Continuous Distributed Monitoring: A Short Survey

Graham Cormode
AT&T Labs—Research
graham@research.att.com

ABSTRACT

In the model of continuous distributed monitoring, a number of observers each see a stream of observations. Their goal is to work together to compute a function of the union of their observations. This can be as simple as counting the total number of observations, or more complex non-linear functions such as tracking the entropy of the induced distribution. Assuming that it is too costly to simply centralize all the observations, it becomes quite challenging to design solutions which provide a good approximation to the current answer, while bounding the communication cost of the observers, and their other resources such as their space usage. This survey introduces this model, and describe a selection results in this setting, from the simple counting problem to a variety of other functions that have been studied.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems; F.2 [**Theory of Computation**]: Analysis of Algorithms and problem complexity

General Terms

Algorithms, Theory

Keywords

Continuous Distributed Monitoring, Data Streams, Communication Protocols

1. INTRODUCTION

The model of continuous, distributed monitoring is a quite natural one, which has arisen only in the early years of the 21st century. It abstracts an increasingly common situation: a number of observers are making observations, and wish to work together to compute a function of the combination of all their observations. This abstract description can be applied to a number of settings:

- Network elements within the network of a large ISP are observing local usage of links, and wish to work together to compute functions which determine the overall health of the network.
- Many sensors have been deployed in the field, with the aim of collecting environmental information, and need to cooperate to track global changes in this data.
- A large social network monitors the usage of many compute nodes in data centers spread around the world, and wants to coordinate this information to track shifts in usage patterns and detect any unusual events, possibly indicative of an attack or exploit.

Each of these examples maps naturally onto the outline above: the network elements, sensors and compute nodes respectively play the part of the observers, who want to collaborate in the computation.

There are various "trivial" solutions to these problems. Studying the drawbacks of these helps us to identify the properties to optimize. A first approach is to simply have all the observers send all their observations to a single, centralized location. For cases where the flow of new observations is sufficiently slow, then indeed this is a satisfactory solution. However, in the above scenarios, this places an intolerable burden on the underlying network. For example, in the ISP example, the number of observations may be equivalent to the total number of packets traveling on a link: generating this much extra traffic on the network for the purpose of health monitoring will quickly contribute to the ill-health of the network!

A second approach is to perform "periodic polling": at some fixed interval, say every five minutes, or once an hour, a central monitor polls each observer for information about their observations since the last poll, and collates these together to get a snapshot of the current status. Again, in some situations, this will suffice. Indeed, many network protocols, such as the Simple Network Management Protocol (SNMP) operate on exactly this basis. Still, often this too is insufficient. Firstly, we require that the information needed can be summarized compactly. For example, SNMP allows the reporting of the total amount of traffic (measured in packets or bytes) processed by a network element within a given time window. Quantities like sums and counts of observations, therefore, fit naturally within this setting. However, when the objective is a more complex function, like measuring some non-linear function of all the (distributed) observations, or detecting when some complex event has occurred, it is less clear how periodic polling can operate.

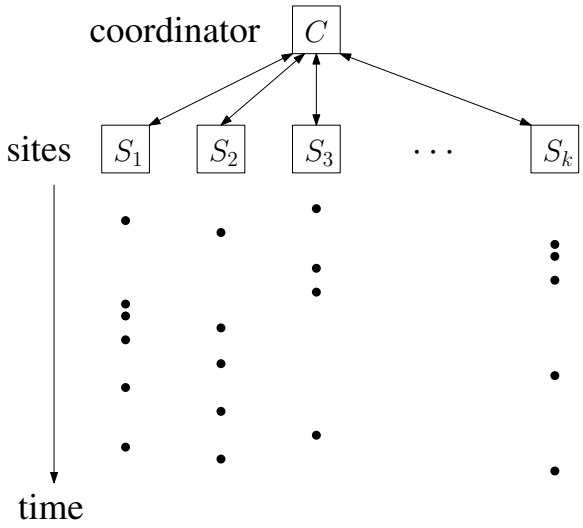

Figure 1: Continuous Distributed Monitoring model

The other limitation of periodic polling is the careful balance needed in setting the frequency of the polling event. Set the gap too narrow, and again the network becomes overloaded with data which may be of limited usefulness. But set the gap too large, and the delay between an important event occurring and it being detected by the protocol may become too large.

In continuous distributed monitoring, we aim to address all these concerns. The central idea is to incur minimal communication when there is nothing important being observed, but at the same time to enable rapid (near-instantaneous) updates when necessary.

There has been considerable research effort in this area since its inception. Progress has principally been made by considering different fundamental functions, and describing protocols which provide strong guarantees on the accuracy of the monitoring, while incurring low costs, in the form of communication required, and computational overhead and storage needed by the observers.

Outline. The rest of this survey proceeds as follows. First, we formalize the model, and define the key cost measures. Then we begin by considering a seemingly simple problem in this setting, the problem of counting a fixed number of events, in Section 2. Section 3 considers monitoring the information theoretic concept of entropy, which varies non-monotonically as the number of events increase. We then describe a very general approach to problems in this model via the "geometric approach", in Section 4. Section 5 considers how to maintain a random sample, of either the entire data, or only a recent selection. In Section 6, we outline the history of the model and other results in this area, while Section 7 presents some concluding remarks and open problems.

1.1 Formalizing the model

In total we have k observers (or sites), indexed $S_1, \ldots S_k$. Each observer sees a stream of observations. Typically, each individual observation is quite simple, but in aggregate these define a complex whole. For example, in a communication network, each event might be the arrival of a packet at a router. The description of each event is quite simple: the destination and payload size, say. But the overall distribution of traffic to different destinations observed over multiple routers is very large and complex.

We treat the observations as items $A = a_1, a_2, \ldots a_n$, such that each observation is seen by exactly one observer. There is also a central site, or coordinator, C, who can communicate directly with each observer. For simplicity, we do not allow communication between observers (this can be achieved by sending messages through the coordinator), and we assume each message has unit cost. Varying these assumptions leads to different cost models, some of which are studied in the works described in Section 6. The goal of the monitoring is for the coordinator to continually track some function $f(A)$ over the complete set of observations.

In this survey, we see several different cases of this problem. In 'threshold monitoring', the goal is to determine whether $f(A)$ is above or below a threshold τ. For example, we may want to know when the total network traffic in the last hour exceeds a given amount; or when the entropy of this traffic distribution exceeds a given bound. In 'value monitoring', the goal is to provide an estimate $\hat{f}(A)$ of $f(A)$, such that the difference $|\hat{f}(A) - f(A)|$ is bounded. In the network example, this corresponds to providing an approximate value of the total network traffic; or of the entropy of the traffic distribution. In 'set monitoring', the goal is to provide a set of values which satisfy some property. This could be a uniform sample of the input items, or an approximate top-k (e.g. the top-k most popular destinations in the network).

