Communication-Efficient Distributed Monitoring of Thresholded Counts

Ram Keralapura *
ECE Department
University of California, Davis
rkeralapura@ucdavis.edu

Graham Cormode
Bell Laboratories
Murray Hill, NJ
cormode@lucent.com

Jai Ramamirtham
Bell Laboratories
Bangalore, India
jai@lucent.com

ABSTRACT

Monitoring is an issue of primary concern in current and next generation networked systems. For example, the objective of sensor networks is to monitor their surroundings for a variety of different applications like atmospheric conditions, wildlife behavior, and troop movements among others. Similarly, monitoring in data networks is critical not only for accounting and management, but also for detecting anomalies and attacks. Such monitoring applications are inherently continuous and distributed, and must be designed to minimize the communication overhead that they introduce. In this context we introduce and study a fundamental class of problems called "thresholded counts" where we must return the *aggregate* frequency count of an event that is continuously monitored by distributed nodes with a user-specified accuracy whenever the actual count exceeds a given threshold value.

In this paper we propose to address the problem of thresholded counts by setting local thresholds at each monitoring node and initiating communication only when the locally observed data exceeds these local thresholds. We explore algorithms in two categories: *static thresholds* and *adaptive thresholds*. In the static case, we consider thresholds based on a linear combination of two alternate strategies, and show that there exists an optimal blend of the two strategies that results in minimum communication overhead. We further show that this optimal blend can be found using a steepest descent search. In the adaptive case, we propose algorithms that adjust the local thresholds based on the observed distributions of updated information in the distributed monitoring system. We use extensive simulations not only to verify the accuracy of our algorithms and validate our theoretical results, but also to evaluate the performance of the two approaches. We find that both approaches yield significant savings over the naive approach of performing processing at a centralized location.

1. INTRODUCTION

Many emerging systems are fundamentally distributed in nature. The current and next generation of networks are large-scale, and widespread. Within this distributed networked systems, a principal concern is *monitoring*: either monitoring the environment surrounding each of the network nodes, or monitoring the behavior of the network itself. Two prototypical applications are in: (i) Sensor networks, whose *raison d'etre* is for monitoring and collating information on atmospheric conditions, wildlife behavior, and troop movements in military applications among others, and (ii) Network traffic monitoring in (wired or wireless) data networks, for traffic management, routing optimization, and anomaly and attack detection.

Over the past few years, the defining characteristics of these applications have been identified to pose new challenges that are not answered by traditional data management systems. The challenges in monitoring arise mainly because the systems are fundamentally:

- *Continuous:* Unlike the traditional database "on-demand" view of the world, where queries are posed in SQL and an answer returned to the user, queries in these monitoring situations are typically long running queries over the data, which must continuously run and return answers as and when they are found.
- *Distributed:* The data required to answer the monitoring queries is distributed throughout the network. Typically, a query requires information to be collated and aggregated from many, if not all, nodes in the network.
- *Resource-constrained:* Efficiency of operation is absolutely vital in the distributed monitoring world. In sensor networks, we must ensure that the life of the network is extended as long as possible by minimizing the energy drain of running the monitoring protocol; in data networks, we must ensure that the protocol does not hinder the principal operation of the network, allowing the delivery of messages unencumbered by the monitoring overhead. These concerns manifest themselves principally as a requirement to minimize (to the extent possible) the *communication cost* of the monitoring protocols: communication is the principal energy drain for a sensor, and excess communication in a data network reduces the capacity for normal operation. As a secondary concern, computation and memory usage should also be minimal for efficient execution of the monitoring.

Thresholded Counts. Within this framework of continuous, distributed, and resource-constrained systems, there are many possible types of monitoring queries that can be posed. Prior work has looked at particular query types such as top-k monitoring [3], set expression cardinality [12], and holistic aggregates such as quantiles [8]. But many queries rely at heart on monitoring sums or counts of values, in combination with thresholds (lower bounds). Consider the following queries from a variety of different domains:

*This work was done when Ram Keralapura was an intern in Bell Labs Research, India

SIGMOD 2006, June 27–29, 2006, Chicago, Illinois, USA.

- (from [22]) Report when at least 40 soldiers have crossed a specified boundary.
- Raise an alert when the total number of cars on the highway exceeds 4000 and report the number of vehicles detected.
- Which species have more than 50 members within a certain region for more than an hour?
- Identify all destinations that receive more than 2GB of traffic from the monitored network in a day, and report their transfer totals
- (Query Q_1 of [21]) Monitor the volume of remote login (telnet, ssh, ftp etc.) request received by hosts within the organization that originate from the external hosts.
- Which users within the monitored network receive more than 1000 different connections?

