose of a streaming algorithm is maintaining a summary over a data stream (or a set of data streams in general) that allows to answer queries about statistical aggregates defined over it. Assume that F and $\hat{F}$ respectively denote the exact and estimated value of a statistical aggregate of interest at any time. We say that $\hat{F}$ is ϵ-approximate if $(1-\epsilon)F \leq \hat{F} \leq (1+\epsilon)F$. We say that $\hat{F}$ is (ϵ, δ)-approximate if $(1-\epsilon)F \leq \hat{F} \leq (1+\epsilon)F$ with probability at least $1-\delta$ [1].

While the above-mentioned measures of accuracy provide a direct measure of our ability to track the evolution of an aggregate of interest over a stream, accuracy so defined may not be suitable to assess the quality of an algorithm. In some cases, a poor (worst case or expected) accuracy does not derive from poor algorithmic choices, but rather it reflects an intrinsic hardness of the problem at hand in the streaming model. We show in Section 2.5 how competitive analysis techniques [30] can prove effective to analyze the performance of streaming algorithms when the focus is on the algorithm's achievable performance rather than its absolute accuracy.

2.4. Fundamental streaming methods and algorithms

In this section, we briefly introduce some of the main algorithmic techniques used to address the problems described above. We only provide details of a few key results that are in our opinion representative of the area, while we only briefly discuss others, pointing the reader to relevant papers or surveys.

Sampling. Sampling has found wide application in streaming techniques. Sampling in a streaming scenario means every input/update is seen but only a (polylogarithmic sized) subset of items are retained, possibly with associated data such as the count of times the item has been seen so far. The item sample can be chosen deterministically or in a randomized way. In general, the hope is that the sample is a statistically significant representative of the whole stream and that aggregates computed exactly on it can provide good approximations of their counterparts computed over the whole stream. In the remainder, we consider sampling in its most basic version: given a collection of n items, we want to collect a sample of size $m < n$ with uniform probability. In a streaming context, a further complication arises, due to the fact that the length n of the stream is not known a priori. The main techniques used to circumvent this problem are *reservoir sampling* and *minwise hashing.*

In its most basic form, reservoir sampling works as follows: we put the first m items in a reservoir. From this point onwards, the i-th items has probability m/i to be sampled. If sampled, the i-th items replaces an item from the reservoir uniformly at random. For $m = 1$, it is easy to see that the generic i-th item is chosen with probability $1/n$. This follows since i) i is sampled with probability $1/i$; provided it is still in the reservoir when the j-th item is observed (of course, $j > i$) it remains in the reservoir with probability $1 - 1/j$. Hence the overall probability that i is the sample returned at the end of the stream is $\frac{1}{i}\prod_{j=1}^{n-1}\frac{j}{j+1} = 1/n$. It is easy to extend this argument to the case $m > 1$. This basic sampling alhorithm can be made faster, as shown in [31].

Min-wise hashing is another effective way to uniformly sample from a stream. Assume we have a set of of items from a finite discrete set A, which we assume without loss of generality to be $\{0, \ldots, n-1\}$, for a suitable n. [2] Assume we have a family $\mathcal{H}$ of hash functions, each producing a permutation of $\{0, \ldots n-1\}$. Assume $\mathcal{H}$ is such that, if h is chosen uniformly at random from it then, for every $A \subseteq \{0, \ldots n-1\}$ we have:

$$\mathbf{P}[\min\{h(A)\} = h(x)] = \frac{1}{|A|}, \forall x \in A.$$

This means that every item in A has an equal chance to achieve the minimum if H is chosen uniformly at random from $\mathcal{H}$.

Assume we observe a stream of items from a finite discrete set, as defined above. Assume we keep the item achieving the minimum value with respect to a hash function chosen uniformly at random from a minwise family. It is obvious that this achieves a uniform sample, since each item has an equal chance to achieve the minimum. Unfortunately, minwise independent hash function families require $\Omega(n \log n)$ truly random bits [32]. In practice, it is possible to use approximately minwise independent hash functions that require a logarithmic number of truly random bits [33].

Compact summaries or sketches. Many problems are not easily solved using sampling techniques. Such is for example the problem of counting the number of *distinct* items observed in a data stream, i.e., $|\{i : \mathbf{A}[i] > 0\}|$. If a large fraction of items aren't sampled, we cannot know if they are all same or all different for example. A complementary

[2]For example, if we were interested in the set of IP addresses, n would be 2^{32}.

approach relies on the use of compact data structures, or *synopses* or *sketches.* A sketch is intended to record, possibly in an approximate way, important trends in the observed stream, so as to at least approximately track the evolution of one or more aggregates of interest. Using a sketch, it may be hard or impossible to track the single components of the signal $\mathbf{A}$, but it might be possible to track the evolution of one or more statistics of interest. An important class is that of the sketches that are *composable* and *duplicate insensitive* [24, 34, 35]. In the remainder, we restrict to the cash register (and hence the time series) model, but the definitions we provide next hold for more general models. Consider an order insensitive statistics of interest and assume we want to estimate it over the union of two streams $\mathbf{A}$ and $\mathbf{B}$, i.e., we want to estimate the aggregate over $\mathbf{A} + \mathbf{B}$. A sketching algorithm for its estimation is composable if $\mathbf{Sk}(\mathbf{A} + \mathbf{B}) = \mathit{merge}(\mathbf{A}, \mathbf{B})$, where $\mathit{merge}(\cdot)$ is a suitable sketch aggregation function that depends on the sketching algorithm and the aggregate of interest [34]. Now, consider a stream in which each item is a pair (i, a), with i a unique event identifier (e.g., geographical coordinates and time-stamp) and a a value associated to the event (e.g., a temperature value)[3]. Assume the same event can be reported more than once (e.g., due to the presence of multiple data-paths). A sketching algorithm is duplicate insensitive if, at any point in time, the sketch computed by the algorithm is the same as if each event were observed only once. Examples of composable and duplicate-resilient sketches are the FM sketches discussed a few paragraphs below.

Sum, mean and max. Sum is trivial in both the time series and cash register models. As remarked above, mean is not obvious in the cash register model, since it requires to keep track of the number of non-zero components components of $\mathbf{A}$. Keeping track of the max is obvious in the time series model, but it requires $\Omega(N)$ space in the cash register model [15].

Frequency moments. In this paragraph, we mainly focus on F_2 and briefly discuss results for F_k when $k \geq 3$. F_1 is trivial, since it is simply the sum of all updates. F_0 is the count of $\mathbf{A}$'s non-zero components. While it is trivial in the time series model, computing it in the cash register model is an important problem that has received considerable attention dating back to the 80's. For this reason, it is treated in a separate paragraph. The problem of maintaining the second frequency moment over $\mathbf{A}$ is a special case of the problem of succinctly maintaining the 2-norm of a set of vectors in $\Re^n$. This problem has been successfully addresses in the past and we point the reader to the contribution of Achlioptas [36], which also provides an overview of previous work on the subject. The special case of maintaining the second frequency moment has been elegantly addressed in [15] and its proof is presented below. Before presenting the result, we briefly provide an intuition. Assume we consider the scalar product $\mathbf{A} \cdot \mathbf{R}$, where $\mathbf{R}$ is random vector whose entries are in $\{-1, 1\}$ with probability $1/2$. If $\mathbf{R}$'s entries are pairwise independent, then it is easy to prove that $\mathbf{E}\left[(\mathbf{A} \cdot \mathbf{R})^2\right] = \sum_{i=1}^{n} \mathbf{A}[i]^2$. Intuitively, this tells us that $(\mathbf{A} \cdot \mathbf{R})^2$ is an unbiased estimator of F_2. To obtain an (ϵ, δ)-approximation of F_2 we need to expand this basic idea and show how $\mathbf{R}$ can be maintained implicitly using (poly)logarithmic space and truly random bits.

Theorem 1 ([15], Theorem 2.2)**.** *Considered any stream $\mathcal{S}$ in the time-series or the cash register model[4], it is possible to maintain an (ϵ, δ)-approximation of F_2 over $\mathbf{A}$ using $O\left(\frac{\log(1/\delta)}{\epsilon^2}\right)$ counters and $O\left(\frac{\log(1/\delta)}{\epsilon^2} \log n\right)$ truly random bits.*

Proof. Set $s_1 = \frac{16}{\epsilon^2}$ and $s_2 = 2 \log \frac{1}{\delta}$. We implicitly maintain s_2 independent $N \times s_1$ matrices $\mathbf{R}^1, \ldots, \mathbf{R}^{s_2}$, where $\mathbf{R}^r_{ij} \in \{-1, 1\}$ with equal probability and where, for every *fixed* j and for every r, the $\mathbf{R}^r_{ij}$'s are 4-wise independent. One way to achieve this is for instance shown in the proof of Theorem 2.2 in [15].

Maintaining a sketch of $\mathbf{A}$*.* Our sketch of $\mathbf{A}$ consists of $s_1 s_2$ counters C^r_j, where $C^r_j = \mathbf{A}\mathbf{R}^r_{*j}$, for $r = 1, \ldots, s_2$. As a result, we have $s_1 s_2 = \frac{32}{\epsilon^2} \log \frac{1}{\delta}$ counters. As to the number of random bits, for every matrix $\mathbf{R}^r$, we need $O(\log N)$ bits to generate the generic column $\mathbf{R}^r_{*j}$ so that the $\mathbf{R}^r_{ij}$'s are 4-wise independent, for a total of $s_1 s_2 O(\log N) = \left(\frac{1}{\epsilon^2} \log \frac{1}{\delta} \log N\right)$ truly random bits. Assume at time t item $a_t = (i, I_t)$. We update our sketch as follows: for every $j = 1, \ldots, s_1$ and $r = 1, \ldots, s_2$, we let $C^r_j = C^r_j + I_t \mathbf{R}^r_{ij}$, where $\mathbf{R}^r_{ij}$ is generated *on demand.* Notice that the sketch thus obtained is obviously the same we would obtain if we updated $\mathbf{A}$ until time t *and then* computed its sketch.

Estimating F_2*.* To estimate F_2 at any point in time we proceed as follows. Let

$$Y_r = \frac{\sum_{j=1}^{s_1} (C^r_j)^2}{s_1}.$$

Our estimation of F_2 is $\mathit{median}\{Y_1, \ldots, Y_r\}$.

Analysis. For any $j = 1, \ldots, s_1$ and $r = 1, \ldots, s_2$, set $X = (C^r_j)^2$. We have:

$$\mathbf{E}[X] = \mathbf{E}\left[(\sum_{i=1}^{N} \mathbf{A}[i] \mathbf{R}^r_{ij})^2 \right] = \sum_{i=1}^{N} \mathbf{A}[i]^2,$$

where the inequality immediately follows from 4-wise (and thus pairwise) independence of the $\mathbf{R}^r_{ij}$. On the other hand, 4-wise independence implies that $\mathbf{E}\left[X^2\right] = \sum_{i=1}^{N} \mathbf{A}[i]^4 + 6 \sum_{1 \leq i < h \leq N} \mathbf{A}[i]^2 \mathbf{A}[h]^2$. Hence, simple manipulations show

[3]Note that we are thus assuming that i uniquely identifies the event, i.e., (i, a) and (j, b) with $i = j$ implies $a = b$.

[4]The result actually holds in the more general, turnstile model [1, Section 4.1].

that $\mathbf{Var}(X) \leq 4\sum_{1\leq i<h\leq N} \mathbf{A}[i]^2\mathbf{A}[h]^2 \leq 2F_2^2$. Now, for every r, the C_j^r's (and thus the $(C_j^r)^2$'s) are statistically independent, since they depend on the $\mathbf{R}_{*j}^r$'s, which are generated independently. Hence, $\mathbf{Var}(Y_r) \leq 2F_2^2/s_1$ and, from Chebyshev's inequality:

$$\mathbf{P}[|Y_r - F_2| > \epsilon F_2|] \leq \frac{2F_2^2}{s_1\epsilon^2 F_2^2} = \frac{1}{8},$$

by our choice of s_1. Now, given our choice of s_2 and recalling that our estimate of F_2 is $median\{Y_1, \dots, Y_r\}$, we conclude the proof by observing that a standard application of Chernoff's bound allows to conclude that the probability that more than $s/2$ of the Y_r's deviate from F_2 by more than ϵF_2 is at most $1/\delta$. □

The approach just shown is particularly suited for F_2 and easily extends to maintaining pairwise distances in subspaces of $\Re^N$. Unfortunately, it does not extend to F_0 and F_k, $k > 2$. In particular, maintaining F_k requires $N^{1-2/k}$ for any real $k > 2$ [15, 37].