Figure 1 gives a schematic of the model: communication is between the coordinator and the k different sites. New observations are made over time, which prompts more communication between the parties.

1.2 Comparison to Other Models

There are several other models of computation over data which may be rapidly arriving or distributed. Here, we identify some common models, and outline the key differences.

Communication Complexity. The model of communication complexity focuses on the case where there are two parties, Alice who holds input x and Bob who holds input y, and they wish to work together to compute $f(x, y)$ for some fixed function f [31]. The most important difference between this model and the continuous distributed monitoring case is that the inputs x and y are fixed for communication complexity, whereas in our case, they are allowed to vary. Moreover, it turns out that the main focus of communication complexity is providing lower bounds or impossibility results for various functions, whereas in continuous distributed monitoring, there has been most interest in providing protocols with low communication costs. However, the models are closely related: techniques from communication complexity have been used to show lower bounds for problems in continuous distributed monitoring.

The Data Streaming Model. In the streaming model, a single observer sees a large stream of events, and must keep a sublinear amount of information in order to approximate a desired function f [33]. This omits the key feature of the continuous distributed model, the fact that multiple distributed observers need to compute a function of all their inputs combined. While each observer in our model sees a

stream of inputs, the model does not insist that they use sublinear space—rather, the space used by each observer is an additional property of any given protocol. However, it is often desirable that the observers use small space, and techniques from stream processing are therefore useful to help achieve this.

Distributed Computation. Clearly, the continuous distributed model is a special case within the general area of distributed computation. The focus on continually maintaining a function of evolving input distinguishes it from the general case. There are other models within distributed computation, such as the Distributed Streams Model [20, 21] or the Massive, Unordered Data model [18]. These capture the emphasis on distributed streams of data, but focus on a one-time computation, rather than continually tracking a function.

2. THE COUNTDOWN PROBLEM

We begin with a seemingly simple problem which nevertheless admits some fairly sophisticated solutions. In the *countdown problem*, each observer sees some events (non-overlapping, so each event is seen by only one observer), and we wish to determine when a total of τ events have been seen. This is an instance of threshold monitoring. This abstract problem captures many natural settings: we want to raise an alert when more than τ unusual network events have been seen; report when more than 10,000 vehicles have crossed a highway; or identify the 1,000,000th customer; and so on. A trivial solution has each observer send a bit for each event they observe, which uses $O(\tau)$ communication. We aim to considerably improve over this baseline.

2.1 A first approach

A smarter approach takes advantage of the fact that there have to be many events at each site before the threshold τ can have been reached. A necessary condition is that at least one of the k sites must observe τ/k events before the threshold can be reached. This leads to a relatively simple scheme (derived from [30]): Each site begins with an initial upper bound value of τ/k, and begins to observe events. Whenever its local count n_i exceeds this upper bound, it informs the coordinator, which collects n_i from each observer, and the n_is are reset to zero. From these, we can determine the current "slack": the difference S between the current count N and the threshold τ, i.e. $S = \tau - N$. This slack can then be redistributed to the observers, so each site now enforces an upper bound of S/k on n_i. Each iteration reduces the slack by a factor of $(1 - 1/k)$. When the slack (initially τ) reaches k, the observers can switch to reporting every event. The number of slack updates is then

$$\log_{1/(1-\frac{1}{k})}\left(\frac{\tau}{k}\right) = \frac{\log(\frac{\tau}{k})}{\log(\frac{1}{1-\frac{1}{k}})} = O\big(k \log \frac{\tau}{k}\big)$$

The total communication is $O(k^2 \log \tau/k)$, since each update causes communication of $O(k)$.

2.2 A quadratic improvement

The step of updating every node whenever one node reports that it has exceeded its current local threshold is somewhat wasteful. This can be improved on by tolerating more updates before a global communication is triggered. This idea was introduced in [10], and we follow the simplified version described in [11].

Now the protocol operates over $\lceil \log(\tau/k) \rceil$ rounds. In the jth round, each observer sends a message to the coordinator when its local count n_i reaches $\lfloor 2^{-j}\tau/k \rfloor$, and then subtracts this amount from n_i. So, in the first round, this bound is $\lfloor \tau/2k \rfloor$. In the jth round, the coordinator waits until it has received k messages, at which point the round is terminated, and the coordinator alerts each site to begin the $j+1$th round, causing the bound to approximately halve. This continues until the bound reaches 1, when each site reports each event when it occurs. Observe now that the communication in each round is more "balanced": the sites send a total of k messages, and the coordinator sends k messages (to inform each site that the new round has begun). Each of these messages can be constant size. Thus, the total communication is $O(k \log \tau/k)$: a factor k improvement over the naive approach.

It also follows immediately that protocol is correct: in any round, the total "unreported" count is at most

$$k\lfloor \tau 2^{-j}/k \rfloor \leq \tau/2^j,$$

while the "reported" count is at most

$$\sum_{i=1}^{j-1} k\lfloor \tau 2^{-i}/k \rfloor \leq \tau \sum_{i=1}^{j-1} 2^{-i} \leq \tau(1 - 2^{-j}).$$

Hence, the total count never exceeds τ until the final round, when every event is reported directly.

Approximate Countdown. We can improve on the cost of this protocol if we are prepared to tolerate some imprecision in the result. Specifically, we consider protocols which approximate the answer. To approximate, we introduce a parameter ϵ, and ask that the coordinator can determine that the true count is below $(1-\epsilon)\tau$ or above τ; when the true count is in between, then the coordinator can indicate either state.

The protocol is almost identical, but now we terminate when the bound on the unreported count reaches $\epsilon\tau$. The number of rounds is reduced to $\log 1/\epsilon$. This removes τ from the bounds, and makes the total cost of the protocol $O(k \log 1/\epsilon)$ communication.

2.3 A randomized twist

Countdown lower bounds. We might ask if we can improve further on this result. For deterministic solutions, the answer is no: this bound is tight. This was shown formally in [10]. The intuition is natural: consider the perspective of a single observer, who witnesses a number of events. When this number is substantial enough, it could be part of a global trend, and so must be reported in case they push the total count above the threshold τ. At the same time, it might just be a local phenomenon, in which case any communication does not change the overall answer. Since the observer cannot distinguish these two cases unless it receives a message from the coordinator, then it is forced to communicate. Based on this argument, it is possible to show that the total amount of communication is at least $\Omega(k \log \tau/k)$.