In each case there are two parts to the query: a request for a sum or count of a particular quantity (vehicles, animals, network connections etc.), and a minimum threshold, or trigger, for when we need to know this information. Such thresholds are vital in focusing the information returned to the user, and in reducing the monitoring burden on the network. In almost every application involving measuring quantities like these, it is only important to know the quantities when they exceed a specified level. Small counts (of remote logins, human activity, network traffic) are prevalent and can be ignored to reduce the reporting burden on the monitoring system. But in all the above situations we can define a threshold, such that it is critical to know when this threshold has been crossed. Our focus in this paper is on designing protocols and algorithms to monitor such sums and counts with thresholds. In the extreme case, these thresholds can be trivial (i.e. zero or one), but in all the scenarios we have outlined, there exists a non-trivial threshold which can be used to reduce the communication cost. In general, the thresholds can be specified either as part of the query, or learned by the system in response to the observed data. For this work, without loss of generality we assume that the threshold is fixed *a priori* and focus on answering queries for thresholded counts given such a threshold; dynamic thresholds can be accommodated, but we do not discuss them here.

The second component of these types of queries is to return a count of a particular set of values. Here, one can make the observation that an application rarely needs to know the exact count so long as the answer is given with reasonable precision: it is not necessary to know whether the number of cars on the highway is 4237 or 4251, if either answer is accurate to within 1%. So instead of demanding exact results, we can explore the tradeoff between accuracy and communication: clearly, allowing larger uncertainties about counts allows monitoring sites to be more conservative about when they send their updates to a central monitor. This benefit becomes clear in our experiments, which show significant savings as the allowable uncertainty increases.

Our Contributions. We address the problem of continuously monitoring thresholded counts in a distributed environment. Our contributions are as follows:

1. We introduce and formalize the thresholded counting problem, which is fundamental to several distributed monitoring scenarios. We give guaranteed solutions to the problem based on monitors comparing their local counts to local thresholds, and postponing communication until these thresholds are violated. We consider two approaches, depending on whether the thresholds are determined statically in advance, or can be allocated adaptively as the distribution of updated information is observed.

2. In the static case, we show two different fundamental techniques for setting the local thresholds. We introduce a blended approach, based on a linear combination of the two fundamental methods while retaining the correctness guarantee. We give a careful and detailed analysis of the optimal setting of this blend which depends only on coarse properties of the total count.

3. In the adaptive case, we introduce a variety of increasingly sophisticated algorithms that attempt to capture the observed distribution of count updates, and hence reduce the overall number of messages sent within the system.

4. We show that our static and adaptive algorithms can be easily extended to include negative updates, sliding windows, approximate counts, and time-dependent threshold values.

5. We conduct a thorough and detailed set of experiments to verify the efficacy of our methods for giving a low cost monitoring scheme for thresholded queries. We compare to applications of prior work on a variety of real and synthetic data, and show that there are significant savings for using our methods.

Outline. In Section 2, we review prior work in continuous, distributed monitoring. We formally define the problem of thresholded counts in Section 3. We propose algorithms which set static thresholds in Section 4, and adaptive thresholds in Section 5. We discuss some extensions to our current work in Section 6. We present results from our experimental evaluation in Section 7 and conclude in Section 8.

2. RELATED WORK

Our work is related to the design and construction of systems to handle large streams of data. There are several general purpose "Data Stream Management Systems" (DSMSs), such as Aurora [1], Stream [2], and Telegraph [4], and special purpose systems (such as for network data streams) including Gigascope [11] and Tribeca [23]. However, our problem is inherently distributed, whereas these systems are primarily focused on centralized monitoring. Recently, distributed data stream management systems have been proposed such as Borealis [14] and Medusa [24, 5]. The main focus of these systems is how to effectively distribute the work of the system, and hence balance the load; it is therefore assumed possible to redirect data streams. In our model, we explicitly rule out the possibility of sending all observed data to a different processor: in resource constrained situations, our only possibility is to process the stream of data at the monitor where it is observed. (In our experiments, we are able to improve on the cost of sending the whole data stream by three orders of magnitude by processing at the point of observation).

The model of continuous distributed computation has been adopted by a series of papers over recent years. Within this context, a number of different problems have been addressed, including monitoring the (exact) top-k set [3], evaluating set expressions [12], and quantiles and heavy hitters [8, 16]. Techniques have been proposed which allow the continuous, distributed monitoring of a variety of queries based on using sketches [7] (from which join-sizes and many other functions can be estimated accurately) and based on techniques that are resilient to duplication of information [10].

The problem we consider, of maintaining accurate counts above a threshold, is not covered by any of these approaches. We now discuss previous work that is closer to this problem. Various works have considered accurate maintenance of sums and counts in a distributed sensor network setting, but these are typically for duplicate-resilient "on-demand" computations, rather than continuous [19, 17, 6]. The "heavy hitters" problem is to find all items whose count is above some fraction ϕ of the count of all items (e.g. users in a wired network consuming more than 1% of the total bandwidth). While semantically related to our threshold based problem, con-

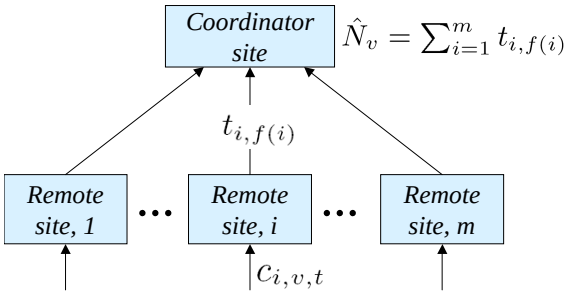


Figure 1: Distributed Monitoring Architecture

tinuous monitoring of the heavy hitters is a somewhat easier problem, since there can be at most $1/\phi$ heavy hitters at any one time, whereas there can be an unbounded number of monitored counts exceeding their thresholds. The permitted error for Heavy Hitters also scales with the sum of the counts, rather than the count of the individual item, as we demand in our formal definition of this problem (see next section). A continuous, distributed algorithm for heavy hitters follows as a special case of quantile monitoring [8], but this cannot be applied to our problem since the guarantees are insufficient to meet our requirements.