Distinct counting. Consider a stream of integers, each belonging to the discrete interval $\{1, \dots, N\}$. The distinct counting problem asks to return the number of *distinct* values that were observed. It is trivial to note that this is exactly the problem of computing F_0 in the cash register model, i.e., $\|\{i : \mathbf{A}[i] \neq 0\}\|$, provided that, considered the generic item $a_t = (i, I_t)$, $I_t > 0$ always holds. For this reason, in the remainder of this paragraph we refer to the basic problem defined above and the t-th item a_t in the stream is assumed to be an integer value. The basic tool we consider is a *counting sketch*, i.e., a composable and duplicate insensitive counter of the number of distinct items appearing in a stream. We consider two approaches: the first is the approach proposed in the seminal paper of Flajolet and Martin [16], considered in [38] and modified in [15]. The second is a slightly different technique proposed in [21].

FM sketches. In the following, we use the phrase "FM sketch" to refer to any implementation of the original counting sketch of [16]. FM sketches [16] use a simple approach in which each sketch is a vector of m entries, each entry being a bitmap of length $k = O(\log N)$. Considered the s-th bitmap of the sketch. Every observed value is hashed onto the bitmap bits using a (independently chosen) hash function $h_s(\cdot) : \{1, \dots, N\} \to \{0, \dots, \log_2 N - 1\}$, such that the probability of hashing onto the h-th bit is 2^{-h}. The bit under consideration is set to 1 if it was 0. After processing the stream, let r_s denote the position of the least significant bit that is still 0 in the s-th bitmap: it is easy to see that r_s is a good estimator for $\log_2 F_0$, the logarithm of the number of distinct pairs observed. To improve accuracy, we consider $\frac{1}{m}\sum_{s=1}^m r_s$ as an estimator of $\log_2 F_0$, where $m = O\left(\frac{1}{\epsilon^2}\log\frac{1}{\delta}\right)$. This variant on the basic FM sketch algorithm is called *Probabilistic Counting with Stochastic Averaging (PCSA).*

LogLog Counting. In [39] a variant of the *PCSA algorithm* is presented that reduces the size of the accumulation synopsis from $\log N$ to $\log\log N$. However, the standard error is increased from $0.78/\sqrt{k}$ to $1.30/\sqrt{k}$, where k is the number of bitmaps. Which means that the LogLog Counting is less accurate but the space complexity is improved considerably. The algorithm differs from the PCSA in two aspects. The first is that the *maximum bit set to 1* is maintained, in contrast to the PSCA where the position of the *least significant 0-bit* is maintained, and the second is the function used to compute the estimate.

Bar-Yossef et al. [21]. The approach of [21] relies on the following intuition. Consider the elementary algorithm that first picks a random hash function $h : \{1, \dots N\} \to [0, 1]$ and then applies $h(\cdot)$ to all the items in the stream, maintaining the value $v = \min_{t=1}^n h(a_t)$, with n the stream size. Then $1/v$ is in expectation the right approximation, since if there are F_0 independent and uniform values in $[0, 1]$ (the images of the stream's items), then their expected minimum is around $1/F_0$.

In order to achieve (ϵ, δ)-accuracy, this basic idea is extended in [21], with a sketching algorithm that maps every observed value to an integer using a pairwise independent hash function $h(\cdot)$ and at any point in time maintains the list of the L smallest distinct values observed so far, where $L = \lceil 96/\epsilon^2 \rceil$. If v is the L-th smallest distinct value maintained by the algorithm, LM/v is an estimator of F_0, where $M = N^3$. Precision can be increased by the standard trick of considering m independent and parallel copies of the algorithm and taking the *median* of the corresponding estimations, where $m = O\left(\frac{1}{\epsilon^2}\log\frac{1}{\delta}\right)$.

Composability and duplicate-resilience. Both sketches are composable and duplicate insensitive: Consider two streams $\mathcal{S}_1$ and $\mathcal{S}_1$ of values from the same universe $\{1, \dots, N\}$ and let $\mathbf{Sk}^{FM}(S_1)$ and $\mathbf{Sk}^{FM}(S_1)$ be their FM sketches. Then it is straightforward to see that be $\mathbf{Sk}(S_1)$ `OR` $\mathbf{Sk}(S_2)$ is the sketch corresponding to $S_1 \cup S_2$. As for the sketches of [21], every such sketch is an array of m lists, each maintained according to the algorithm described above. Merging of two such sketches is simply achieved as follows: for every $s = 1, \dots, m$, merge the s-th lists of the two sketches and keep the L smallest values of the merged list. Both approaches achieve similar bounds in terms of efficiency and precision, as stated by the following

Theorem 2 ([38, 16, 21])**.** *Considered a stream S of integers, it is possible to maintain an estimate $\hat{F}_0$ of the number F_0 of distinct items in S using $O\left(\frac{1}{\epsilon^2}\log\frac{1}{\delta}\right)$ memory words, such that:*

$$\mathbf{P}\left[|\hat{F}_0 - F_0| > \epsilon F_0\right] \leq \delta.$$

Quantiles. The quantile problem is trivial to solve in the time series model. On the other hand, $\Omega(N)$ space required to compute quantiles exactly in the cash register model [17]. The folklore method of sampling values randomly and returning the sample median works in $O\left((1/\epsilon^2)\log(1/\delta)\right)$

space to return the median, i.e., the $(1/2, \epsilon)$-quantile, to ϵ-approximation with probability at least $1-\delta$ [1]. Better strategies allow to use $O\left((1/\epsilon)(\log^2(1/\epsilon) + log^2 \log(1/\delta))\right)$ space and are based on randomized sampling [40]. The (ϕ, ϵ)-quantile problem can also be addressed deterministically in a streaming setting. In this case, the currently best known algorithm uses $O\left((\log^2(\epsilon\|\mathbf{A}\|1))/\epsilon\right)$ space [18], where this strategy is applied to the case of answering quantile queries in a sensor network.

Top-k items and heavy hitters. Negative results for the top-k and heavy hitter problems have been discussed in Subsection 2.2. Therefore, in this paragraph we restrict to effective algorithmic strategies for the (ϕ, ϵ)-heavy hitter problem. Research about top-k and heavy hitter problems has been extremely active in the recent past. For the sake of space, in this paragraph we discuss some key contributions, referring the reader to [1] for a thorough overview.

Deterministic strategies: the Space-Saving algorithm. When each update increases the value of one component of $\mathbf{A}$ by exactly 1 unit, the space-saving algorithm proposed in [41] solves the (ϕ, ϵ) heavy hitter ptoblem in $O(1/\epsilon)$ space. The idea is to maintain, at any point of the execution, m counters, each recording (possibly in an approximate way) the current count of an element (i.e., a component of $\mathbf{A}$ in the cash register model). The value of m depends on the required accuracy, in particular $m = O(1/\epsilon)$. The counters are updated in a way that accurately estimates the frequencies of the significant elements, and a lightweight data structure is used to keep elements sorted by their estimated frequencies. Ideally, the i-th most frequent element e_i should be accommodated in the i-th counter C_i. Assume f_i is the true frequency of such element. In general, $C_i \neq f_i$, due to errors in estimating the frequencies of the elements. For the same reason, the order of the elements in the data structure might not reflect their exact ranks, so that C_i might contain some element $\hat{e}_i$ different from e_i. Denote by min the value of the counter associated to the element with lowest estimated frequency kept in the summary, i.e., e_m.

Given these definitions, the algorithm is straightforward. If a monitored element is observed, the corresponding counter is incremented. If the observed element e is not among monitored ones, it is given the benefit of doubt and it replaces the element with lowest estimated hit number, i.e., $e_m = e$. Counter C_m is incremented to $C_m + 1$, since the new element e could have actually occurred between 1 and $C_m + 1$ times. For each monitored element e_i, the algorithm keeps track of its maximum overestimation, ϵ_i, resulting from the initialization of its counter when it was inserted into the list. That is, when starting to monitor e by counter C_m, its maximum overestimation error ϵ_m is set to the counter value that was evicted. To implement this algorithm, the authors propose a simple *stream summary* data structure that cheaply increments counters without violating their order, and that ensures constant time retrievals/updates.

Using randomization: the Count-Min sketch. We describe the CM-Min sketch proposed in [22], since this a versatile data structure whose application extends beyond the heavy hitter problem to rangesum queries, scalar product, quantiles and moments. The Count-Min sketch data structure (CM-sketch for short) is based on a 2-dimensional array whose size is determined by design parameters ϵ and δ (their meaning explained further). Every element of the array is a counter $C[j, l]$, where $j = 1, \ldots, d$ and $l = 1, \ldots, w$, with $d = \lceil \ln \frac{1}{\delta} \rceil$ (depth) and $w = \lceil \frac{e}{\epsilon} \rceil$ (width). Every entry is initially 0. The data structure also consists of d hash functions $h_1, \ldots, h_d$ chosen uniformly at random from a pairwise-independent family, so that $h_r : \{1, \ldots, N\} \to \{1, \ldots, w\}$.

The update procedure of the CM-sketch is straightforward: upon observing item $a_t = (i, I_t)$ at time t, for every $j = 1, \ldots, d$, set $C[j, h_j(i)] = C[j, h_j(i)] + I_t$. Ideally, this operation corresponds to the (exact) update operation on $\mathbf{A}$: $\mathbf{A}[i] = \mathbf{A}[i] + I_t$. When using the CM-sketch, our estimation of $\mathbf{A}[i]$ (point-query) is $\min_{j=1\ldots d} C[j, h_j(i)]$.

CM sketch and heavy hitters. First note that it is possible to maintain the current value of $\|\mathbf{A}\|_1$ at any time t, since $\|\mathbf{A}\|_1 = \sum_{i=1}^{t} I_i$. The CM-sketch based algorithm for heavy hitters in the cash register model makes use of a min-heap and it works as follows. Upon receiving item (i, I_t), it updates the sketch as before and it estimates $\mathbf{A}[i]$ to check whether it is above the threshold $\phi\|\mathbf{A}|\|_1$, in which case $\mathbf{A}[i]$'s estimate is added to a heap. The heap is kept small by checking that the current estimated count for the item with lowest count is above threshold; if not, it is deleted from the heap. At any point in time, all items in the heap whose estimated count is above $\phi\|\mathbf{A}|\|_1$ are candidate heavy hitters. It is possible to prove the following result:

Theorem 3 ([22]). *The heavy hitters can be maintained in the cash register streaming model by using CM sketches with space* $O((1/\epsilon)\log(\|\mathbf{A}\|_1/\delta))$*, and time* $O(\log(\|\mathbf{A}\|_1/\delta))$ *per item. Every item which occurs with count more than* $\phi\|\mathbf{A}|\|_1$ *time is output, and with probability at least* $1-\delta$*, no item whose count is less than* $(\phi - \epsilon)\|\mathbf{A}\|_1$ *is output.*

Histograms. There is a simple dynamic programming solution [1, 23] when there are no space constraints. This algorithm computes the optimal B-bucket histogram in time $O(N^2 B)$ and space $O(BN)$. This result has been improved to $O(N^2 B)$ time and $O(N)$ space in [42]. In [23], the authors propose an algorithm for approximate histogram maintenance that requires time and space $poly(B, 1/\epsilon, \log\|\mathbf{A}\|, \log N)$.

Computing aggregates in the window streaming model. We briefly discuss some key contributions for the sliding-window model. A general consideration is that even simple primitives such as counting or summing cannot be performed exactly in this model, unless one uses an amount of memory that is linear in N, i.e., the support of $\mathbf{A}$.