Randomized Countdown Protocol. However, we can give tighter bounds if we relax the requirements, and allow both randomization and approximation. Allowing randomization means that we let the protocol have a small prob-

ability of giving an erroneous answer at some point in its operation.

The randomized protocol operates as follows, based on a constant c determined by the analysis. Each site observes events, and after collecting a "bundle" $\epsilon^2\tau/(ck)$ of observations, it decides whether to send a message to the coordinator. With probability $1/k$ it sends a message, but with probability $1 - 1/k$, it stays silent. The coordinator declares that enough events have been seen once it has received $c(1/\epsilon^2 - 1/2\epsilon)$ messages. The idea here is that there will be enough opportunities to send messages that with high probability the coordinator will not declare too early or too late.

Cost of the randomized protocol. We omit a full analysis here, but observe that each message corresponds to k bundles in expectation, which correspond to $k\epsilon^2\tau/ck = \epsilon^2\tau/c$ events. Then the coordinator declares after

$$c(1/\epsilon^2 - 1/2\epsilon)\epsilon^2\tau/c = \tau(1 - \epsilon/2)$$

elements have arrived (in expectation). Setting c to a moderate constant is sufficient to ensure that this random variable does not deviate too far from its expectation. Further, the number of events seen at each site but not reported is at most $\epsilon^2\tau/c$, which is a tiny fraction of τ, and is absorbed in the bounds.

Lastly, we can observe that the amount of communication from sites is $O(1/\epsilon^2)$. Note that this is independent of k, the number of observers! It is perhaps surprising that there can be protocols whose cost is independent of k (although this ignores the cost to initiate and terminate the protocol, which should involve contacting all k sites). In this case, the intuition is that we can (almost) treat the events as if they are being observed at a single site: the randomization is independent, and the same across all sites. So it does not matter how the "bundles" are spread around. And if some site does not receive enough events to fill a bundle, it can just stay quiet and not participate in the protocol, without introducing any significant error.

3. MONITORING ENTROPY

The approach outlined for the countdown problem relied critically on the fact that the function being monitored was monotonic: the number of events kept increasing. As a result, it was easier to bound the communication needed. But more generally, we are interested in functions that may not depend monotonically on their input. A key example is the entropy function. Consider the case where the observers are now witnessing events in the form of arrivals of different items. These arrivals generate an empirical probability distribution (recording the relative proportion of each different item observed), which we can compute the entropy of.

Entropy. Specifically, suppose that f_i denotes the number of occurrences of item i observed across the whole system, and m denotes the total number of items (so $m = \sum_i f_i$). Then the empirical probability of i is just f_i/m, and the entropy H of the distribution is given by

$$H = \sum_i \frac{f_i}{m} \log \frac{m}{f_i}$$

Observe that H is non-monotone: as the f_is vary, this quantity can rise and fall. Nevertheless, the entropy H is an important metric on the distribution: if all f_is are about equal, then the entropy is high, while if most f_is are small and only one or a few are significant, then the entropy is low. It has been argued that changes in entropy are an important indicator of changes in behavior in distributed systems and networks [32].

Entropy Protocol Outline. Arackaparambil *et al.* design a protocol to monitor entropy in the continuous distributed monitoring model [2]. Specifically, they design an approximate protocol, which determines whether the current entropy H is above a given boundary τ, or below $(1 - \epsilon)\tau$.

The first problem to overcome is how to collect information on the current distribution at each site that can be combined to approximate the entropy of the global distribution. In general, the full description of the distribution can be large when there are many items with non-zero frequency. Here, we can take advantage of recent results on *sketching* entropy. The idea of a sketch data structure is to create a compact summary of data, so that multiple sketches can be combined, and an estimate obtained for a function (in this case, entropy) of the combination of the inputs. There are sketches which provide $(1 \pm \epsilon)$ approximation of the entropy using a data structure of size $\tilde{O}(\frac{1}{\epsilon^2})$, where the $\tilde{O}$ notation suppresses logarithmic factors [19, 22]. They also carry a small probability of giving an estimate outside this range, and some care is required in combining the sketch guarantees to obtain the desired overall accuracy, but we gloss over these details in this presentation.

Given such sketches, the overall protocol is then surprisingly straightforward. In fact, the key step is an invocation of an approximate protocol for the countdown problem from Section 2. The protocol proceeds in a number of rounds. In the first round, each site sends every item it receives directly to the coordinator, until some constant number (say, 100) of items have been observed across all sites. This is because the entropy can change quickly in this initial stage. In each subsequent round i, the coordinator computes a parameter τ_i, and runs an instance of the approximate countdown protocol for threshold τ_i, with a constant approximation factor $\epsilon = \frac{1}{2}$. When this protocol terminates, the coordinate contacts each site, which sends a sketch of its current distribution. The coordinator combines these to estimate the current entropy, and uses this to compute the parameter τ_{i+1} for the next round.

Analysis Details. The reason that this process can work relies on a basic property of the entropy function: the change in entropy between two points is bounded in terms of the number of new observations. Specifically, if the number of observations at the first point is m, and there are n new arrivals, the change in entropy is at most $\frac{n}{m}\log(2m)$ [2]. Thus, since we know the entropy (approximately) at the end of round i, and we wish to know if it changes by at most $\epsilon\tau/2$ (the minimum change needed to change the output of the coordinator), we can set

$$\tau_{i+1} = \frac{\epsilon\tau m}{2\log(2m)},$$

where m is the total number of observations made at the end of round i. Given an upper bound N on the total number of observations, we can ensure that m_i, the total number of

observations at the end of round i, satisfies

$$m_{i+1} = m_i + \tau_{i+1} = m_i \left(1 + \frac{\epsilon\tau}{2\log(2m_i)}\right)$$
$$\geq m_i \left(1 + \frac{\epsilon\tau}{2\log(2N)}\right)$$

and hence the number of rounds to reach N observations is $O(\frac{1}{\epsilon\tau}\log^2 N)$ (provided $\log N \geq \tau\epsilon$). The total communication cost is to send k sketches and run the countdown protocol in each round, so the cost is dominated by the $O(\frac{k}{\epsilon\tau}\log^2 N)$ sketches sent. The precise calculations, which require some adjusting of constants and rescaling of parameters, are given in [2].