More closely related to our problem is the work of Jain et al [15], which considers triggers. The authors make a compelling argument for continuous distributed monitoring of trigger conditions based on local counts f_i, and suggest some potential approaches. The method outlined relates to triggers of the form $\sum_i f_i > C$, i.e. the trigger should be raised if the total of the f_is at different sites exceeds the threshold C. This is clearly a simplified version of our problem. The algorithms given are randomized, and in expectation give errors of constant factors times C, which are much larger than the δ guarantees we provide.

Olston *et al* [21] propose a filter based approach to maintaining accurate counts. There, the central site wishes to know each sum $F_j = \sum_i f_{i,j}$ with uncertainty at most δ_j for a fixed δ_j. The approach is to allocate "filters" (allowable ranges) at each site, and when values are observed that lie within these filters, no action is taken; if they fall outside the range, then they are passed up to the central site. To adapt to the observed distribution, filters are periodically resized based on how long the filter has been valid. This approach can be modified to apply to our problem, and we compare against this algorithm in our experimental section.

Most recently, an approach to monitoring general functions has been proposed by Sharfman et al [25]. Applied to our problem, their technique gives algorithms of a similar nature to those we propose, however our algorithms are designed to take advantage of specific features of our problem. It will be of interest to compare the two in future.

3. PROBLEM DEFINITION AND APPROACH

We define the problem of efficiently maintaining approximate counts in a distributed scenario in this section and describe our approach.

3.1 System Architecture

We consider a distributed monitoring system as shown in Figure 1. The system consists of m *remote sites* and a central *coordinator site*. The remote sites observe a continuous stream of *updates*, which taken together define a distribution of values. The remote sites can communicate with the coordinator in order to ensure that the coordinator is able to accurately answer queries over the *union* of the update streams. In general, the remote sites can communicate amongst themselves as well as with the coordinator; however, in line with previous work, we only consider protocols that have (pairwise) communications between the coordinator and remote sites. This is the model that is adopted in most prior work on continuous, distributed monitoring problems [3, 21, 12, 8, 7][1].

Each site $i \in \{1 \ldots m\}$ monitors a set of k values $N_{v,i}$, $v \in \{1 \ldots k\}$, which are defined incrementally. We model each stream of updates observed at the remote site i as a sequence of tuples $\langle i, v, t, c_{i,v,t} \rangle$. This is interpreted as an update of $c_{i,v,t}$ to N_v in site i at time t. We assume that updates are ordered by timestamp, and site i only sees updates to itself[2]. Then $N_{v,i}(t)$, the value of the count in site i at time t, is defined as $N_{v,i}(t) = \sum_{t' < t} c_{i,v,t'}$. The global count, $N_v(t)$, is defined as $N_v(t) = \sum_{i \in \{1 \ldots m\}} N_{v,i}(t)$. Our goal is to monitor the value of each $N_v(t)$ within specified accuracy bounds. Since we are interested only in the "current" value of counts, in the rest of the paper we drop reference to t and use N_v and $N_{v,i}$ to represent the global and local counts.

This model accurately captures the scenarios we described in the introduction. For example, in network traffic monitoring, each update might correspond to the observation of a packet at a remote site (or monitor). In this context, t is the current time, i is the identifier of the monitor, and v and $c_{i,v,t}$ are properties of the packet, such as destination IP address and size of packet, respectively. Based on the inputs from all the remote sites, the coordinator tracks the aggregate traffic to various destinations and raises an alarm when the total traffic becomes high, indicative of an unusual activity (like a DDoS attack). Monitoring in sensor networks can also be mapped onto this model in a natural way.

In general, the updates, $c_{i,v,t}$, can be *negative* (corresponding to a decrease in $N_{v,i}$, such as temperature updates in sensor networks) or *fractional* (like rainfall measurements). All our methods will handle such settings, but for clarity we focus on the case where $c_{i,v,t}$s are positive integers, and postpone discussion of negative updates to Section 6.

3.2 The Thresholded Count Problem

Our focus is on monitoring the N_v at the central coordinator. Since N_v is defined by updates to remote sites, if we require to know N_v exactly, then we must eagerly send every update from a remote site to the coordinator as soon as it is observed. This ensures accurate values at the coordinator at all times, but comes with huge communication overhead. As observed in Section 1, such fine accuracy is not needed in practice. Another possibility is for the remote sites to send their counts *periodically* to the coordinator site. This reduces the communication burden, but still has some issues in practice: updates in real systems are typically bursty, i.e., counts change rapidly in some time periods while it may hardly change in others. The former results in inaccurate values at the coordinator, while the latter results in unnecessary communications. In this paper, we define the problem of continuously monitoring thresholded counts, which ensures that the coordinator always has an accurate count with minimal delay[3].