One key contribution to this line of research is the paper [20] (see also [43, Chapter 8]). Here, the authors first address the following elementary Basic Counting problem: given a stream of data elements, consisting of 0's and 1's, maintain at every time instant the count of the number of 1's among the last W elements. It is possible to prove [20] that maintaining the exact count requires $\Theta(N)$ space. The authors show a general technique addressing this problem and related ones, which relies on the use of Exponential Histograms. They are thus able to prove that basic counting can be performed with accuracy ϵ using $O((1/\epsilon)\log^2 \epsilon W)$ space, with updates that are performed in $O(1)$ amortized and $O(\log W)$ worst-case time. This result is asymptotically tight and it is further extended to the Sum problem: Given a stream of data elements that are positive integers in some range $[0\ldots, R]$, maintain at every time instant the sum of the last W elements. Again, ϵ accuracy can be obtained using polylogarithmic space [20]. Further results are shown for a general class of *weakly additive* functions, including L_p norm for $p \in [1,2]$. Improved results for the aggregates considered in [20] (in particular weakly additive functions) and results for L_p, when $p \notin [1,2]$ and geometric mean are presented in [44]. Other basic tools, such as reservoir sampling, have been extended to the windowed model [45].

Time Decaying Sketches. A different, more complex approach is presented in [46] introducing a new sketch that maintains duplicate insensitivity, asynchronous arrivals and time decay of the processed data simultaneously. The algorithm is capable of generating a lot more than distinct-values estimates, such as *selectivity*, *frequent items* and *decayed-sum* estimates. Considering the fact that it supports *time-decay* through user defined *decay-functions*, such as a sliding window function, it is a very powerful approach to sensor data aggregation.

Every item is a tuple $i = (u_i, w_i, t_i, id_i)$, where id_i is a unique integer value identifying the specified item, u_i is an observed value and t_i a timestamp. Item i is further assigned a weight w_i that, combined with a user-defined decay function $f(\kappa - t)$, reflects the change in the significance or confidence in the value u_i over time, where κ is the query time and t the timestamp of the item. The overall weight of an item is defined as: $w_i \cdot f(\kappa - t)$.

For every *distinct* item a range of integers, r_i^κ is defined such that all ranges are disjoint, i.e. $r_i^\kappa = [w_{max} \cdot id_i, w_{max} \cdot id + w_i \cdot f(\kappa - t) - 1]$. The size of the range is exactly $w \cdot f(\kappa - t)$. Then, $M+1$, initially empty random samples $S_0, S_1, \ldots, S_M$ are defined, where M is of the order of $\log_2(w_{max} \cdot id_{max})$. At first, each integer in r_i^κ is assigned in sample S_0 and then for $j = 0 \ldots M-1$ each integer in S_j is placed in S_{j+1} with probability approximately 1/2. Hence the probability that an integer is assigned in S_j is $p_j \approx 1/2^j$.

The assignment of integers to samples is done through the time-efficient $Range - Sampling$ technique [47] that quickly samples the whole range r_i^κ in time $O(\log|r_i^\kappa|)$, using a pairwise independent hash function. The function call $Range - Sample(r_i^\kappa, l)$ actually returns the number of integers that belong to sample S_l. When at least one integer of a r_i^κ range is assigned to a sample then the item i is actually stored in that sample.

Because the query time κ, and hence the decayed weight of an item, $w_i \cdot f(\kappa - t)$, is unknown at the time the item arrives in the stream, an "expiry time" is defined for item i at level j such that as long as $\kappa < expiry(i,j)$ the range r_i^κ has at least one integer assigned to S_j, and for $\kappa \geq expiry(i,j)$, r_i^κ has no integers assigned to S_j. This way, with the arrival of an item in the stream the item is tagged with its expiry time and is assigned to S_j as long as the current time is less than $expiry(i,j)$.

For smaller values of j, S_j the size of the samples may be too large and hence take too much space. As a result the algorithm stores only at most τ items in each sample with the largest expiry times. The size τ of the samples is a critical factor of the algorithm and depends on the desired accuracy (the bigger the sample the more accurate the estimate will be). Moreover, since the guarantees of the algorithm are of the form: *"With probability at least $1-\delta$, the estimate is an ϵ-approximation to the desired aggregate"*, the maximum size of a sample depends on the approximation parameter ϵ.

An essential decayed aggregate to the computation of distinct-value estimates is the decayed sum of all distinct items in the stream, i.e. $V = \sum_{(u,w,t,id)\in D} w \cdot f(c-t)$, where D is the set of distinct items. We recall that *Range−Sample*(r_i^κ, l) returns exactly the number of integer in r_i^κ that belong to S_l, i.e. the size of the range. As a result, to compute the decayed sum it suffices to compute: $\frac{1}{p_l}\sum_{i\in S_l} RangeSample(r_i^\kappa, l)$. To estimate the number of distinct-value items, we only have to set the weight, w_i of each item to 1 and compute the decayed sum over a sliding window.

2.5. Competitive analysis of streaming algorithms

We briefly discuss a novel approach that was recently proposed to measure the performance of a streaming algorithm, based on the use of competitive analysis [30]. Literature on streaming algorithms has mostly considered the degree of accuracy of algorithms and not their performance with respect to the best that can be achieved. In order to assess the quality of the algorithm, it may be useful to adopt the competitive point of view and judge the accuracy of a streaming algorithm against the accuracy of an offline algorithm with the same resource limitations.

This line of research was initiated in [48], where the authors consider the following problem: we consider a streaming model in which the algorithm observes a sequence $\{a_1, a_2, \ldots\}$ of items over time, a_t being the item observed during the t-th *time step*. In the sequel, we assume without loss of generality that the value field of a_j belongs to the (integer) interval $[1, M]$ for some $M > 1$. In fact, this is an example of the time series, where $a_j =$

(j, I_j). For the sake of simplicity, in the rest of this subsection we denote by a_j *both the j-th item and its value.*

We further assume that, at every time t, we are only interested in maintaining statistics over the window of items observed in the interval $\{t - W + 1, \ldots, t\}$. In the following, g_t denotes the item of maximum value maintained by the algorithm at time t and m_t denotes the offline optimum's value in the window at time t. The required memory space is measured in units, each unit being the amount of memory necessary to exactly store an item. The offline algorithm has the same memory restrictions as the online algorithm, but it knows the future. In particular, both algorithms are allowed to maintain, at any time t, at most k items among those observed in $\{t - W + 1, \ldots, t\}$ (where, typically, $k << W$).

While approximating the maximum value over a sliding window can be done using polylogarithmic space [44], the basic task of maintaining the maximum value exactly is not feasible, unless one stores a number of elements in the order of W and this result also holds for randomized algorithms [20]. In [48] the authors consider the following objective functions, that measure how far the streaming algorithm is from achieving this goal, both at every point in time and in the average over the entire sequence.

Aggregate max: Maximize $\sum_t g_t$, i.e., the average value of the largest item maintained by the algorithm.

Anytime max: For every t, maximize g_t, i.e., maximize the value of the largest item in the algorithm's memory at time t. As shown further, this function is harder to maintain for every t.

Competitive analysis. The performance of the streaming algorithm is compared against the optimal algorithm that knows the entire sequence in advance [30]. Hence, in this the competitive ratio $\mathbf{r}(k, W)$ is defined as:

$$\mathbf{r}(k, W) = \max_{a \in \mathcal{S}} \frac{\sum_t g_t(a)}{\sum_t m_t(a)},$$

where $\mathcal{S}$ is the set of possible input sequences and $g_t(a)$ (respectively, $m_t(a)$) is the online algorithm's (respectively the offline algorithm's) maximum value at time t when input sequence a is observed.

Aggregate max. In [48] the authors prove an $1 + \Omega(1/k)$-competitive randomized lower bound for this problem. At the same time, they prove that a simple, bucket-based heuristic is asymptotically achieves competitive ratio $(k-1)/k$. The heuristic works as follows: the stream sequence is partitioned into parts of size W/k. Part s starts at time $(s-1)W/k + 1$ and ends at time sW/k. For every part s of the sequence, a particular slot of memory is active, the memory slot $i = 1 + (s \mod k)$. For each part of the sequence, the active slot of the memory accepts the first item. In every other respect, the active slot in each part is updated greedily: the algorithm updates the slot value whenever an item of larger (or the same) value appears. Clearly, the partition-greedy algorithm can be implemented very efficiently (using two counters to keep track of the current active slot of memory and the number of items seen in each part). Interestingly enough, there is an asymmetry between max and min for this problem. In particular, the aggregate min problem turns out to have unbounded competitive ratio.

Anytime max. In the aggregate max problem, the objective is to optimize the *sum* of the maximum value whereas here it is to optimize the *maximum.* That is, we are interested in minimizing

$$\max_t \frac{g_t}{m_t}.$$

for a worst case stream of values. It is possible to show that the competitive ratio in this case cannot be independent of the values. More precisely, the competitive ratio is $O(\sqrt[k+1]{M})$, where M is the maximum value in the stream. This is achieved by a simple bucket-based algorithm. It is also relatively easy to show that this result is tight.

2.6. Experimental comparison of duplicate insensitive counting algorithms

In this section we discuss the effectiveness of streaming algorithms in current commercial off-the-shelf hardware that are used in mobile/sensor environments based on the experimental evaluation conducted in [49]. In particular, three streaming algorithms PCSA ([16], Sec. 2.4), LogLog-Count ([39], Sec. 2.4) and TDS ([46], Sec. 2.4) were evaluated in a testbed of resource-limited hardware devices. The focus of the evaluation was on the space complexity, time complexity and absolute error of the algorithms in real-world conditions, in order to determine if they have acceptable efficiency when applied in highly restrained environments. The available platforms are listed in Table 1.

A core aspect in the implementation of [16, 39] is the selection of the hash function. In fact, Flajolet and Martin analyze the error guarantees of their algorithm assuming the use of an explicit family of hash functions with ideal random properties (more precisely, that the hash function maps each value uniformly at random to an integer in the specified range). Given that fact, the best selection of hash functions to be used for the experiments is MD4, which both approximately conforms to the assumption of Flajolet and Martin and is widely used efficiently in a variety of applications. The parameters in these algorithms are the number of bits per bitmap vector, *bbits*, and the number of bitmaps, k. We set *bbits* $= 32$ bits/bitmap and $k = 2^8 (= 256)$ bitmaps. Thus the standard error for the LogLog Counting algorithm is $\approx 8.1\%$ and for the PCSA algorithm is $\approx 4.8\%$. This is a decent selection regarding the hardware we have at our disposal, as the memory space requirements are too small for these two algorithms to consider it a restraint.