Lower bounds for entropy monitoring. Lower bounds for this problem can be generating by defining a set of possible inputs chosen so that any individual site cannot tell which case it is in, and so is forced to communicate to resolve this uncertainty. This leads to a deterministic lower bound of $\Omega(k\epsilon^{-1/2}\log(\epsilon N/k))$ and a randomized lower bound of $\Omega(\epsilon^{-1/2}\log(\epsilon N/k))$. Note that, apart from the sketches, the above protocol is essentially deterministic, and so the stronger bound applies to this case. It remains to close the gap between these upper and lower bounds.

4. THE GEOMETRIC APPROACH

The two results we have seen thus far have considered specific problems (countdown and entropy), and provided tailored protocols based on exploiting specific properties of each function. At this point it is natural to ask whether there are general purpose techniques for generating protocols in this model. The "geometric approach", due to Sharfman, Schuster and Keren aims to do exactly this [36]. The basic idea is to take any desired function, f, and break down the testing of whether $f(x) > \tau$ or $f(x) \leq \tau$ into conditions which can be checked locally, even though x represents the global state of the system. The central result relies on a neat geometric fact, that the area of a convex hull of a set of points can be fully covered by a set of spheres, one sphere incident on each point.

4.1 Formal Description

Preliminaries. Each stream observed at each site is assumed to define a current d dimensional vector v_i. In the countdown case, each v_i was simply the local count; in the entropy case it was the local frequency distribution. With each site we associate a weight λ_i such that these weights sum to 1, i.e. $\sum_{i=1}^k \lambda_i = 1$. These weights might reflect the number of observations at each site, so in this case $\lambda_i = n_i / \sum_{i=1}^k n_i$. Or they may simply be uniform, i.e. $\lambda_i = 1/k$ for all i. Initially, assume that these weights are fixed and known to all nodes.

The weighted combination of all local vectors v_i gives the global vector $v = \sum_{i=1}^k \lambda_i v_i$. The instance of the threshold monitoring problem is then to determine whether $f(v) \leq \tau$ or $f(v) > \tau$, for a fixed function f and threshold τ. For example, we can map the countdown problem into this setting: here, we set $\lambda_i = 1/k$, each v_i is the single dimensional quantity n_i (number of event observations at site i), and $f(v) = \|v\|_1$. We set τ here to be $1/k$ times the desired threshold. In other words, v is the mean of the event counts at each site, and we want to alert when this mean exceeds a threshold that implies that the total count is above the global threshold.

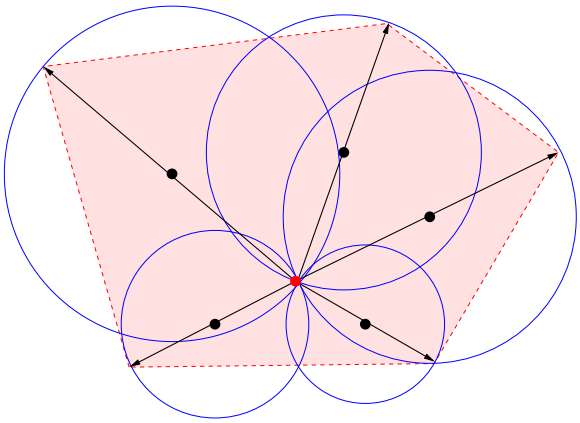

Figure 2: Current estimate e (central red dot), drift vectors Δv_i (arrows out of e), convex hull (dotted outline) and enclosing balls

Protocol Description. At any moment during the protocol, each site has previously informed the coordinator of some prior state of its local vector, v_i'. So the coordinator knows v_i', but not the current state v_i. Based on this knowledge, the coordinator has an estimated global vector $e = \sum_{i=1}^k \lambda_i v_i$. Clearly, if the local vectors v_i move too far from their last reported value v_i', it is possible that the τ threshold may be violated. Therefore, each site monitors its *drift* from its last reported value, as $\Delta v_i = v_i - v_i'$. Thus we can write the current global vector, v, in terms of the current estimate e and the drift vectors:

$$v = \sum_{i=1}^k \lambda_i v_i = \sum_{i=1}^k \lambda_i (e + \Delta v_i) = e + \sum_{i=1}^k \lambda_i \Delta v_i$$

Observe that this is a convex combination of drift vectors. Therefore, the current global vector v is guaranteed to lie somewhere within the convex hull of the drift vectors v_i around e. Figure 2 shows an example in $d = 2$ dimensions, with five drift vectors emanating from an estimate e, and their convex hull. The current value must lie somewhere within this shaded region.

To transform the global condition into a local one, we place a ball on each local drift vector, of radius $\frac{1}{2}\|\Delta v_i\|_2$ and centered at $e + \frac{1}{2}\Delta v_i$. This is illustrated in Figure 2. It can be shown that the union of all these balls entirely covers the convex hull of drift vectors [36]. Thus, we reduce the problem of monitoring the global vector to the local problem of each site monitoring the ball of its drift vector.

Specifically, given the function f, we can partition the space into two sets: X, which is those points x for which $f(x) \leq \tau$, and $\bar{X}$, which is those for which $f(x) > \tau$. Note that these each set is not necessarily connected, but may consist of many disjoint regions. The basic protocol is now quite simple: each site monitors its drift vector Δv_i, and checks with each new observation if the ball given by $e + \frac{1}{2}\Delta v_i$ is *monochromatic*, i.e. all points in the ball fall in the same set (X or $\bar{X}$). If this is not the case, then the site communicates to the coordinator. The coordinator then collects the current vectors v_i from each site to compute a

new estimate e, which resets all drift vectors to 0. From the above discussion of convex hulls, it is clear that when all balls are monochromatic in the same set (X or $\bar{X}$), then v must also be in the same set, and so the coordinator knows the correct state.

Countdown Example. Applying this to the example of the countdown problem, the effect is to set $e = \sum_{i=1}^{k} v_i$, and the "ball" for each site allows each site to receive up to $(\tau/k - e)$ new updates before triggering another communication. In other words, this scheme is equivalent to the basic scheme in Section 2.1: each site is given a uniform fraction of the current "slack", and a global communication is forced whenever any site uses up its slack. This exposes both a strength and a weakness of this approach. The direct application of the geometric approach immediately generates a quite natural protocol for the problem in question, without any detailed study of the local semantics. At the same time, in this case it is apparent that improved results can be obtained by generating protocols which take more account of the function in question. Further, while the geometric method is very general, it does not lead to bounds on the communication used, unless we make some statistical assumptions about how the updates vary [36]. To some extent this is unavoidable, since the scheme allows arbitrary updates to the monitored vectors, allowing a worst case where the threshold τ is repeatedly crossed, forcing a lot of communication.