DEFINITION 1. *Given a threshold T_v and an error guarantee δ_v, the δ_v-*deficient thresholded count*, $\hat{N}_v$ satisfies the following*

[1] As in the cited prior work, we concentrate on the key algorithmic challenges in designing effective protocols for this setting, and hence we do not consider issues of message loss. Instead, we assume that there are sufficient mechanisms in place such as message acknowledgments, retransmissions, and timestamping to ensure that the correct behavior is observed.

[2] This is not a strong assumption; typically, the site observes only the pair $v, c_{i,v,t}$, and supplies t and i itself.

[3] Since network delay from remote sites to coordinator is unavoidable, every tracking algorithm must incur some minimum delay.

properties

- $0 \leq \hat{N}_v < T_v$ *when* $N_v < T_v$
- $N_v(1-\delta_v) < \hat{N}_v < N_v$ *when* $N_v \geq T_v$

Where it is clear from context, we will drop the qualification v and refer to N, T, δ. Note that this definition is distinct from the Heavy Hitter definition in data streams [18], which requires an *additive* error that scales as the sum of all monitored counts; instead, we have a much more demanding requirement to monitor all counts with *relative* error on each count, above the threshold. Without a threshold, T, the communication overhead is high to begin with, as low counts require every update to be pushed to the coordinator in order to maintain the error guarantee δ. Since low counts are typically uninteresting for monitoring applications, by supressing communication for these counts, the overhead of the monitoring can be kept low.

The value of the threshold depends on individual applications. For applications in network monitoring that track anomalous behavior, like DDoS attacks, the value of the threshold can be high, while applications like traffic accounting [13] that count the traffic sent by hosts or networks beyond a certain initial minimum can use a lower threshold value.

3.3 Our Approach

Our basic approach is to set local thresholds at each remote site such that the current count is bounded by the local threshold. When a local threshold in a remote site is violated, the remote site will communicate this to the coordinator and sets a new threshold. The ith remote site maintains local thresholds, $t_{i,j}, j = 0, 1, \ldots$, and ensures that $t_{i,f(i)} \leq N_{v,i} < t_{i,f(i)+1}$ for some threshold $f(i)$ that is known to the coordinator. If the ith remote site's count violates this condition, it sends an update to the coordinator with a new $f'(i)$ and $t_{i,f'(i)}$ such that $t_{i,f'(i)} \leq N_{v,i} < t_{i,f'(i)+1}$ for the current value of $N_{v,i}$. The coordinator can use the set of $t_{i,f(i)}$s to estimate any global count as $\hat{N}_v = \sum_{i=1}^{m} t_{i,f(i)}$.

Note that while the count at a remote site obeys the $t_{i,f(i)} \leq N_{v,i} < t_{i,f(i)+1}$ bounds, the remote site does not send any updates until the count is outside these bounds. Until the coordinator receives the next threshold update the actual count can lie anywhere between the two threshold values. Hence, the maximum error contributed to the global count error by remote site i is given by $t_{i,f(i)+1} - t_{i,f(i)}$. An algorithm that tracks counts must ensure that the error is within the δ-deficient requirement when the count is greater than the specified threshold. Formally, we must ensure that $0 < \sum_{i \in \{1 \ldots m\}} t_{i,f(i)+1} - t_{i,f(i)} < \delta N_v$ when $N_v > T$. Thus, adjacent thresholds need to be chosen to be close enough to satisfy this requirement. The total number of updates sent from remote sites to the coordinator corresponds to the number of threshold boundaries crossed at the remote sites. This means we want to set the local thresholds as far apart as possible to minimize the communication overhead.

Algorithms that track the δ-deficient thresholded count of an item need to balance the error requirement with minimal communication overhead. We consider two fundamental categories for setting threshold: *static thresholding*, and *adaptive thresholding*. In static thresholding methods, each remote site is assigned a predetermined set of thresholds that do not change over the entire course of tracking the count. It simply tracks between which pair of thresholds its count currently lies, and informs the coordinator when this changes. In the adaptive case, when old thresholds are violated, new thresholds at the remote sites are chosen by the central site according to the observed conditions, to dynamically reduce the communication overhead.

While the adaptive thresholding methods can be expected to perform better than the static methods, the static methods are desirable when the capabilities of the remote sites and the coordinator are limited. The adaptive thresholding places additional processing overhead and additional functional requirements on the remote sites and the coordinator. The coordinator needs to recompute new thresholds and export them to the remote sites, in addition to processing updates from the remote sites to maintain the count. In certain cases, like sensor networks or high speed routers, this additional processing overhead may be too expensive to accommodate. A further practical issue with using adaptive thresholding is that the system has to be more resilient to network delays. Specifically, the coordinator may need to collect current values from sites, and send out many new thresholds, which incurs appreciable delay where the current counts may be outdated. The static thresholding scheme does not have this problem because the communication is performed from the remote site to the coordinator only. Thus the choice of adaptive or static thresholds will depend not only on their relative cost (which we analyze in detail in subsequent sections), but also on the underlying network properties and performance.