A key component of the algorithm of [46] is the computation of the function *expiry(i, j)*, which includes the use of the *RangeSample(r_i^κ, j)*. In the formal description

Table 1: Time measurement (sec) for 6000 items, duplicate rate=20%

Processor	**RAM**	**TDS**	**PCSA**	**LogLogCount**
266MHz Intel XScale (Netbridge)	32MB	23.468149	0.127454	0.129718
400MHz ARM920T (Freerunner)	128MB	6.310182	0.106261	0.106734
500MHz AMD Geode LX800 (Alix)	256MB	1.282864	0.042732	0.043274

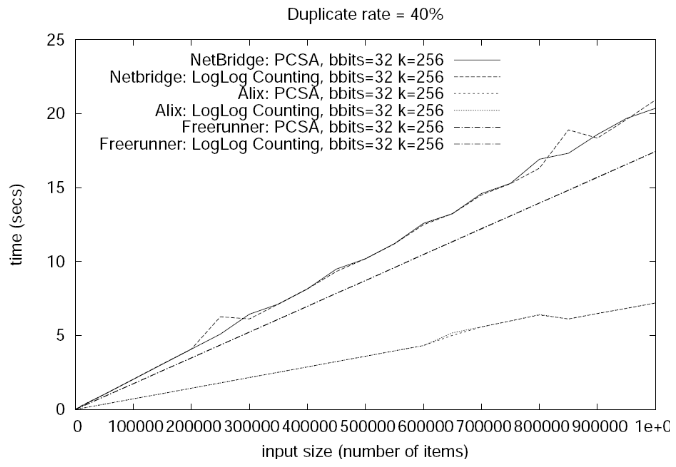


Figure 1: PCSA and LogLog Counting over the different platforms

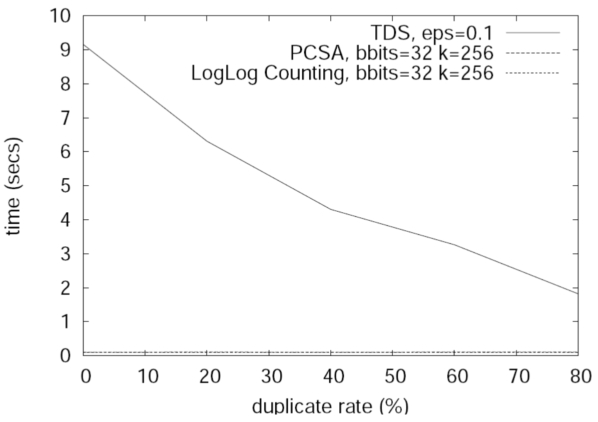


Figure 2: Duplicate rate performed on Freerunner

of the *Range Sampling* algorithm[47], the recursive procedure *Hits* achieves time complexity of separately sampling each of the integers is of the order of $\log_2(|r_i^{\kappa}|)$. The approximation parameter, ϵ is set to 0.1, so as each sample size, τ, is equal to $\tau = 60/\epsilon^2$ (which is proposed by the authors of [46] for the computation of the decayed-sum). The selection of ϵ to 0.1 is done because any smaller selection increases the space complexity and hence the time complexity of the algorithm to borderline bounds. The selection of the approximation parameter result to the maximum number of the distinct elements to be counted, i.e. the size of a sample($\tau = 60/\epsilon^2$). Hence, the maximum number of elements to the correct counting, for $\epsilon = 0.1$, is 6000 elements.

In Fig. 1, we observe that both PCSA and LogLog Counting algorithms are efficient enough when applied in a such resource-restrained environment, even when considering input sizes of the order of hundreds of thousands or even millions.

When inserting duplicates in the input dataset, the results in time measurement and standard error evaluation are the same as before for the two algorithms PCSA and LogLog Counting. Whereas, the sketch-based algorithm performs much faster than without duplicates. This is observed because the most time-consuming part of the process of the input dataset is the computation of the $expiry(i, j)$ of the item for the levels it belongs to, as well as managing the overflow (in size) of a sample and the sorting of its items' expiries for that purpose. The first is just a check whether the item has already been processed or not, and if not then continue with the next input. In Fig. 2 the outcome of the experiments performed in the *open moko neo freerunner* for input size = 6000 items.

3. Aggregation in Networks of Tiny Artifacts

3.1. Introduction

3.1.1. Distributed vs centralized streaming models

In section 2 we presented aggregation techniques for *centralized* streaming models, where a *single* processor observes a huge, rapidly updating data stream in a single pass, and needs to efficiently compute important statistical aggregates over the distribution of the data.

In this section, we will use these techniques as building blocks to study much more general, *distributed* streaming models, where many, independent streams are continuously generated from remote sites over a wide network infrastructure, and the statistical queries now refer to the union of these streams. The prevailing motivation behind such distributed models is that of sensor networks, where devices with limited storage and processing capabilities and extremely restricting energy independence are continuously monitoring physical parameters, such as temperature, and a base-station must be able to quickly query the entire network for aggregates such as average values, or critical events that exceed certain thresholds. Usually, such networks are wireless and battery-powered, and the cost of communication between the sensors, with respect to battery life, is orders of magnitude greater than the cost of local processing within each sensor.

There are two levels of computation in this distributed setting. First, each sensor must locally process its own stream, respecting its physical space and time complexity limitations. This can be done by deploying the techniques of section 2. At a higher level, all this information needs to be aggregated properly throughout the network structure (and usually routed towards a base station) in order to answer some query, a task requiring further in-network computation and coordination. Notice that the trivial solution of forwarding all data to a base station and performing there a local computation is not a realistic option, since the magnitude of the data far exceeds the communication (and storage) capacity of the network.

This makes clear the need of applying clever *in-network aggregation* techniques that distribute the computational burden among the nodes of the network, minimizing the communication cost and extending our network's energy lifespan. Appropriate and efficient *data summaries* must be maintained, forwarded and updated throughout the nodes of our network, capable of answering the statistical queries we are interested in, as quickly and with the highest precision possible.

3.1.2. The model

We have a set of *nodes* (sites, sensors) $I = \{1, 2, \dots, |I|\}$, indexed by i, each of which is observing a data stream $\mathcal{S}_i$. In the current section we want to focus on the decentralized aspects of network aggregation, and so, to keep things clear, we are not going to consider formal streaming models at the lower, sensor level like we did in section 2. Instead, for our exposition it is enough to think of the individual streams, simply as being multisets drawn from a (finite) universe U, $|U| = N$, i.e. $\mathcal{S}_i = \{a_{i\,1}, a_{i\,2}, \dots, a_{i\,m_i}\}$, $a_{i\,j} \in U$. Notice, though, that all algorithms we are going to present can be applied to the cash register, some of them even to the more general turnstile model. We set $n = \max\left\{|I|, \sum_{i\in I} m_i\right\}$. In general, we want to answer aggregate queries $Q\left(\cup_i \mathcal{S}_i\right)$ over the union of our data streams., i.e. the entire observation set.

Efficiency. We will consider our algorithms being efficient if each node transmits, at the worst case, at most a polylogarithmic number of messages (values), i.e. has a communication load of $O(polylog(n))$.

Network topology. The predominant assumption in sensor network computation is that our devices form a *tree*, at the root of which is a special device called the *base-station* which is responsible for answering the aggregate queries. This is done in roughly two phases: first, the base station distributes the query to all the nodes and then the needed information is pulled back to the root, in most cases using a fair amount of sophisticated in-network aggregation by the intermediate nodes along the way, which is essentially the subject matter of our exposition in this section 3. Every node merges (aggregates) information about its own stream with information received from its children and transmits this partial information about the distribution of the entire underlying data set to its parent.

In fact, no explicit physical tree structure of the communication links is required; only a well defined tree-routing protocol that is connected (the information transmitted from each node can reach the base station) and cycle-free (information from the same node is not received twice). This can be either predefined since the deployment of the network, or computed on-the-fly during the distribution phase by some spanning-tree procedure like breadth-first-search, especially in the case of dynamic, wireless networks. We deal with tree-based aggregation techniques in section 3.2.

Although such single-path, tree-based aggregation models capture the essence of sensor network communications and are straightforward to analyze, they come with a huge disadvantage: in case of a packet loss or a node failure, an entire subtree is disconnected and the aggregating procedure is rendered useless, especially when this node is close to the base station. In general, sensor networks are far from reliable with respect to such events, since usually many inexpensive devices are cast over a wide unattended or even hostile terrain. In addition, wireless networks are prone to environmental interferences, signal strength fading, packet collisions, etc [50, 5].

We can deal with lossy networks by using multi-path routing aggregation where each node may forward its information to many neighbors, reaching the base station via more than one path. This is a generic idea and can be approached in many ways, for example using directed diffusion or other gossiping techniques. We discuss this important issue of robustness in unreliable networks in section 3.3.

3.1.3. Queries, summaries and approximations

Queries are answered by maintaining auxiliary data structures called *summaries*[5] which are clever and succinct synopses of the underlying data set observed so far. More formally, following the influential exposition of Madden et al. [14], for an aggregate query Q to be able to be computed in-network, a summary $\mathbf{Sk}$ must be maintained, supporting three fundamental operations:

- Initialization (I): Used by each node i to create an instance $\mathbf{Sk}_{\mathcal{S}_i}$[6] of the summary to describe its own stream, i.e. $I(i) = \mathbf{Sk}_{\mathcal{S}_i}$.
- Aggregation (F): Used in intermediate steps of the aggregation procedure to merge two instances $\mathbf{Sk}_{M_1}$, $\mathbf{Sk}_{M_2}$ describing underlying multisets M_1 and M_2, respectively, into a single summary instance $\mathbf{Sk}_{M_1\cup M_2}$ describing their union, i.e.

$$\mathbf{Sk}_{M_1\cup M_2} = F(\mathbf{Sk}_{M_1}, \mathbf{Sk}_{M_2}). \qquad (1)$$

[5]The terms *sketch* and *synopsis* are also widely used.

[6]Sometimes we will use subscript notation $\mathbf{Sk}_M$ if we want to give emphasis on the underlying multiset M our summary (instance) describes.

- Evaluation (E): Used at the final step of the aggregation procedure (e.g. at the base-station) to extract the answer to query Q from an instance $\mathbf{Sk}_{\cup_i \mathcal{S}_i}$ describing our entire data set, i.e. $Q(\cup_i \mathcal{S}_i) = E(\mathbf{Sk}_{\cup_i \mathcal{S}_i})$.

For example, the AVG aggregate for the mean value can be computed by maintaining a summary consisting of only two numerical values $\langle x, y\rangle$, where $I(\mathcal{S}_i) = \langle \sum_{j=1}^{m_i} a_{i\,j}, m_i\rangle$, $F(\langle x_1, y_1\rangle, \langle x_2, y_2\rangle) = \langle x_1 + x_2, y_1 + y_2\rangle$ and $E(\langle x, y\rangle) = x/y$.

Not every aggregate query can be computed exactly by such a simple summary. As we saw in section 2, some queries are inherently much more difficult to compute and novel summaries must be utilized to achieve efficient approximations. These limitations trivially extend to the more general model of distributed streaming. For example, to compute exactly a complex query such as the MEDIAN would require pushing all $\Omega(n)$ data values, in the worst case, through the whole network and up to the root [17, 14]. What is the key property that characterizes this dichotomy?

Algebraic vs holistic queries. [51, 14]. A query will be called *algebraic* if it can be computed *exactly* by a summary of size $O(1)$ (with respect to the size n of the entire observation multiset). Examples of algebraic queries include: MIN, MAX, SUM, COUNT and AVG. This is not the case for more complex queries for which the summaries for exact computation must grow linearly with every intermediate application of the aggregation operation (1), essentially implementing the brute-force, centralized approach of forwarding the entire data set to a base station. Such queries are called *holistic* and include MEDIAN and, more generally, ϕ-quantiles, top-k/heavy hitters and DCOUNT (counting the number of distinct elements). Non-holistic aggregates are also known as *decomposable*, due to the form of the computational rule (1).

Fortunately, we can overcome these linear space requirements that would result to some sensors having an unacceptable load of transmitting $\Omega(n)$ values, by trying to devise summaries with $O(polylog(n))$ sizes that are able to answer our queries *approximately*, within sufficient accuracy. For this, we are going to use valuable knowledge obtained from the fundamental streaming algorithms of section 2. A rule of thumb is that summaries using $O(poly(\varepsilon^{-1}, \log n, \log N))$ storage, can approximate answers within a factor of ε [19].

Duplicate sensitivity. An aggregate query Q is called *duplicate-insensitive* if it is not affected by the insertion of multiple occurrences of the same element in the underlying data set. Formally, if for every multiset M, $Q(M) = Q(\overline{M})$, where $\overline{M}$ is the simple set induced by M (if we delete multiple occurrences). Duplicate-insensitive queries include MIN, MAX and DCOUNT, while notable duplicate-sensitive queries are COUNT, SUM and the ϕ-quantiles. This notion naturally extends to summaries, where a summary will be called duplicate-insensitive if for every multisets M, $M' \subseteq M$, its aggregation function F satisfies the property

$$F(\mathbf{Sk}_M, \mathbf{Sk}_{M'}) = \mathbf{Sk}_M, \tag{2}$$

for all possible instances of the summaries describing M and M'.

Duplicate sensitivity plays an important role when we design multi-path aggregation schemes (see section 3.1.2). In such settings, it is possible for the very same data point in the stream of some sensor, to be aggregated many times along different paths from the sensor to the base station. Such events obviously affect duplicate-sensitive queries and so we must try to deploy duplicate-insensitive summaries to answer them correctly.