4.2 Extensions

There are several extensions and variations of this basic geometric monitoring scheme which are able to reduce the cost, and avoid some bad cases.

- **Local Resolution via slack.** Whenever a local drift vector creates a non-monotone ball, it precipitates communication with all sites, to collect their current vectors and distribute the new estimate. This global communication can be be postponed by the coordinator, who can introduce additional "slack", in the form of offset vectors. That is, the coordinator can contact a small number of sites, and allocate a set of vectors δ_i chosen so that the balls for $\Delta v_i + \delta_i$ are now monochromatic, and $\sum_{i=1}^{k} \delta_i = 0$. In the countdown example, this could correspond to one site i which has seen a large number of events being told to subtract δ_i from its count while site j is asked to add δ_i to its count. In other words, the slack in this case is used to even out the counts. This idea is discussed in detail in [36]; similar concepts arose earlier, e.g. in work on tracking top-k of frequency distributions [3].

- **Approximate Thresholds.** The version of the protocol described is for an exact version. As we saw in the earlier examples, demanding to know exactly when the threshold τ is reached can require a lot of communication. In the geometric setting, it is easy to reach a scenario where the global vector remains on one side of the threshold, but the balls cannot grow very large before they become non-monochromatic, which causes a lot of communication. As before, we can reduce this impact by relaxing the requirement, and introducing an ϵ tolerance around τ. Applying this, when $f(v) < \tau$, we define the sets X and $\bar{X}$ as before, but when we are above the threshold, we define the sets based on $f(x) < (1-\epsilon)\tau$. This gives more room for the balls to grow, and prevents constant communication when the current value of $f(v)$ is close to τ.

- **Testing Monotonicity.** We have assumed that, given a description of a ball B, it is easy to compute whether it is monochromatic (equivalently, to compute the maximum or minimum value of $f(x)$ for $x \in B$). For simple (differentiable) functions, this may be the case, but in general this can be more complex. In some cases, we can assume that all vectors fall on a grid, and perform a simple search of the grid. In other cases, we may need more insight into the function being monitored to define fast tests for monochromicity.

- **Affine Transformations and Reference Vectors.** The use of spherical balls is a natural one, but it is not the only choice. In [37], the authors observe that one can perform any affine transformation on the input, without changing the region covered by the convex hull. Then spherical balls in the transformed space correspond to (rotated, sheared) ellipsoids in the original space. In some cases, these ellipsoids can more tightly conform to the convex hull than spheres would. In the same work, the authors discuss a variation of the scheme, which replaces the estimate e with a difference reference vector, and argues that this can give better results by providing a larger "safe area" for the drift vectors to occupy.

- **Higher-dimensional data.** The geometric monitoring scheme as presented operates on vectors in d-dimensional space, and transmits these to compute and share the estimate e. In simple cases, such as the countdown case ($d = 1$), there is no problem. But applying this method directly to the entropy case is problematic: d can grow very large here (when there are very many distinct items possible), and the factor of d is unacceptable. The protocol in Section 3 avoids this by sending compact "sketches" of the distribution, whose size is independent of d. In principle, sketches can be combined with geometric monitoring. Now the vectors exist in "sketch space", with small dimension d. This makes most sense when using sketches that embed high dimensional spaces into low dimensional Euclidean space, via the Johnson-Lindenstrauss lemma [29, 26], so balls in the original space correspond to balls in the sketch space. However, care must be taken, since now we need to test the monochromicity of these balls in the sketch space.

5. SAMPLING

So far we have concentrated on the case of threshold monitoring: tracking which side of a threshold τ a given function f is on. This is actually quite a general task. For example, we might instead want to monitor the value of f, so that we always have an approximation to its value (value monitoring). But this can be modeled as multiple instances of the threshold monitoring task, for thresholds $1, (1+\epsilon), (1+\epsilon)^2 \ldots$. Tracking all these in parallel can be done by running $O(\frac{1}{\epsilon} \log T)$ instances of the threshold monitoring solution in parallel, where T is maximum value of the function. Although this $1/\epsilon$ factor is large enough to make

it worthwhile designing solutions for value monitoring problems, the techniques and approaches that have been used for value monitoring and threshold monitoring are quite similar.

However, there are some other monitoring tasks which do not fit either the threshold monitoring or value monitoring paradigms, and instead require us to track the members of a set (set monitoring). For example, we might want to extract information such as which are the k most frequently observed items across all the event streams [3]. In this section, we describe another basic task: to draw a uniform sample from the different event streams, based on the results from [11]. We describe two variations: where we want to draw a sample over all the events ever observed (the infinite window case), and where we want a sample only over the more recent events (the sliding window case).

5.1 Infinite Window

Recall the set-up: we have k distributed sites, each of which is observing events occurring at unpredictable and varying rates. We wish to compute a sample of size s of these events. First, we consider drawing a sample without replacement. The basic idea is to sample across all sites with the same probability p. All sampled items are sent to the coordinator to form a collection, from which s items can be extracted uniformly. Periodically, the coordinator will broadcast to all sites to reduce p, and will also prune its collection. We want to bound the resources taken for this process, in terms of the amount of communication, and space needed by the participants.

Binary Bernoulli Sampling. The solution of [11] introduces a sampling idea called "Binary Bernoulli Sampling". The idea is that every sampling rate p will be a power of 2. To sample an item with probability $p = 2^{-i}$, we can create i random bits, so each bit is chosen independently 0 with probability 1/2, 1 with probability 1/2. Then we include an item in the sample if its random bits are i zeros. This is quite straightforward, but is convenient to use, since it makes reasoning about probabilities very easy. For example, suppose we want to switch from sampling with probability 2^{-i} to probability 2^{-j} for $j > i$. For new items, we just pick j bits, and sample if they are all 0. But we can also subsample old items: these have a prefix of i random bits, so we just pick $j - i$ new random bits, and accept if these too are all zeros. It is then easy to see that both processes give items the same chance to be sampled, since they are conditioned on the same event: picking j random bits all zero.

Infinite Window Sampling Protocol. The protocol is quite straightforward. In round i, each site samples each item with probability 2^{-i}, as outlined above. If the item is chosen, it is forwarded to the coordinator. We begin in round 0, and advance to the next round $i + 1$ when the coordinator sends a message. The coordinator's work is a little more complex. For each item received, it picks an additional random bit, so the item is now associated with $i + 1$ random bits. The coordinator ends round i when it has s items in its collection with $i + 1$ zeros in their random bits. It then ejects all items from the collection whose bits are 0^i1, and keeps those (s) with 0^i0. It goes on to pick an extra random bit for those that remain in the collection, and broadcasts the new round to all sites. At any time during the protocol, the coordinator can pick a uniform subset from its current collection to provide a sample of size exactly s.