4. STATIC THRESHOLDS

We now describe the static thresholding scheme to maintain the δ-deficient thresholded counts. In these schemes, the threshold values in the remote sites are predetermined and do not change over the period of tracking. We present three such threshold assignment regimes to determine the local threshold values at the remote sites and discuss their complexity in terms of communication overhead. In this work we consider all the remote sites to be symmetric and hence use the same set of static threshold values. Our focus is on determining the local threshold values in the remote sites for a given value of δ and T. The static threshold assignment problem can be formally stated as:

DEFINITION 2. *Given m remote sites, a global threshold T, an error guarantee δ, and $f(i)$ (the current threshold level at site i), determine threshold values, $t_j, j = [0, \infty)$, such that the following constraints are satisfied*

- $\forall j \geq 0 : t_{j+1} > t_j$, *and* $t_0 = 0$
- $\forall f \in \mathbb{N}^m : \sum_{i=1}^{m} t_{f(i)+1} - t_{f(i)} \leq \delta \sum_{i=1}^{m} t_{f(i)}$, *when* $\sum_{i=1}^{m} t_{f(i)+1} \geq T$

The first constraint ensures that the threshold values are increasing. The second constraint captures the error requirement of the thresholded count problem. The maximum error in the ith remote site when $f(i)$ is the threshold in force at site i is $t_{f(i)+1} - t_{f(i)}$. Thus, the second constraint states that the total error in the count at the coordinator must satisfy the thresholded error guarantee for all possible threshold values at the remote sites.

4.1 Uniform Threshold Assignment

The simplest solution is to keep the maximum global error level at δT at all times even when the global count, N, is much greater than T. This can be accomplished by setting the threshold levels in each monitor as $t_j = \frac{j\delta T}{m}$. When $N \geq T$, we have the total error $\sum_{i=1}^{m} t_{f(i)+1} - t_{f(i)} \leq \delta T \leq \delta N$, thus satisfying the δ-deficient thresholded count constraints. If the global count is N, the maximum number of updates sent to the coordinator is given by $\lfloor \frac{mN}{\delta T} \rfloor$. This simplicity comes at a price. The method works well for counts that are small (below T or only above T by a small amount), since the threshold gaps are relatively large. But as N increases above T, the cost scales linearly with N, as the overly tight error guarantee is maintained. In summary,

LEMMA 1. *The total number of messages from all remote sites to the coordinator with uniform threshold assignments is* $O(\frac{mN}{\delta T})$.

4.2 Proportional Threshold Assignment

A more scalable solution is to assign threshold values proportional to the local count at the remote site. The thresholds at the remote sites are assigned as $t_j = (1+\delta)t_{j-1}$ and $t_0 = 0, t_1 = 1$. If the threshold value reported by site i to the coordinator is $t_{f(i)}$, the maximum possible error from the site is $t_{f(i)+1} - t_{f(i)} = \delta t_{f(i)}$. The maximum error at the coordinator is:

$$\sum_{i=1}^{m} t_{f(i)+1} - t_{f(i)} = \sum_{i=1}^{m} \delta t_{f(i)} \leq \delta \sum_{i=1}^{m} N_i = \delta N$$

where N is the global count. This assignment satisfies the error requirement even when the global count is less than the threshold T.

LEMMA 2. *The total number of messages from all remote sites to the coordinator with proportional threshold assignments is* $O(\frac{m}{\delta} \log \frac{N}{m})$.

PROOF. If $t_{f(i)} \leq N_i < t_{f(i)+1}$, the number of updates from remote site i is given by $f(i)$. Since $t_{f(i)} = (1+\delta)^{f(i)-1}$ we get

$$f(i) = 1 + \frac{\log(t_{f(i)})}{\log(1+\delta)} \leq 1 + \frac{\log(N_i)}{\log(1+\delta)}$$

The total number of messages is bounded by

$$\sum_{i=1}^{m} f(i) \leq m + \sum_{i=1}^{m} \frac{\log(N_i)}{\log(1+\delta)} \leq m + m\frac{\log\left(\frac{N}{m}\right)}{\log(1+\delta)}$$

We use the fact that $\sum_{i=1}^{m} N_i = N, \sum_{i=1}^{m} \log N_i = \log\left(\prod_{i=1}^{m} N_i\right)$, and $\prod_{i=1}^{m} N_i$ is maximized when $\forall i : N_i = \frac{N}{m}$. Since for $\delta < 1$, $\log^{-1}(1+\delta) = O(\frac{1}{\delta})$, the stated bound follows. □

This method of assignment performs well when $N \gg T$. The relative cost of the uniform assignment to the proportional assignment is $O(m/\delta \log(N/m))/O(Nm/(\delta T)) = O(T/N \log(N/m))$. When T is greater than N, the uniform spread assignment performs better, but as N increases above T, the proportional assignment requires fewer communications.

4.3 Blended Threshold Assignment

The main idea of blended threshold assignment is to exploit the best features of the previous two assignments and provide a mechanism to tune the performance for different values of N.