3.2. Tree-based aggregation

As we discussed in section 3.1.2, when we perform in-network aggregation using a tree routing protocol, each observation in our data set is aggregated through a well-defined, single path from its sensor to the root of the tree. Simple, algebraic queries can be answered exactly, using natural summaries in the spirit of our example for AVG in section 3.1.3. This is not the case for holistic queries. In section 3.2.1 we present two very important summaries for sensor networks computing, namely q-digest and GK summaries, that efficiently give approximate answers to quantile queries and in 3.2.2 we discuss how we can readily apply Flajolet-Martin sketches to compute distinct items queries, since this is going to play a critical role in the development of duplicate-insensitive summaries for multi-path aggregation of section 3.3.

3.2.1. Quantiles

q-digests [52]. Consider a perfect binary tree of height $\log N$. To each node v we assign a *bucket* $[v^{\min}, v^{\max}]$ such that the root's bucket is our entire range of possible values $[1, N] = U$ and each node's bucket is divided equally among its children, i.e. root's children have ranges $[1, N/2]$, $[N/2, N]$ and, at the lowest level, all leaves have buckets of length 1. A *q-digest* $\mathbf{QD}_M$ over a multiset M is a subset of the nodes of the above binary tree. To each node v of $\mathbf{QD}_M$, along with his bucket $[v^{\min}, v^{\max}]$ we assign a counter c_v. Furthermore, each element $x \in M$ is assigned to *only one*[7] node-bucket v with $x \in [v^{\min}, v^{\max}]$. Counter c_v equals the number of M's elements assigned to v. Obviously, $\sum_{v \in \mathbf{QD}_M} c_v = |M|$. We will maintain the following two key properties for every q-digest. For all nodes v in $\mathbf{QD}_M$,

$$c_v \leq \lceil |M| / k \rceil, \tag{3}$$

if v is not a leaf and

$$c_v + c_{v_p} + c_{v_s} > \lceil |M| / k \rceil, \tag{4}$$

[7] This is the crucial point which differentiates q-digests with traditional histograms where every bucket counts *all* items from M in its range.

if v is not the root and v_p and v_s are v's parent and sibling respectively. k is a user defined integer, called *compression parameter*, to be determined later on. Intuitively, condition (3) tries to keep the bucket counts balanced throughout the summary and will also guaranty the desired formal error bounds for q-digests. Condition (4) prevents having two consecutive generations of nodes with small counts. As we will see, we achieve this by merging two siblings' buckets to the parent's count. This is essentially the main idea behind the compression achieved by q-digests.

Initialization. Each sensor i creates a q-digest $\mathbf{QD}_{\mathcal{S}_i}$ for its stream in the following way. It maintains a *conceptual* copy of the perfect binary tree (at the end, only the needed nodes will be instantiated). Every leaf of the tree corresponds to a single element in U and, by observing their frequencies in $\mathcal{S}_i$, we set accordingly the counters of the leaves' unit length buckets. Then, by traversing the tree in a bottom-up and left-to-right way, whenever a node v violates property (4), we set $c_{v_p} \leftarrow c_v + c_{v_s}$ and deactivate nodes' v and v_s counters. We call this operation *compression*. Notice that, because of the bottom-up way of our compression scheme, property (3) will never be violated. We continue this process, until no further compressions can be done. The nodes left with active counters constitute $\mathbf{QD}_{\mathcal{S}_i}$.

Aggregation. Suppose we have two q-digests $\mathbf{QD}_{M_1}$, $\mathbf{QD}_{M_2}$ using the same compression parameter k. Then we aggregate them to a single q-digest $\mathbf{QD}_{M_1 \cup M_2}$ in the most simple way: We merge the node-buckets of the two q-digests by taking the union of their nodes $\mathbf{QD}_{M_1} \cup \mathbf{QD}_{M_2}$, making sure that if a node appears in both summaries, we set its count equal to the sum of the two individual counts. Finally, we compress the resulting tree.

Evaluation. When a summary $\mathbf{QD}_M$ is questioned for a ϕ-quantile of the multiset M, we do a *post-order* traversal of its nodes, summing the node counts along the way. As soon as this sum exceeds $\lceil \phi |M| \rceil$, we output $v^{\max}$ as our answer, where v is the node currently visited.

Using property (3) one can prove [52] that for every node v in a q-digest $\mathbf{QD}_M$ with compression parameter k, the (additive) error of its count c_v (with respect to the actual frequencies of the distribution of M) is at most $\frac{\log N}{k} |M|$ and, most importantly, the aggregation operation respects this errors. From property (4) we can show that $|\mathbf{QD}_M| \le 3k$ and so, by choosing a compression parameter of $k = \frac{\log N}{\varepsilon}$ one can answer (ϕ, ε)-quantiles using a q-digest of size $\frac{3}{\varepsilon} \log N$. It is an immediate consequence that, by utilizing the above in-network aggregation technique, we can answer ε-approximate quantiles for our entire distributed stream $\bigcup_i \mathcal{S}_i$, with each sensor transmitting only a q-digest of size $O(\frac{1}{\varepsilon} \log N)$.

GK summaries [18, 53]. For the following, $\mathrm{r}_M(x)$ will denote the rank[8] of element x in the multiset M. An *(ε-approximate) GK summary* $\mathbf{GK} = \mathbf{GK}_{\varepsilon,M}$ over a multiset M is a finite *non-decreasing* sequence $\langle x_1, x_2, \dots, x_\ell \rangle$, $x_j \in M$, where $x_1 = \min M$ and $x_l = \max M$, together with two functions $\mathrm{r}^{\min}_{\mathbf{GK}}$, $\mathrm{r}^{\max}_{\mathbf{GK}}$[9] such that $\mathrm{r}^{\min}(x_1) = \mathrm{r}^{\max}(x_1) = 1$, $\mathrm{r}^{\min}(x_\ell) = \mathrm{r}^{\max}(x_\ell) = |M|$ and

$$\mathrm{r}^{\min}(x_j) \le \mathrm{r}_M(x_j) \le \mathrm{r}^{\max}(x_j) \qquad (5)$$

for all $x_j \in M$ and

$$\mathrm{r}^{\max}(x_{j+1}) - \mathrm{r}^{\min}(x_j) \le 2\varepsilon |M|, \qquad (6)$$

for all $j = 1, 2, \dots, \ell - 1$. The intuition behind (5) is that we maintaining lower and upper bounds on the true ranks of the elements in the summary and, in addition, from (6), these approximations are "dense" with respect to M.

For the in-network aggregation procedure, in order to describe an underlying multiset M we are not going to maintain at every step just a single GK summary, but instead collections $\{\mathbf{GK}_{\varepsilon_1,M_1}, \mathbf{GK}_{\varepsilon_2,M_2}, \dots, \mathbf{GK}_{\varepsilon_k,M_k}\}$, wich we' ll call *extended GK summaries*, such that the M_j's are mutually disjoint multisets with $M = \bigcup_{j=1}^k M_j$ and no two GK summaries are of the same *class*, where the class of a GK summary is defined to be $\mathtt{class}(\mathbf{GK}_{\varepsilon_j,M_j}) = \lfloor \log |M_j| \rfloor$. Notice that the above properties trivially implies

$$k \le \log |M|. \qquad (7)$$

Initialization. Fix some (desired approximation) $\varepsilon > 0$. Each sensor i creates an extended GK summary $\{\mathbf{GK}_{\varepsilon/2,\mathcal{S}_i}\}$ consisting of a single GK summary, by sorting the entire stream $\mathcal{S}_i$ and picking elements of rank $1, \varepsilon |\mathcal{S}_i|$, $2\varepsilon |\mathcal{S}_i|, \dots, |\mathcal{S}_i|$ and setting, naturally enough, $\mathrm{r}^{\max}_{\mathbf{GK}_{\varepsilon/2,\mathcal{S}_i}} = \mathrm{r}^{\min}_{\mathbf{GK}_{\varepsilon/2,\mathcal{S}_i}} = \mathrm{r}_{\mathcal{S}_i}$. It is easy to see that the summary created is a valid $\varepsilon/2$-approximate GK summary, i.e. property (6) holds, and that $|\mathbf{GK}_{\varepsilon/2,\mathcal{S}_i}| \le 1 + 1/\varepsilon$.

Aggregation. Let $\mathbf{eGK}_1$, $\mathbf{eGK}_2$ be two extended GK summaries. We aggregate them by initially taking their union $\mathbf{eGK}_1 \cup \mathbf{eGK}_2$. The resulting summary may not be valid since it is possible to contain more than one (but at most two) GK summaries of the same class. We deal with this by scanning $\mathbf{eGK}_1 \cup \mathbf{eGK}_2$, in increasing order of classes, and whenever we find two GK summaries having the same class κ, we merge them into one new GK summary of class $\kappa + 1$. After this process is completed, we apply a compression to the extended GK summary to reduce the size of its newly created GK summaries. We describe these two operations, called by Greenwald and Khanna [18] *combine* and *prune*, below:

[8]Defining a notion of rank for multisets may initially seem problematic, since the same value x may appear more than once in M. We can override this difficulty by differentiating elements of the same value by adding, for example, a time-stamp parameter to our observations and sorting our streams using a lexicographic ordering. For the remaining of this survey we will assume that for every multiset M there is a predefined sorting with which we can answer order statistics $M_{(j)}$ for M and define $\mathrm{r}_M(x)$ to be the unique index j, $j = 1, 2, \dots, |M|$ such that $x = M_{(j)}$.

[9]To keep the notation light, we will feel free to drop the subscript $\mathbf{GK}_{\varepsilon,M}$ whenever it is clear to which GK summary we are referring to.

- *Combine:* Let $\mathbf{GK}_1 = \mathbf{GK}_{\varepsilon_1, M'_1} = \langle x_1, x_2, \dots, x_{\ell_1} \rangle$, $\mathbf{GK}_2 = \mathbf{GK}_{\varepsilon_2, M'_2} = \langle y_1, y_2, \dots, y_{\ell_2} \rangle$ be the two GK summaries, of the same class κ, we wish to merge. For each $x \in \mathbf{GK}_1$ define

$$y'(x) = \max\{y \in \mathbf{GK}_2 \mid y < x\}$$
$$y''(x) = \min\{y \in \mathbf{GK}_2 \mid y > x\}$$

  Now, create the new quantile summary $\mathbf{GK}' = \mathbf{GK}_{\varepsilon', M_1 \cup M_2}$ by simply sorting the union $\mathbf{GK}_1 \cup \mathbf{GK}_2$ and defining functions $\mathrm{r}^{\min}_{\mathbf{GK}'}$, $\mathrm{r}^{\max}_{\mathbf{GK}'}$ as

$$\mathrm{r}^{\min}_{\mathbf{GK}'}(x) = \begin{cases} \mathrm{r}^{\min}_{\mathbf{GK}_1}(x) + \mathrm{r}^{\min}_{\mathbf{GK}_2}(y'(x)), & y'(x) \text{ exists,} \\ \mathrm{r}^{\min}_{\mathbf{GK}_1}(x), & \text{otherwise,} \end{cases}$$

  and

$$\mathrm{r}^{\max}_{\mathbf{GK}'}(x) = \begin{cases} \mathrm{r}^{\max}_{\mathbf{GK}_1}(x) + \mathrm{r}^{\max}_{\mathbf{GK}_2}(y''(x)) - 1, & y''(x) \text{ exists,} \\ \mathrm{r}^{\max}_{\mathbf{GK}_1}(x) + \mathrm{r}^{\max}_{\mathbf{GK}_2}(y'(x)), & \text{otherwise,} \end{cases}$$

  for all $x \in \mathbf{GK}_1$. The definitions for $y \in \mathbf{GK}_2$ are completely symmetric. It can be shown ([18]) that $\varepsilon' \leq \max\{\varepsilon_1, \varepsilon_2\}$, i.e. $\mathbf{GK}'$ is a $\max\{\varepsilon_1, \varepsilon_2\}$-approximate GK summary.

- *Prune:* Given a GK summary $\mathbf{GK} = \mathbf{GK}_{\varepsilon', M}$ and a compression parameter B (to be determined later) we compress it by picking only elements of rank $1, |M|/B, 2|M|/B, \dots, |M|$ along with their original $\mathrm{r}^{\min}$, $\mathrm{r}^{\max}$ values. It is easy to see that the resulting structure is an $(\varepsilon' + 1/(2B))$-approximate GK summary over the same underlying multiset M, with a reduced size of at most $B + 1$.