Correctness and Analysis. The correctness of this process follows immediately from the discussion of binary Bernoulli sampling. That is, the coordinator always maintains a collection of at least s items, so that all items in round i are chosen uniformly and independently with probability 2^{-i}. We need to calculate how long each round lasts, in terms of how many items are sent by sites. As the coordinator receives each item, it chooses a new random bit, and the round ends when s of those are 0. So the only way that a round can be very long is if the coordinator is unlucky, and picks many more 1 values than 0s. Via Chernoff bounds, the probability of a round involving more than $4s$ items being sent is exponentially small in s; similarly, the total number of rounds while processing n input items is bounded with very high probability as $O(\log n)$. So the communication cost is bounded (with high probability) as $O((k + s) \log n)$.

Moreover, one can show a lower bound of $\Omega(k + s \log n)$ by arguing that this many different items should appear in a random sample over the course of the protocol [11]. So the protocol is optimal when $k \approx s$; when this is not the case, the costs can be reduced to $O(k \log k/sn + s \log n)$ by modifying the sampling probabilities slightly [12]. The protocol can also be extended to sample with replacement. A trivial solution just runs the above protocol with $s = 1$ in parallel s times over. However, this blows up the costs by a factor of s. Instead, it is possible to take this idea, but to keep all instances of the protocol operating at the same level i, thus reducing the communication from the coordinator. Analyzing this process allows us to argue that communication of this protocol is bounded by $O((k + s \log s) \log n)$.

5.2 Sliding Windows

A natural variation of continuous distributed monitoring problems is when we do not want to track events across an unbounded history, but rather to see only the impact of recent events. For example, in a network we may only want to include events which have happened within the last hour; in a sensor network, we may only want to track a window of 1 million recent events, and so on. A naive solution would just be to pick a fixed interval—say, 1 hour—and restart the protocol afresh at multiples of this interval. This has the benefit of simplicity, but means that we re-enter a 'start-up' phase every time the protocol restarts, and so we lose information and history around this time. Instead, we describe an approach that is almost as simple as this naive solution, but which provides a sample of an exact sliding window.

A Tale of Two Windows. The key insight needed to generate the solution is due to Braverman, Ostrovsky and Zaniolo [4], who observed that any sliding window can be decomposed into two pieces, relative to a fixed point in time: a growing window as new items arrive after the fixed point, and a shrinking or expiring window of items from before the fixed point. Suppose we want to maintain a sample of items drawn from the last W global arrivals. To draw a sample uniform from these W, we want to take all unexpired sampled items from the expiring window, and make up the shortfall by sampling from those in the growing window. A simple probability calculation shows that this does indeed provide us a uniform sample from the most recent W arrivals.

To implement this idea, we can run an instance of the countdown protocol to count off every W arrivals. We can also run an instance of the above sliding window protocol for

drawing a sample beginning at every multiple of W arrivals, which we halt when W further items have arrived. The only additional information needed is that the coordinator needs to know when an item sampled in the expiring window has expired. This can be done by starting a fresh instance of the countdown problem for every sampled item (and terminating this when the item is ejected from the coordinator's collection). This gives the coordinator exactly what is needed to perform the above sampling process: drawing unexpired items from the expiring window, and making up the shortfall from the growing window. The cost of this protocol now grows as $O(ks \log(W/s))$ per window, but higher cost is unavoidable: [11] shows that any protocol for this problem must incur an $\Omega(ks \log(W/ks))$ cost.

6. OTHER WORK

The idea of continuous distributed monitoring is a natural one, and as such it has arisen independently in different areas, under different labels. An early form was as 'Reactive Monitoring' in the networking world. Here, Dilman and Raz introduced a problem that was essentially a variant of the countdown problem, and provided some solutions based on distributing slack amongst the observers [17]. The notion of testing whether a function had exceeded a global threshold appeared under the name of "distributed triggers", and was motivated by Jain *et al.* in a workshop paper [27].

The continuous distributed model has attracted most attention in the data management community. Early work by Olston, Jiang and Widom focused on tracking a function over single values which could vary up and down, such as monitoring the sum [35]. Here, some uncertainty can be tolerated, so they introduce a natural "filter" approach, which assigns a local filter to each site so that if the current value is within the filter, it does not need to be reported. When a site's value falls outside its filter, the current value is reported, and the filter is re-centered on this value. Over time, some filters can be widened and others narrowed so that the total uncertainty remains bounded, but more slack is allocated to values that are less stable.

A similar approach was used by Babcock and Olston to report the top-k items from a distribution [3]. Again, some tolerance for approximate answers is necessary to avoid communicating every change. The central idea is to choose a set of "adjustment factors" for each item at each site, so that the local distribution after adjustment appears identical to the global distribution. Each site monitors its (adjusted) distribution, and reports if the local (adjusted) top-k changes. In this case, a costly 'rebalancing' stage is invoked.

The concept of "prediction" was introduced by Cormode, Garofalakis, Muthukrishnan and Rastogi [9]. The idea is that if items are continually arriving at approximately even rates, then each site can share a simple prediction model of where its distribution will be at any given point in time, rather than relying on a static historical snapshot. This idea was applied to the problems of tracking quantiles and heavy hitters (frequent items) of the observed item distribution. Additional aspects considered included the fact that the sites should send a compact approximate description of their distribution, rather than a full description; sites might have limited space and so can only maintain a small space summary of their input; and that the network topology may be a multi-level hierarchy, so communication no longer has uniform cost.

The prediction idea was extended to more complex functions such as join sizes (or equivalently, the inner product of large vectors) by Cormode and Garofalakis [7]. Here, the idea was to operate predominantly in "sketch space": a random linear transformation of the input down to low-dimensional vectors. Due to the linearity of the sketch transformation, a prediction based on linear or quadratic growth in different dimensions could be captured by a sketch of the (first order or second order) difference between past values, which in turn is the appropriate difference of sketches. Violations of predictions can be detected by testing the deviation between the actual and predicted sketches.