DEFINITION 3. *The blended assignment sets the local threshold values as*

- $t_j = (1+\alpha\delta)t_{j-1} + (1-\alpha)\frac{\delta T}{m}$, *for a parameter* $0 \leq \alpha \leq 1$
- $t_0 = 0$, and *when* $\alpha = 1$, $t_1 = 1$

Note that $\alpha = 0$ corresponds to the uniform assignment while $\alpha = 1$ corresponds to the proportional assignment. Varying the value of α helps in tuning the threshold values to combine uniform and proportional thresholds.

THEOREM 4. *The blended threshold assignment satisfies the* δ*-deficient thresholded error guarantee for all values of* $\alpha \in [0, 1]$.

PROOF. Using the blended threshold assignment, the maximum error in the ith remote site is

$$t_{f(i)+1} - t_{f(i)} = \alpha\delta t_{f(i)} + (1-\alpha)\frac{\delta T}{m}$$

Thus, the total error in the global count is given by

$$\begin{aligned} \sum_{i=1}^{m} t_{f(i)+1} - t_{f(i)} &= \alpha\delta(\sum_{i=1}^{m} t_{f(i)}) + (1-\alpha)\delta T \\ &\leq \alpha\delta N + (1-\alpha)\delta T \\ &\leq \delta N, \text{when } N > T \end{aligned}$$

□

LEMMA 3. *The total number of messages from all remote sites to the coordinator with blended threshold assignments and* $0 < \alpha < 1$ *is* $O(\frac{m}{\alpha\delta} \log(1 + \alpha(\frac{N}{T} - 1)))$

PROOF. The threshold values using the blended assignment for $\alpha \in (0, 1)$ can be written as

$$t_j = \left(\frac{(1+\alpha\delta)^j - 1}{\alpha\delta}\right)\frac{(1-\alpha)\delta T}{m}$$

Thus, the number of updates from remote site i when the threshold value exceeded is $f(i)$ is

$$\begin{aligned} f(i) =& \frac{\log(1 + t_{f(i)}\frac{\alpha m}{(1-\alpha)T})}{\log(1+\alpha\delta)} \\ \leq& \frac{\log(1 + N_i\frac{\alpha m}{(1-\alpha)T})}{\log(1+\alpha\delta)}, \text{since } t_{f(i)} \leq N_i \\ =& \frac{\log(1+\alpha h_i) - \log(1-\alpha)}{\log(1+\alpha\delta)}, \text{where } h_i = \frac{N_i m}{T} - 1 \end{aligned}$$

Note that given $\sum_{i=1}^{m} h_i = \frac{Nm}{T} - m$, the expression $\prod_{i=1}^{m}(1+\alpha h_i)$ is maximized when $\forall i : h_i = h = \frac{N}{T} - 1$. The total number of updates from all remote sites is

$$\begin{aligned} \sum_{i=1}^{m} f(i) =& \frac{\log(\prod_{i=1}^{m}(1+\alpha h_i)) - m\log(1-\alpha)}{\log(1+\alpha\delta)} \quad (1) \\ \leq& m\frac{\log(1+\alpha h) - \log(1-\alpha)}{\log(1+\alpha\delta)} \quad (2) \end{aligned}$$

Upper bounding this expression gives the stated worst case bound. □

Determining the optimum value of α. For small values of $N < T$, $\alpha = 0$ gives us the best possible assignment and for large values of $N \gg T$, $\alpha = 1$ gives us the best assignment. For intermediate values of N, the best value of α can be determined by minimizing the number of updates.

Note that the communication cost in Lemma 3 is dependent on the global count, N. Hence, the optimal value of α depends on N. We advocate two approaches to determining the best value of α. The first approach is to track the global count and determine an expected value of N, N_e after a long period of observation, and use this value to determine the optimal value of α. This can be expected to result in good performance if the actual value of N does not vary a lot from the estimate N_e. A more sophisticated approach is to track the distribution of N over a large set of observations and determine the value of α that minimizes the expected number of update messages over this distribution.

THEOREM 5. *The total number of updates (from Eqn 2),* $K_N = m\frac{\log(1+\alpha h) - \log(1-\alpha)}{\log(1+\alpha\delta)}$ *is a convex function in* α *in the range* $\alpha \in (0, 1)$ *for small values of* δ.

The proof of this theorem is presented in Appendix A.

THEOREM 6. *Given an expected value of* N *or a discrete probability distribution of* N*, we can find a value of* α *that minimizes the number of messages with blended threshold assignments.*

PROOF. First, observe that if $p(N)$ is the probability density function of N, then the expected maximum number of updates given by $K = \sum_{N=1}^{\infty} p(N)K_N$ is a convex function in α in the range $\alpha \in (0,1)$. Since K is a convex combination of convex functions K_N, K is itself convex.