When the in-network aggregation reaches the base station, instead of merging the GK summaries of the same class, we use the combine operation to merge *all* GK summaries together, into a single summary upon which we perform a final prune operation to reduce its size. The resulting GK summary describes our entire data set $\bigcup_i \mathcal{S}_i$ and, most importantly, it can be shown that it is ϵ-approximate [18], by setting $B = \frac{1}{\varepsilon} \log n$.

Evaluation: Based on the previous, at the end of the tree-routing aggregation we are left with an extended GK summary $\{\mathbf{GK}_{\varepsilon, \cup_i \mathcal{S}_i}\}$ at the root. From that, we extract the answer to the ϕ-quantile query Q be returning $Q(\bigcup_i \mathcal{S}_i) = x$ where x is the unique element in $\mathbf{GK}_{\varepsilon, \cup_i \mathcal{S}_i}$ such that $\mathrm{r}^{\min}(x) \leq \phi n \leq \mathrm{r}^{\max}(x)$.

It is straightforward to show ([53]) that, due to properties (5) and (6), an ε-approximation GK summary can answer a (ϕ, ε)-quantile query. Also, during th in-network aggregation, each sensor transmits an extended GK summary consisting of at most $\log n$ (see (7)) GK summaries, each one of which has a size of at most $B + 1 = O(\frac{1}{\varepsilon} \log n)$ (due to the prune operation). This gives an efficient in-network procedure for computing quantiles, using a communication load of $O(\frac{1}{\varepsilon} \log^2 n)$ per sensor.

Finally, we should mention that Greenwald and Khanna [18] give two variations of the GK summaries we presented, that may provide better performance guarantees in case we have further information about the height h of the aggregation tree. In case h is sufficiently small, say $h = O(\log n)$, for example in balanced trees with constant size streams per sensor, by simply combining at every node *all* GK summaries, irrespectively of class, to a single summary and appling the prune operation with h/ε compression, we get a communication load of $O(h/\varepsilon)$. On the other hand, by introducing an additional, third operation called *reduce*, which ensures that every node transmits $O(\log(h/\varepsilon))$ quantile summaries, a performance of $O(\frac{1}{\varepsilon} \log n \log \frac{h}{\varepsilon})$ can be achieved, giving better results for $h = o(n)$ (which is the case for many real life topologies).

q-digests vs GK summaries. Although both summaries are capable of efficiently answering (ϕ, ε)-quntiles, their performances, $O(\frac{1}{\varepsilon} \log N)$ and $O(\frac{1}{\varepsilon} \log^2 n)$, respectively, are not directly comparable. Theoretically, q-digests ensure a better communication load when our observation set $\bigcup_i \mathcal{S}_i$ is very large but the values are drawn from a moderate sized universe U and GK summaries perform better with smaller streams drawn from a very large universe. For experimental evaluation, the reader is refered to the original papers [52], [18]. We should also note that, although GK summaries are ingenious constructions, which also demonstrate very well how algorithms for traditional streaming problems ([53]) can be extended to sensor networks scenarios, q-digests are simpler to implement and analyze and well-fitted for distributed streaming settings, properties that make them easily extendable to more complex problems (see, e.g., QDFM summaries in section 3.3). On the other side, while GK summaries can be applied to real-valued universes U, q-digests assume integer values.

Another subtle point worth mentioning, is that of space complexity at the lower level of in-sensor computation. The initialization operations we give in these section for both summaries (and are proposed in the original papers) require *linear* space computation by each sensor, with respect to the length m_i of its own stream. This means that in case some sensor i observes a large stream with length of the order of the entire data set, i.e. $m_i = \Theta(n)$, the initial generation of summary instances can be computationally intensive (and impractical). Finally, notice that both q-digests and GK summaries are not duplicate-insensitive. If, for example, one aggregates (1) some instance of these summaries with itself, a new, different instance is produced (violating property (2)).

CM summaries. In section 2.4 we described Count-Min sketches [22], primarily in the content of answering heavy-hitters queries. CM sketches are very versatile data structures that can be used to approximate many other aggregates, such as quantile queries. Furthermore, they can be easily used for in-network aggregation in distributed streaming settings, since it is trivial to see (from the definition of the update procedure of CM sketches) that if $\mathbf{CM}_{M_1}$, $\mathbf{CM}_{M_2}$, $\mathbf{CM}_{M_1 \cup M_2}$ are the matrices of the CM sketches of the multisets M_1, M_2 and their union $M_1 \cup M_2$, then

$$\mathbf{CM}_{M_1 \cup M_2} = \mathbf{CM}_{M_1} + \mathbf{CM}_{M_2},$$

where the $+$ operation is a standard matrix addition. Using this as an aggregation operation (1) we can maintain CM summaries to answer many queries in distributed settings. The evaluation operation (computed by the base station) are the same as those used to extract answers in centralized settings (see section 2.4 page 8 and [22]).

Like q-digests and GK summaries, they are not duplicate-insensitive. The main disadvantage, though, of using CM summaries for tree-based computation of ε-approximate quantiles is that the communication load bound $O(\frac{1}{\varepsilon} \log^2 n \log\log \frac{n}{\delta})$ they give is inferior to those of the q-digest and GK summaries and, in addition, it is only guaranteed with high probability $1-\delta$ (in contrast to q-digests and GK summaries that are deterministic). However, CM summaries have the advantage of being easily extendable and, combined with other data structures, can be made to work for many different problems and settings (see, e.g. the duplicate-insensitive CMFM summaries of section 3.3).

3.2.2. Distinct items

As we saw in section 2.4, FM sketches [16] can be used in centralized streaming settings to answer distinct items queries (`DCOUNT`) ε-approximately with high probability of $1-\delta$, maintaining m bitmaps of length k for a total space requirement of $mk = O(\frac{1}{\varepsilon^2} \log n \log \frac{1}{\delta})$. Due to the decomposability of FM sketches (see section 2.4) they can be readily used as summaries for in-network computation of `DCOUNT`, using as aggregation operation (1) the bitwise `OR` of the corresponding bitmaps. The evaluation operation is the same as in the centralized setting, i.e. we output $(1/\phi)2^{\sum_s r_s/m}$ where r_s is the position of the leftmost 0 bit in the s-th bitmap of the FM sketch describing our entire data set $\bigcup_i \mathcal{S}_i$ and $\phi \approx 0.77351$ is a constant.

But the most important property of such FM summaries is that, unlike any other summary we have presented so far for tree-based aggregation, is *duplicate insensitive.* Due to this, they will be the most essential building blocks to construct more complex duplicate insensitive summaries to deal with multi-path aggregation problems in section 3.3.

3.3. Multi-path aggregation

Our exposition in the previous section 3.2 was built upon the assumption that our networks are absolutely reliable: partial aggregate information transmitted by each node always reaches the base station. So, the aggregation procedure is developed in a specific, well-defined way from the leaves to the root, across unique (shortest) paths, each data point from our observation set being inserted only once (through the aggregation operation (1)) into our summaries and maintained until all information reaches the base station.

However, as we discussed in the introductory section 3.1.2, such an assumption is far from realistic in sensor networks, where packet loss and node failures in tree-based protocols can have catastrophic consequences to the evaluation of our queries, especially when they occur near the root. A simple and fundamental solution to this problem is to deploy *multi-path* routing protocols, where the partial infromation transmitted by each node is aggregated across many different paths towards the base-station. The motivation is obvious: even at the event of multiple communication failures, it is very likely that a large portion of our information has reached the root and thus, been included in the final aggregate output.

Although this generic routing principle surely addresses the issue of lossy networks, another problem arises: the same information may reach the base station through different paths, resulting to some data points from our observation set being aggregatted more than once and over-contributing to the final aggregate result. The solution is to construct summaries that are *order and duplicate insensitive (ODI)* [54], i.e. are not affected by the order in which data points are aggregated (inserted) into them, neither by the insertion of multiple copies of the same element. We have formally introduced duplicate insensitivity in section 3.1.3. Regarding order insensitivity, as we mentioned in section 2.1, page 3, we have been silently assuming this property and will continue to do so throughout this survey, since all aggregates and summaries we will present are order-insensitive. More formally, an order insensitive summary has a symmetric aggregating function (1), i.e. $F(\mathbf{Sk}_1, \mathbf{Sk}_2) = F(\mathbf{Sk}_2, \mathbf{Sk}_1)$.

3.3.1. Summary diffusion

Nath et al. [54] proposed a formal framework to study multi-path aggregation, called *summary diffusion.* The main idea is very similar to that of the TAG framework of Madden et al. [14] and what we presented in sections 3.1.3 and 3.1.2 in the context of tree-based aggregation: first, durig a *distribution phase* the aggregate query is flooded through the network and then, during the *aggregation phase* the partial information maintained by each sensor is routed towards the base station, performing an in-network computation of the query. However, in contrast to tree-based routing, there is not a single, predefined structure implied upon our nodes and so this must be constructed (dynamically) during the distribution phase and a routing protocol decided for the aggregation phase.

For this, many best-effort multi-path routing schemes, that try to build "good" paths from the sensors to the base station, have been proposed, e.g. directed diffusion [55, 25], braided diffusion [56] and GRAB [57]. For example, consider the *Rings* topology of Nath et al. [54]: during the distribution phase the sensors are assigned to different rings $R_0, R_1, \ldots, R_m$ around the base station. The base station is the only node in R_0 and a node is assigned to ring R_j if it received the flooded query from a node belonging in R_{j-1}. The hop distance of the nodes in R_j is exactly j. During the aggregation phase, the sensors in the outermost ring R_m *initialize* summary instances describing their underlying streams and broadcast them to all sensors in ring R_{m-1} within, of course, transmission range. Nodes in ring R_{m-1} *aggregate* the summaries they received with the summary they have generated for their own stream and forward the new information to ring R_{m-2}. The procedure is continued until reaching the base station for a final *evaluation* of the query.

The authors in [54] also give formal necessary and sufficient conditions for ODI-correctness, however this is out of the scope of this survey, and we refer the reader to the original paper. We must mention that alternative, fully-decentralized topologies for robust aggregation, that are not base-station-centred and do not fall into the directed diffusion framework have been proposed, most notably *gossiping* protocols [58–61].

3.3.2. Duplicate insensitive summaries for duplicate sensitive queries

In this section, we present duplicate insensitive techniques to answer duplicate sensitive queries. As we have discussed, this is essential in multi-path aggregation protocols. We use the duplicate insensitive FM summaries (see section 3.2.2) as building blocks. For an experimental evaluation of the summaries we present here, we refer to [5] and [54]. Before proceeding, we need to slightly enrich our distributed streaming model of section 3.1.2. It is crucial to be able to tell the difference between true, undesired duplicates inserted due to multi-path aggregation and different observations that just happen to be of the same value. So, each sensor i will not observe just single values $x \in U$ but tuples $x = \langle s(x), t(x), v(x) \rangle$ where $v(x) \in U$ is the value of element x, $t(x)$ is some kind of unique id for observation x, e.g. a timestamp, and $s(x) = i$ is the sensor observing x. In this way, two data points x, y are true duplicates if and only if $s(x) = s(y)$ and $t(x) = t(y)$. Streams, now, are multisets of such triplets.

Observation set cardinality (`COUNT`*).* This query asks for the number $n = |\cup_i \mathcal{S}_i|$ of data points in our entire observation set. A special case is that of finding the number of currently alive nodes in a sensor network (just assume that each sensor i observes a single value i). It is easy to see that the answer to this query is the same as the distinct elements query `DCOUNT` of section 3.2.2 (since instead of simple values, our data points are unique triplets), so the FM summaries can be deployed (of course, the hash functions need to take triplets as input).