Huang *et al.* worked on tracking spectral properties of distributed data, where each time step adds a new row to a matrix of observations from different observers. The quantity of interest to be monitored here was the residual energy of the signal after removing the projections along the principal components [25]. Other work studied anomaly detection, where an anomaly occurs when the number of events exceeds an expected rate, over any historical window [24]. This can be seen as a variant of the countdown problem where there is a background process which depletes the number of observed events at a uniform rate. A different approach to this problem is due to Jain *et al.* [28], who consider optimizing slack allocation within a hierarchical network topology, and robustness within a dynamic network (nodes dying, or new nodes joining).

Many other specific functions have been studied in this model, including monitoring the cardinality of set expressions [16] tracking the (large) number of distinct elements observed [13], tracking clusterings of points in a metric space [14], sparse approximation of signals [34], and conditional entropy [1].

The continuous distributed model has also been studied from a more theoretical perspective. [10] revisited various fundamental functions: F_0 (number of distinct elements), F_1 (count/countdown) and F_2 (self-join size/Euclidean norm), and gave the first or improved worst-case bounds for these problems, as well as the first lower bounds. An alternate cost model was proposed by Yi and Zhang [39]: in the multi-dimensional online-tracking model, we seek to bound the *competitive ratio* of the communication. That is, the ratio between an online protocol and the best offline protocol which is allowed to see the whole input upfront. This is model proposed for a single site. We can weakly generalize these results to the case of multiple sites, by arguing that each site may be competitive in informing the coordinator of its value (for example, for tracking a count). But showing a protocol which is competitive over the global communication remains an open problem.

Yi and Zhang have also proposed improved bounds for tracking quantiles and heavy hitters [40]. Specifically, they show how both problems can be solved with total communication $O(k/\epsilon \log n)$ to provide ϵ-approximate results over streams of total length n. Chan *et al.* study the same problems in the context of time-based sliding windows, where only recent events are counted [5]. Most recently, Cormode and Yi observed that the 'two window' approach used for sampling can also be applied to simplify the problems, and achieve improved bounds [15].

7. CONCLUDING REMARKS

This survey has aimed to give a flavour of the line of work in continuous distributed monitoring, by highlighting a few problems and approaches, and identifying the breadth of other related work. For a different perspective (with, admittedly, a similar authorial tone), there are a couple of surveys and tutorials from about five years ago [8, 6].

Since those prior surveys, there has certainly been progress made in this area. In particular, additional problems have been studied; more robust bounds—both upper and lower bounds—have been proved on the communication costs, as well as other costs such as space; variant models have been introduced, such as sliding windows and the online-tracking model; and a broader set of researchers have worked on related problems (see, for example, the EU-supported LIFT project, `http://lift-eu.org/`).

At the same time, many questions posed previously have yet to be fully addressed. Next, I outline two quite different directions for this area that seem quite open, and capable of generating interesting and important results.

Systems for Continuous Distributed Monitoring. While there has been considerable progress on developing protocols and techniques for continuous distributed monitoring, these have yet to translate to practical implementations. There have been several prototype studies of protocols in the works introducing them, which have indicated the potential for orders of magnitude savings in the amount of communication incurred. However, as far as I am aware, these have not translated to widespread adoption of these ideas, or incorporation into standard protocols. Moreover, these trials have tended to be in simulated environments on recorded data streams, rather than "live" tests. Possibly the lack of uptake of these methods is due to a lack of urgency for the problems considered. While orders of magnitude saving may be possible, if the overhead of centralization, or the delay of polling is considered acceptable, then there is no need to implement a more complex monitoring solution. In other words, attention needs to focus on settings where the naive solutions do place an intolerable burden on the network. One interesting example arises in the context on Massively Multi-player Online Role Playing Games (MMORPGs). Here, it has been argued that the number of different quantities that need to be monitored simultaneously in a distributed setting (such as the health scores of players and enemies) is so large that a more efficient solution is needed [23].

In terms of open problems, the basic challenge is to first develop libraries of code, and then evolve these into general purpose systems, so that they can be easily adopted by programmers and data owners. That is, it should be possible for someone who wants to incorporate continuous distributed monitoring into their system (say, into their MMORPG) to just plug in code "off the shelf", rather than reimplementing protocols themselves. Or, there should exist systems for distributed monitoring which are as accessible and general purpose as traditional centralized database management systems. Both directions require considerable work to achieve from the current state of the art. In particular, what classes of functions should such tools support? Should they be based on a collection of "typical" functions (such as the countdown and entropy monitoring problems), or adopt the more generic geometric monitoring approach? Should there be a general purpose, high level query language for flexibly specifying monitoring problems?

A Deeper Theory of Continuous Distributed Monitoring. In recent years, there have been theoretical results shown for problems in continuous distributed monitoring. For the first time, strong upper bounds on the amount of communication of certain protocols have been shown, when previously only heuristic results were known. In some cases these are complemented by lower bounds, sometimes matching or almost matching.

The upper bounds follow by quantifying the worst case amount of communication that can be forced by any input. Typically, this is in a setting when the total number of events, n, is "growing" in some sense, and we can bound the communication in terms of a sublinear function of n — see the results above for countdown and entropy, which essentially bound the cost in terms of the logarithm of n. For more general cases, where events may represent arrivals and departures, a lot of communication may be forced if the aggregate number of active items at any time remains low. Instead, we might hope to follow Yi and Zhang, and prove a competitive ratio [39].

The lower bounds proved thus far tend to be based on extensions of existing communication complexity ideas [31]—in essence, they count the number of possible configurations, and argue that there must be sufficient communication to allow the sites or the coordinator to distinguish between different outcomes. While this approach has proved effective in some settings, it fails to fully capture the constraints of the model. It seems that a richer notion of continuous communication complexity is called for.

There are several powerful results in the literature which could potentially be extended. The famed Slepian-Wolf theorem [38] captures the case where there are correlated sources. They can encode their outputs to allow correct decoding, while using a total amount of communication proportional to the joint entropy. We can cast this "distributed source coding" as a special case of continuous distributed monitoring, where the target is the streams. Then, what we seek is a generalization of Slepian-Wolf, that will capture a function of multiple inputs, rather than just the identity function. This could also take advantage of correlations over time as well as space.

Acknowledgements

Thanks to S. Muthukrishnan and Ke Yi for comments and suggestions. Thanks also to Luigi Laura for encouraging me to finish this survey on time.

8. REFERENCES

[1] C. Arackaparambil, S. Bratus, J. Brody, and A. Shubina. Distributed monitoring of conditional entropy for anomaly detection in streams. In *IPDPS Workshops*, 2010.

[2] C. Arackaparambil, J. Brody, and A. Chakrabarti. Functional monitoring without monotonicity. In *Intl. Colloq. Automata, Languages and Programming (ICALP)*, 2009.