Since K and K_N are convex functions in α in the range $(0,1)$, there exists a single minimum for K and K_N that can be searched by using techniques like gradient descent. The descent algorithm can be used to determine the optimal values of α for both the proposed approaches. In the first approach where we are given the expected value N_e, we determine the optimal value of α by minimizing K_{N_e}. In the second approach where we are given the distribution of N, we can use the descent method to determine the optimal value of α by minimizing the function K as defined above. □

5. ADAPTIVE THRESHOLDS

Unlike the static thresholding scheme, in the adaptive thresholding scheme the thresholds in the monitoring nodes are adaptively set by the coordinator every time there is a threshold violation in a node. In other words, the coordinator not only receives the threshold violations from the monitoring nodes, but also reacts to them by sending new thresholds back. This gives the coordinator more power to set thresholds based on more information about how the distributions at each site are evolving, and hence try to reduce the number of threshold violations. In a general scenario, the coordinator may wait for multiple violations before resetting thresholds, and may reset thresholds for arbitrary subsets of the nodes based on a complete history of past violations. In this work, we react to each threshold violation, and consider only recent history.

5.1 Adaptive Threshold Assignment Problem

In the adaptive thresholding scheme, two levels of thresholds, lower and higher thresholds, are maintained at every node at all times. The lower threshold at node i is denoted by t_{iL}, and the higher threshold by t_{iH}, so that at all times $t_{iL} \leq N_i < t_{iH}$. If these thresholds are violated, i.e. if this condition is no longer true, then the site i contacts the coordinator site with its current count N_i, and it resets its lower threshold $t_{iL} = N_i$. The coordinator estimates the count as the sum of the reported counts from the remote sites, $\hat{N} = \sum_{i=1}^{m} t_{iL}$. The coordinator then updates the t_{iH} for node i, and possibly those of other nodes, to ensure that its count still meets the δ-deficient requirement. To minimize the communication in the system, the coordinator needs to set the upper thresholds to as high a value as possible. Note that the maximum error contributed by site i is $t_{iH} - t_{iL}$.

The problem of setting the upper thresholds of the remote sites by the coordinator can be formally stated as follows.

DEFINITION 7. *Given m remote monitoring nodes, a global threshold T, an error guarantee δ, and a threshold violation from node j, our objective is to determine the higher threshold values, t_{iH}, in all m monitoring nodes such that the number of messages in the monitoring system is kept as low as possible, and the following constraints are satisfied:*

- $\forall 1 \leq i \leq m, t_{iH} > t_{iL}$
- $\sum_{i=1}^{m} t_{iH} - t_{iL} \leq \delta \sum_{i=1}^{m} t_{iL}$, *when* $\sum_{i=1}^{m} t_{iH} \geq T$

Similar to the static thresholding scheme, the first constraint ensures that the higher thresholds are greater than the lower thresholds in all the nodes and the second constraint ensures that the total error in the count at the coordinator must satisfy the thresholded error guarantee.

BASICADAPT(δ, T, m)
1: $t_{iL} \leftarrow 0$; $t_{iH} \leftarrow \frac{T}{m}$; $\hat{N} \leftarrow 0$
2: **loop** {receive update (i, N_i); }
3: **if** $((\hat{N} < (1-\delta)T)$ and $(\hat{N} + N_i - t_{iL} \geq (1-\delta)T))$ **then**
4: poll all sites j for N_j; $t_{jL} \leftarrow N_j$; send $t_{jH} \leftarrow (1+\delta)t_{jL}$;
5: $t_{iL} \leftarrow N_i$; $\hat{N} \leftarrow \sum_{j=1}^{m} N_j$;
6: **if** $(\hat{N} < (1-\delta)T)$ **then**
7: for all j send $t_{jH} \leftarrow t_{jL}\frac{T}{\hat{N}}$ to j;
8: **else**
9: send $t_{iH} \leftarrow t_{iL}(1+\delta)$ to i

Figure 2: Basic Adaptive Thresholding Algorithm

In the static threshold method, the remote sites do not know if the current global count is greater than T or lesser at any time. Hence, the thresholds need to be set to handle both these cases. A key advantage of the adaptive algorithm is that when the global count is less than the threshold, the coordinator can afford to set higher thresholds at the remote sites than in the static algorithm. To illustrate this, define the *slack* in the system as the difference between the threshold and the current estimate of the global count, $S = T - \hat{N}$. The coordinator can now split this slack among the remote sites in any manner and still be able to satisfy the δ-deficient error requirement. Assume that the slack is split among the remote sites as $\eta_i, i = 1, \ldots, m$, such that $\sum_{i=1}^{m} \eta_i \leq S$. Thus, $t_{iH} = t_{iL} + \eta_i$. If the counts at all the remote sites are less than their respective upper thresholds, then the global count must be lesser than the global threshold because $N < \sum_{i=1}^{m} t_{iH} \leq T$. If at any point the global count exceeds the threshold, at least one of the thresholds in the remote sites will be exceeded. This allows the coordinator to determine when the count exceeds the threshold and switch to the case when $N \geq T$ and track the count closely to satisfy the δ-deficient error requirement.

5.2 Basic Adaptive Algorithm

When the total count estimated at the central site, $\hat{N}$, is less than T, a naive approach is to split the slack equally among all the nodes; instead, we propose to split the difference proportional to the current count in the nodes, since nodes that have larger counts than others are likely to grow larger. We set the new $t_{iH} = t_{iL} + \frac{(T - \sum_{j=1}^{m} t_{jL}) t_{iL}}{\sum_{j=1}^{m} t_{iL}} = t_{iL}\frac{T}{\hat{N}}$.