Sums (`SUM`*).* For this problem, assume that each sensor i has computed the sum of its observations, denoted by $c_i = \sum_{j=1}^{m_i} v(a_{ij})$ and we ask for $\texttt{SUM}(c_1, c_2, \ldots, c_{|I|})$. One obvious solution is for each sensor to generate c_i tuples of the form $\langle i, j \rangle$, $j = 1, 2, \ldots, c_i$, initialize an FM summary with them and, essentially, compute the `COUNT` query of the previous paragraph. The only drawback is that for each initial value of c_i we need to perform c_i distinct insertions, one for each tuple, and this $O(c_i)$ expected running time may be impractical for large values of c_i.

Considine et al. [5] provide an $O(\log c_i)$ time and $O(c_i / \log c_i)$ space, per insertion of $\langle i, c_i \rangle$, algorithm which emulates the result of the above simple idea of c_i distinct insertions but without actually performing them. In an earlier version [34] of that paper an alternative approach giving $O(\log^2 c_i)$ time and $O(c_i / \log^2 c_i)$ space bounds is presented. Notice, however, that for small c_i values, these scalable insertion algorithms may perform poorly as opposed to the naive approach described in the previous paragraph.

In a higher level concerning our main issue of communication load, the $O(\frac{1}{\varepsilon^2} \log n)$ guaranty of the FM summaries obviously holds. Nath et al. [54] give an alternative duplicate insensitive summary for `SUM` that is based on a variant of FM sketches presented in [15], but the algorithm of Considine et al. [5] provides better accuracy in practice.

Quantiles. Considine et al. [5] combined the power of q-digests (see section 3.2.1) with the duplicate insensitivity of FM summaries to construct a duplicate insensitive summary for answering ε-approximate quantile queries. The idea behind the QDFM summaries is to use FM sketches with each bucket, instead of just the simple counters. However, due to the presence of duplicate triplets, it is possible that a new bucket not satisfying property (3) may occur during a merging, and so, no formal analytical guarantees regarding the communication load are provided.

Count-min summaries. As we have already mentioned (see sections 3.2.1, page 15), CM summaries [22] can be used to approximate many important aggregate queries, including quantiles and frequent items. Cormode and Muthukrishnan [62] extended these summaries to make them duplicate insensitive. The underlying idea is to use FM sketches in place of the counters (the entries of the CM matrix). A detailed, formal analysis regarding the approximation guarantees of these CMFM summaries can be found in [5]. A similar idea, of combining duplicate insensitive summaries for `SUM` with CM sketches is also present in [54].

Uniform Sampling. Finally, we should not forget fundamental ideas such as sampling, which is straightforward to implement and analyze. Nath et al. [54] describe a duplicate-insensitive uniform sampling summary. In general, using a random sample of size $O(\frac{1}{\varepsilon^2} \log \frac{1}{\delta})$ we can answer ε-approximate quantile queries with a probability of $1 - \delta$ (see [63]).

3.4. Continuous queries

Up to now, we have approached the issue of distributed streaming in a totally static way: we were answering aggregates based on a single snapshot of the underlying observation set, at a given point in time. However, sensor networks are usually monitoring evolving phenomena and receive huge amounts of data, dynamically over time and in high rates. Naturally enough, we would like to extend our *single-shot* querying algorithms to capture such dynamic processes and be able to *continuously* monitor and extract critical information from the underlying infrastructure, in near real-time. Of course, such an undertaking imposes further challenges with respect to algorithmic design, since the naive approach of simply performing repeated, one-shot queries to simulate constant monitoring, requires unrealistic amounts of computational and communication resources and would quickly drain battery life. So, efficiency and satisfactory approximation, once again, play a crucial role.

In this time-depending setting, we must revisit our model assumptions of section 3.1.2. Instead of considering streams simply as multisets on our universe U, we must adopt formal models in the spirit of section 2.1. Most results are analyzed under the simple time series model, however can be easily extended to fit into the more general turnstile model and even handle sliding-window semantics (see, e.g., [27, 38, 64]). Also, a common assumption is that the remote sensors do *not* communicate with each other, but are all directly linked to a base-station (coordinator), i.e. we have a tree-structure hierarchy of depth 1, although also this assumption can be extended to capture multi-level tree hierarchies (e.g. [27]).

The majority of the algorithms proposed for answering continuous queries follow the same basic principle: they assign *local filters* [65] to each sensor, to prevent unnecessary communication. These are threshold constraints and only if they are violated the sensor pushes new information to the base station and an update of our network is triggered. Obviously, in such schemes there is a trade-off between how elastic we are when setting these constraints, i.e. what degree of variation in each sensor's observations we consider insignificant to report, and the communication load incurred. Depending on the various algorithms and summaries, the overall *slack* [65] (error tolerance) is dynamically distributed and updated across the participating sites, coordinated by the base station which is responsible for keeping track and adjusting the individual slacks and balancing the overall effect. Many important aggregate queries can be addressed in this way: algebraic [66], top-k [64], fundamental techniques for duplicate insensitivity [38], and others, e.g. [67].

A recent direction that extends this standard approach of local-slack allocation where each sensor's behavior is considered constant (or irrelevant) between consecutive update triggering, is that of introducing *prediction models* that can be maintained by the coordinator and the remote sites, capturing trends and predictions with respect to the evolution of data over time. In a way, these models play both the role of local filters, since the communication can be prevented as long as a sensor's stream does not deviate substantially from the model currently maintained, as well as the role of reliable estimators that can answer quickly and continuously our aggregate queries. For such approaches see [68, 27, 69–71].

Finally, in many sensor network deployments we are interested in detecting, as soon as possible, anomalies in the distribution of the incoming data and critical events. This is particular the case in security settings (both physical and electronic) and environmental monitoring, such as fire detection. Several algorithms have been proposed for such settings of *distributed threshold triggering*, e.g. [72–74].

4. Future directions

Distributed data streaming is a very challenging, wide and active research area that touches upon many disciplines within Computer Science. From an algorithmic point of view, we need to lay solid foundations and models upon which we can develop a formal theory of lower/upper bounds and approach communication complexity theory issues more rigorously. In a nutshell, the goal is to reach the maturity of the field of classical data streaming. Some steps towards this approach are being made, see e.g. [26].

Regarding the existing work we presented in this survey, apparently there is large space to provide formal complexity guarantees for algorithms and summaries that, up to now, we can only evaluate by experimental results. Another direction would be to study richer classes of queries, e.g. set-valued query answers and more complex tasks such as performing machine learning over our distributed settings (see, e.g. [75]). Finally, security issues play a major role in real life distributed computation and provide a fertile ground upon which we can adapt aggregation problems (see [76]). For a more detailed analysis on possible future directions, we refer to [65].

Acknowledgements

This work has been partially supported by the ICT Programme of the European Union under contract number ICT-2008-215270 (FRONTS). The third author acknowledges the support of Propondis Foundation. We thank Elias Koutsoupias for helpful discussions.

References

[1] S. Muthukrishnan, Data streams: Algorithms and applications, Now Publishers Inc, 2005.

[2] B. Babcock, S. Babu, M. Datar, R. Motwani, J. Widom, Models and issues in data stream systems, in: Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, ACM, Madison, Wisconsin, 2002, pp. 1–16.

[3] J. Gama, P. P. Rodriques, Data stream processing, in: J. a. Gama, M. M. Geber (Eds.), Learning from Data Streams: Processing Techniques in Sensor Networks, Springer, 2007, pp. 25–39.

[4] N. Shrivastava, C. Buragohain, Aggregation and summarization in sensor networks, in: J. a. Gama, M. M. Geber (Eds.), Learning from Data Streams: Processing Techniques in Sensor Networks, Springer, 2007, pp. 87–106.

[5] J. Considine, M. Hadjieleftheriou, F. Li, J. Byers, G. Kollios, Robust approximate aggregation in sensor data management systems, ACM Trans. Database Syst. 34 (2009) 1–35.

[6] S. Muthukrishnan, Massive Data Streams Research: Where to Go, 2010. `http://algo.research.googlepages.com/lect1.pdf`.

[7] M. Garofalakis, Processing massive data streams, in: VLDB Database School, Cairo University.

[8] G. Cormode, M. Garofalakis, Streaming in a connected world: querying and tracking distributed data streams, in: SIGMOD '07: Proceedings of the 2007 ACM SIGMOD international conference on Management of data, ACM, New York, NY, USA, 2007, pp. 1178–1181.

[9] J. Barros, Sensor networks: An overview, in: J. a. Gama, M. M. Geber (Eds.), Learning from Data Streams: Processing Techniques in Sensor Networks, Springer, 2007, pp. 9–24.

[10] M. M. Gaber, Data stream processing in sensor networks, in: J. a. Gama, M. M. Geber (Eds.), Learning from Data Streams: Processing Techniques in Sensor Networks, Springer, 2007, pp. 41–50.

[11] D. Culler, D. Estrin, M. Srivastava, Overview of sensor networks, IEEE Computer 37 (2004) 41–49.

[12] S. Subramaniam, D. Gunopulos, A survey of stream processing problems and techniques in sensor networks, in: C. Aggarwal (Ed.), Data Streams: Models and Algorithms, Advances in Database Systems, Springer, 2007, pp. 333–352.

[13] C. C. Aggarwal, An introduction to data streams, in: C. Aggarwal (Ed.), Data Streams: Models and Algorithms, Advances in Database Systems, Springer, 2007, pp. 1–8.

[14] S. Madden, M. J. Franklin, J. M. Hellerstein, W. Hong, Tag: a tiny aggregation service for ad-hoc sensor networks, SIGOPS Oper. Syst. Rev. 36 (2002) 131–146.

[15] N. Alon, Y. Matias, M. Szegedy, The Space Complexity of Approximating the Frequency Moments, Journal of Computer and System Sciences 58 (1999) 137–147.

[16] P. Flajolet, G. N. Martin, Probabilistic counting algorithms for data base applications, J. Comput. Syst. Sci. 31 (1985) 182–209.

[17] J. I. Munro, M. S. Paterson, Selection and sorting with limited storage, in: SFCS '78: Proceedings of the 19th Annual Symposium on Foundations of Computer Science, IEEE Computer Society, Washington, DC, USA, 1978, pp. 253–258.

[18] M. B. Greenwald, S. Khanna, Power-conserving computation of order-statistics over sensor networks, in: Proceedings of the twenty-third ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems - PODS '04, ACM Press, New York, New York, USA, 2004, pp. 275–285.

[19] J. a. Gama, M. M. Geber (Eds.), Learning from Data Streams: Processing Techniques in Sensor Networks, Springer, 2007.

[20] M. Datar, A. Gionis, P. Indyk, R. Motwani, Maintaining stream statistics over sliding windows, in: SIAM Journal on Computing, pp. 635–644.

[21] Z. Bar-Yossef, T. S. Jayram, R. Kumar, D. Sivakumar, L. Trevisan, Counting distinct elements in a data stream, in: Proceedings of the 6th International Workshop on Randomization and Approximation Techniques, Springer-Verlag, Cambridge, Ma, USA, 2002, pp. 1–10.

[22] G. Cormode, S. Muthukrishnan, An improved data stream summary: the count-min sketch and its applications, J. Algorithms 55 (2005) 58–75.

[23] A. C. Gilbert, S. Guha, P. Indyk, Y. Kotidis, S. Muthukrishnan, M. J. Strauss, Fast, small-space algorithms for approximate histogram maintenance, in: STOC '02: Proceedings of the thiry-fourth annual ACM symposium on Theory of computing, ACM, 2002, pp. 389–398.

[24] S. Nath, P. B. Gibbons, S. Seshan, Z. R. Anderson, Synopsis diffusion for robust aggregation in sensor networks, in: Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems, ACM, 2004, pp. 250–262.

[25] C. Intanagonwiwat, R. Govindan, D. Estrin, Directed diffusion: a scalable and robust communication paradigm for sensor networks, in: MobiCom '00: Proceedings of the 6th annual international conference on Mobile computing and networking, ACM, 2000, pp. 56–67.

[26] G. Cormode, S. Muthukrishnan, K. Yi, Algorithms for distributed functional monitoring, in: SODA '08: Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms, Society for Industrial and Applied Mathematics, 2008, pp. 1076–1085.

[27] G. Cormode, M. Garofalakis, S. Muthukrishnan, R. Rastogi, Holistic aggregates in a networked world: distributed tracking of approximate quantiles, in: SIGMOD '05: Proceedings of the 2005 ACM SIGMOD international conference on Management of data, ACM, New York, NY, USA, 2005, pp. 25–36.