[3] B. Babcock and C. Olston. Distributed top-k monitoring. In *ACM SIGMOD Intl. Conf. Management of Data*, 2003.

[4] V. Braverman, R. Ostrovsky, and C. Zaniolo. Optimal sampling from sliding windows. In *ACM Principles of Database Systems*, 2009.

[5] H.-L. Chan, T.-W. Lam, L.-K. Lee, and H.-F. Ting. Continuous monitoring of distributed data streams over a time-based sliding window. In *Symp. Theoretical Aspects of Computer Science*, 2010.

[6] G. Cormode and M. Garofalakis. Efficient strategies for continuous distributed tracking tasks. *IEEE Data Engineering Bulletin*, 28(1):33–39, March 2005.

[7] G. Cormode and M. Garofalakis. Sketching streams through the net: Distributed approximate query tracking. In *Intl. Conf. Very Large Data Bases*, 2005.

[8] G. Cormode and M. Garofalakis. Streaming in a connected world: Querying and tracking distributed data streams. In *ACM SIGMOD Intl. Conf. Management of Data*, `http://dimacs.rutgers.edu/~graham/pubs/html/CormodeGarofalakis06VLDB.html`, 2007.

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

[10] G. Cormode, S. Muthukrishnan, and K. Yi. Algorithms for distributed, functional monitoring. In *ACM-SIAM Symp. Discrete Algorithms*, 2008.

[11] G. Cormode, S. Muthukrishnan, K. Yi, and Q. Zhang. Optimal sampling from distributed streams. In *ACM Principles of Database Systems*, 2010.

[12] G. Cormode, S. Muthukrishnan, K. Yi, and Q. Zhang. Optimal sampling from distributed streams. *J. ACM*, 2011.

[13] G. Cormode, S. Muthukrishnan, and W. Zhuang. What's different: Distributed, continuous monitoring of duplicate resilient aggregates on data streams. In *IEEE Intl. Conf. Data Engineering*, 2006.

[14] G. Cormode, S. Muthukrishnan, and W. Zhuang. Conquering the divide: Continuous clustering of distributed data streams. In *IEEE Intl. Conf. Data Engineering*, 2007.

[15] G. Cormode and K. Yi. Tracking distributed aggregates over time-based sliding windows. In *ACM Conf. Principles of Distributed Computing*, 2011.

[16] A. Das, S. Ganguly, M. Garofalakis, and R. Rastogi. Distributed set-expression cardinality estimation. In *Intl. Conf. Very Large Data Bases*, 2004.

[17] M. Dilman and D. Raz. Efficient reactive monitoring. In *IEEE INFOCOMM*, 2001.

[18] J. Feldman, S. Muthukrishnan, A. Sidiropoulos, C. Stein, and Z. Svitkina. On distributing symmetric streaming computations. In *ACM-SIAM Symp. Discrete Algorithms*, 2008.

[19] S. Ganguly and B. Lakshminath. Estimating entropy over data streams. In *European Symp. Algorithms (ESA)*, 2006.

[20] P. Gibbons and S. Tirthapura. Estimating simple functions on the union of data streams. In *ACM Symp. Parallel Algorithms and Architectures*, 2001.

[21] P. Gibbons and S. Tirthapura. Distributed streams algorithms for sliding windows. In *ACM Parallel Algorithms and Architectures*, 2002.

[22] N. J. A. Harvey, J. Nelson, and K. Onak. Sketching and streaming entropy via approximation theory. In *IEEE Foundations of Computer Science*, 2008.

[23] K. Heffner and G. Malecha. Design and implementation of generalized functional monitoring. `http://www.people.fas.harvard.edu/~gmalecha/proj/funkymon.pdf`, 2009.

[24] L. Huang, M. N. Garofalakis, A. D. Joseph, and N. Taft. Communication-efficient tracking of distributed cumulative triggers. In *ICDCS*, 2007.

[25] L. Huang, X. Nguyen, M. Garofalakis, J. Hellerstein, A. D. Joseph, M. Jordan, and N. Taft. Communication-efficient online detection of network-wide anomalies. In *IEEE INFOCOMM*, 2007.

[26] P. Indyk and R. Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *ACM Symp. Theory of Computing*, pages 604–613, 1998.

[27] A. Jain, J. Hellerstein, S. Ratnasamy, and D. Wetherall. A wakeup call for internet monitoring systems: The case for distributed triggers. In *Workshop on Hot Topics in Networks (Hotnets)*, 2004.

[28] N. Jain, M. Dahlin, Y. Zhang, D. Kit, P. Mahajan, and P. Yalagandula. STAR: Self-tuning aggregation for scalable monitoring. In *Intl. Conf. Very Large Data Bases*, 2007.

[29] W. Johnson and J. Lindenstrauss. Extensions of Lipshitz mapping into Hilbert space. *Contemporary Mathematics*, 26:189–206, 1984.

[30] R. Keralapura, G. Cormode, and J. Ramamirtham. Communication-efficient distributed monitoring of thresholded counts. In *ACM SIGMOD Intl. Conf. Management of Data*, 2006.

[31] E. Kushilevitz and N. Nisan. *Communication Complexity*. Cambridge University Press, 1997.

[32] A. Lakhina, M. Crovella, and C. Diot. Mining anomalies using traffic feature distributions. In *ACM SIGCOMM*, 2005.

[33] S. Muthukrishnan. Data streams: Algorithms and applications. In *ACM-SIAM Symp. Discrete Algorithms*, 2003.

[34] S. Muthukrishnan. Some algorithmic problems and results in compressed sensing. In *Allerton Conf.*, 2006.

[35] C. Olston, J. Jiang, and J. Widom. Adaptive filters for continuous queries over distributed data streams. In *ACM SIGMOD Intl. Conf. Management of Data*, 2003.

[36] I. Sharfman, A. Schuster, and D. Keren. A geometric approach to monitoring threshold functions over distributed data streams. In *ACM SIGMOD Intl. Conf. Management of Data*, 2006.

[37] I. Sharfman, A. Schuster, and D. Keren. Shape sensitive geometric monitoring. In *ACM Principles of Database Systems*, 2008.

[38] D. Slepian and J. Wolf. Noiseless coding of correlated information sources. *IEEE Transactions on on Information Theory*, 19:471–480, 1973.

[39] K. Yi and Q. Zhang. Multi-dimensional online tracking. In *ACM-SIAM Symp. Discrete Algorithms*, 2009.

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