[28] G. Cormode, S. Muthukrishnan, W. Zhuang, What's different: Distributed, continuous monitoring of duplicate-resilient aggregates on data streams, in: ICDE '06: Proceedings of the 22nd International Conference on Data Engineering, IEEE Computer Society, 2006, p. 57.

[29] M. R. Henzinger, P. Raghavan, S. Rajagopalan, Computing on data streams, External memory algorithms (1999) 107–118.

[30] A. Borodin, R. El-Yaniv, Online computation and competitive analysis, Cambridge University Press, 1998.

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

[32] A. Z. Broder, M. Charikar, A. M. Frieze, M. Mitzenmacher, Min-wise independent permutations, J. Comput. Syst. Sci. 60 (2000) 630–659.

[33] T. Bohman, C. Cooper, A. M. Frieze, Min-wise independent linear permutations, The Electronic Journal of Combinatorics, 7 (2000).

[34] J. Considine, F. Li, G. Kollios, J. W. Byers, Approximate aggregation techniques for sensor databases, in: ICDE, pp. 449–460.

[35] M. Hadjieleftheriou, J. Byers, G. Kollios, Robust Sketching and Aggregation of Distributed Data Streams, Technical Report 2005-011, CS Department, Boston University, 2005.

[36] D. Achlioptas, Database-friendly random projections: Johnson-lindenstrauss with binary coins, J. Comput. Syst. Sci 66 (2003) 671–687.

[37] Z. Bar-Yossef, T. S. Jayram, R. Kumar, D. Sivakumar, aloan information statistics approach to data stream and communication complexity, J. Comput. Syst. Sci. 68 (2004) 702–732.

[38] G. Cormode, S. Muthukrishnan, W. Zhuang, What's different: Distributed, continuous monitoring of duplicate-resilient aggregates on data streams, in: Proceedings of the 22nd International Conference on Data Engineering, ICDE 2006, IEEE Computer Society, 2006, p. 57.

[39] M. Durand, P. Flajolet, Loglog counting of large cardinalities, in: Algorithms-ESA 2003, volume 2832, Springer, 2003, pp. 605–617.

[40] G. S. Manku, S. Rajagopalan, B. G. Lindsay, Random sampling techniques for space efficient online computation of order statistics of large datasets, in: SIGMOD '99: Proceedings of the 1999 ACM SIGMOD international conference on Management of data, ACM, 1999, pp. 251–262.

[41] A. Metwally, D. Agrawal, A. E. Abbadi, An integrated efficient solution for computing frequent and top-k elements in data streams, ACM Trans. Database Syst. 31 (2006) 1095–1133.

[42] S. Guha, Space efficiency in synopsis construction algorithms, in: VLDB '05: Proceedings of the 31st international conference on Very large data bases, pp. 409–420.

[43] C. Aggarwal (Ed.), Data Streams: Models and Algorithms, Advances in Database Systems, Springer, 2007.

[44] V. Braverman, R. Ostrovsky, Smooth histograms for sliding windows, in: Proceedings of the IEEE Symposium on Foundations of Computer Science, pp. 283–293.

[45] B. Babcock, M. Datar, R. Motwani, Sampling from a moving window over streaming data, in: SODA '02: Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete

algorithms, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2002, pp. 633–634.

[46] G. Cormode, S. Tirthapura, B. Xu, Time-decaying sketches for sensor data aggregation, in: Proceedings of the 26th annual ACM symposium on Principles of distributed computing (PODC), ACM, New York, NY, USA, 2007, pp. 215–224.

[47] A. Pavan, S. Tirthapura, Range-efficient counting of distinct elements in a massive data stream, SIAM Journal of Computing 37 (2007) 359–379.

[48] L. Becchetti, E. Koutsoupias, Competitive analysis of aggregate max in windowed streaming, in: ICALP '09: 36th International Colloquium on Automata, Languages and Programming, Springer-Verlag, 2009, pp. 156–170.

[49] M. Resvanis, I. Chatzigiannakis, Experimental evaluation of duplicate insensitive counting algorithms, Panhellenic Conference on Informatics 0 (2009) 60–64.

[50] J. Zhao, R. Govindan, D. Estrin, Computing aggregates for monitoring wireless sensor networks, in: Proceedings of the First IEEE International Workshop on Sensor Network Protocols and Applications, pp. 139–148.

[51] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, H. Pirahesh, Data cube: A relational aggregation operator generalizing group-by, cross-tab, and subtotals, Data Mining and Knowledge Discovery 1 (1997) 29–53.

[52] N. Shrivastava, C. Buragohain, D. Agrawal, S. Suri, Medians and beyond: new aggregation techniques for sensor networks, in: Proceedings of the 2nd international conference on Embedded networked sensor systems, ACM, 2004, pp. 239–249.

[53] M. Greenwald, S. Khanna, Space-efficient online computation of quantile summaries, in: SIGMOD '01: Proceedings of the 2001 ACM SIGMOD international conference on Management of data, ACM, New York, NY, USA, 2001, pp. 58–66.

[54] S. Nath, P. B. Gibbons, S. Seshan, Z. Anderson, Synopsis diffusion for robust aggregation in sensor networks, ACM Trans. Sen. Netw. 4 (2008) 1–40.

[55] C. Intanagonwiwat, R. Govindan, D. Estrin, J. Heidemann, F. Silva, Directed diffusion for wireless sensor networking, IEEE/ACM Trans. Netw. 11 (2003) 2–16.

[56] D. Ganesan, R. Govindan, S. Shenker, D. Estrin, Highly-resilient, energy-efficient multipath routing in wireless sensor networks, in: MobiHoc '01: Proceedings of the 2nd ACM international symposium on Mobile ad hoc networking & computing, ACM, New York, NY, USA, 2001, pp. 251–254.

[57] F. Ye, G. Zhong, S. Lu, L. Zhang, Gradient broadcast: a robust data delivery protocol for large scale sensor networks, Wirel. Netw. 11 (2005) 285–298.

[58] D. Kempe, A. Dobra, J. Gehrke, Gossip-based computation of aggregate information, in: FOCS '03: Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Washington, DC, USA, 2003, p. 482.

[59] S. Boyd, A. Ghosh, B. Prabhakar, D. Shah, Randomized gossip algorithms, IEEE/ACM Trans. Netw. 14 (2006) 2508–2530.

[60] J.-Y. Chen, G. Pandurangan, D. Xu, Robust computation of aggregates in wireless sensor networks: Distributed randomized algorithms and analysis, IEEE Trans. Parallel Distrib. Syst. 17 (2006) 987–1000.

[61] A. G. Dimakis, A. D. Sarwate, M. J. Wainwright, Geographic gossip: efficient aggregation for sensor networks, in: IPSN '06: Proceedings of the 5th international conference on Information processing in sensor networks, ACM, New York, NY, USA, 2006, pp. 69–76.

[62] G. Cormode, S. Muthukrishnan, Space efficient mining of multigraph streams, in: ACM (Ed.), Proceedings of the Twenty-Fourth ACM Symposium on Principles of Database Systems, ACM Press, 2005, pp. 271–282.

[63] Z. Bar-Yossef, R. Kumar, D. Sivakumar, Sampling algorithms: lower bounds and applications, in: STOC '01: Proceedings of the thirty-third annual ACM symposium on Theory of computing, ACM, New York, NY, USA, 2001, pp. 266–275.

[64] B. Babcock, C. Olston, Distributed top-k monitoring, in: SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data, ACM, New York, NY, USA, 2003, pp. 28–39.

[65] M. Garofalakis, Distributed data streams, in: L. Liu, M. T. Ozsu (Eds.), Encyclopedia of Database Systems, Springer, 2009, pp. 883–890.

[66] C. Olston, J. Jiang, J. Widom, Adaptive filters for continuous queries over distributed data streams, in: SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data, ACM, New York, NY, USA, 2003, pp. 563–574.

[67] A. Das, S. Ganguly, M. Garofalakis, R. Rastogi, Distributed set-expression cardinality estimation, in: VLDB '04: Proceedings of the Thirtieth international conference on Very large data bases, VLDB Endowment, 2004, pp. 312–323.

[68] G. Cormode, M. Garofalakis, Sketching streams through the net: distributed approximate query tracking, in: VLDB '05: Proceedings of the 31st international conference on Very large data bases, VLDB Endowment, 2005, pp. 13–24.

[69] G. Cormode, M. Garofalakis, Efficient strategies for continuous distributed tracking tasks, IEEE Data Engineering Bulletin 28 (2005) 33–39.

[70] D. Chu, A. Deshpande, J. Hellerstein, W. Hong, U. Berkeley, Approximate data collection in sensor networks using probabilistic models, in: Data Engineering, 2006. ICDE'06. Proceedings of the 22nd International Conference on, pp. 48–48.

[71] A. Meliou, C. Guestrin, J. M. Hellerstein, Approximating sensor network queries using in-network summaries, in: IPSN '09: Proceedings of the 2009 International Conference on Information Processing in Sensor Networks, IEEE Computer Society, Washington, DC, USA, 2009, pp. 229–240.

[72] L. Huang, X. Nguyen, M. Garofalakis, J. Hellerstein, M. Jordan, A. Joseph, N. Taft, Communication-efficient online detection of network-wide anomalies, in: IEEE INFOCOM 2007. 26th IEEE International Conference on Computer Communications, pp. 134–142.

[73] R. Keralapura, G. Cormode, J. Ramamirtham, Communication-efficient distributed monitoring of thresholded counts, in: SIGMOD '06: Proceedings of the 2006 ACM SIGMOD international conference on Management of data, ACM, New York, NY, USA, 2006, pp. 289–300.

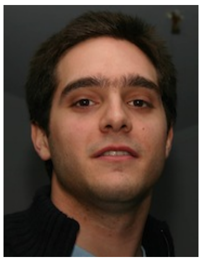
[74] I. Sharfman, A. Schuster, D. Keren, A geometric approach to monitoring threshold functions over distributed data streams, in: SIGMOD '06: Proceedings of the 2006 ACM SIGMOD international conference on Management of data, ACM, New York, NY, USA, 2006, pp. 301–312.

[75] C. Guestrin, P. Bodik, R. Thibaux, M. Paskin, S. Madden, Distributed regression: an efficient framework for modeling sensor network data, in: IPSN '04: Proceedings of the 3rd international symposium on Information processing in sensor networks, ACM, New York, NY, USA, 2004, pp. 1–10.

[76] M. Garofalakis, J. Hellerstein, P. Maniatis, I. Berkeley, Proof sketches: Verifiable in-network aggregation, in: IEEE 23rd International Conference on Data Engineering, 2007. ICDE 2007, pp. 996–1005.

Dr. Luca Becchetti received his PhD in Computer Engineering in 1999 and is currently a Research Associate at the University of Roma "La Sapienza". He has co-authored over 35 publications in international journals and conferences. His main research interests lie in the design and analysis of algorithms for network resource management and scheduling and in the design of efficient algorithms for the analysis of large and complex networks and massive data sets. Luca Becchetti is currently involved in the program committees of international conferences and workshops.

Dr. Ioannis Chatzigiannakis obtained his Ph.D. from the Department of Computer Engineering & Informatics of the University of Patras in 2003. He is currently Adjunct Faculty at the Computer Engineering & Informatics Department of the University of Patras (since October 2005). He is the Director of the Research Unit 1 of RACTI (since July 2007). He has coauthored over 70 scientific publications. His main research interests include distributed and mobile computing, wireless sensor networks, algorithm engineering and software systems. He has served as a consultant to major Greek computing industries. He is the Secretary of the European Association for Theoretical Computer Science since July 2008.



Yiannis Giannakopoulos obtained his B.Sc. in Mathematics in 2006 and his M.Sc. in Logic, Algorithms and Computation in 2008, both from the University of Athens. He is currently a Ph.D. student at the Computer Science department of the same university, under the supervision of Prof. Elias Koutsoupias, and a Propondis Foundation scholar. His main research interests include online algorithms, algorithmic game theory and data streaming.