If $\hat{N} \geq (1-\delta)T$, we set $t_{iH} - t_{iL} = \delta t_{iL}$, so that the maximum error in each node is δt_{iL}. This approach is similar to the proportional spread threshold assignment algorithm for static thresholding problem presented in Section 4.2. The adaptive thresholding algorithm is presented in Figure 2. Line 7 performs the proportional split when the counts are small, and line 9 performs the proportional growth when the counts are large. Lines 3 and 4 handle the case when we switch from having $\hat{N} < (1-\delta)T$ to $\hat{N} \geq (1-\delta)T$.

LEMMA 4. *The adaptive thresholding assignment algorithm presented in Figure 2 satisfies the δ-deficient thresholded count constraints.*

PROOF. When $\hat{N} = \sum_{i=1}^{m} t_{iL} < (1-\delta)T$, we have that $N = \sum_{i=1}^{m} N_i < \sum_{i=1}^{m} t_{iH} = \frac{T}{\hat{N}} \sum_{i=1}^{m} t_{iL} = T$, so we know that the total count is less than the threshold T. When $\hat{N} \geq (1-\delta)T$, we know that the total count exceeds $(1-\delta)T$ and the algorithm is similar to the proportional spread threshold assignment algorithm for the static thresholding scheme. In this case, $t_{iH} - t_{iL} = \delta t_{iL}$ and so $\sum_{i=1}^{m}(t_{iH} - t_{iL}) = \delta\hat{N} \leq \delta N$, as required. □

MODIFIEDADAPT(δ, T, m)

1: $t_{iL} \leftarrow 0$; $t_{iH} \leftarrow \frac{T}{m}$; $R \leftarrow \emptyset$; $\hat{N} = 0$;
2: **loop** {receive update (i, N_i); }
3: **if** $(\hat{N} < (1-\delta)T)$ and $(\hat{N} - t_{iL} + N_i \geq (1-\delta)T)$ **then**
4: poll all sites j for N_j; $t_{jL} \leftarrow N_j$; send $t_{jH} \leftarrow (1+\delta)t_{jL}$;
5: **if** $(R = \emptyset)$ **then**
6: **for** $j = 1$ to m **do**
7: Poll site j for N_j; $t_{jL} \leftarrow N_j$; $s_j \leftarrow \max\{t_{jL}, \frac{\delta T}{m}\}$;
8: **if** $N_j < \frac{\delta T}{m}$ **then** send $t_{jH} \leftarrow \frac{\delta T}{m}$ to j
9: $t_{iL} \leftarrow N_i$; $\hat{N} \leftarrow \sum_j t_{jL}$; $s_i \leftarrow t_{iL}$; $R \leftarrow R \cup \{i\}$;
10: **if** $(\hat{N} < (1-\delta)T)$ **then**
11: $S \leftarrow \sum_{r=1}^{m} s_r$; $t_{min} \leftarrow \min_j\{\frac{t_{jL}}{\sum_{r\in R} t_{rL}}(T-S)\}$;
12: **for all** $j \in E$ **do**
13: **if** $t_{min} < \frac{\delta T}{m}$ **then** send $t_{jH} \leftarrow \frac{\delta T}{m}$ to j;
14: **else** send $t_{jH} \leftarrow t_{jL} + \frac{t_{jL}}{\sum_{r\in S} t_{rL}}(T-S)\}$ to j
15: **else** send $t_{iH} \leftarrow (1+\delta)t_{iL}$ to i;

Figure 3: Modified Adaptive Thresholding Algorithm

Although this algorithm is simple and intuitive, it has some drawbacks: The first time there is a threshold violation from some remote site i, the t_{iH} value at the node is set to T while the value at all other nodes will be set to 0, since $N_j = 0$ at the coordinator site initially. This could unnecessarily trigger a lot of communications especially when several nodes have non-zero counts. Secondly, when the estimated aggregate count at the central node is close to T the new threshold will be very close to the old threshold, thus triggering a lot of threshold violations. In the following section we present a modified algorithm that addresses the above shortcomings.

5.3 Modified Adaptive Algorithm

In order to avoid the problems in the original algorithm for adaptive thresholds we modify the original algorithm, which is illustrated in Figure 3. There are two main differences between the original and modified algorithms: (a) As soon as the central node receives the first threshold violation the t_{iH} values in all the nodes whose counts N_i are below $\frac{\delta T}{m}$ are initialized to $\frac{\delta T}{m}$, and (b) when the difference between global threshold and the estimated aggregate count is small (i.e., below $\frac{\delta T}{m}$), instead of using the adaptive strategy of distributing the difference to all the nodes, we maintain a constant difference between the upper and lower thresholds, i.e., $t_{iH} - t_{iL} = \frac{\delta T}{m}$. In our algorithm in Figure 3, we maintain a set R of nodes whose count exceeds $\frac{\delta T}{m}$. Lines 5–9 deal with the first threshold violation, by polling all nodes to initialize S, and setting upper bounds for the nodes not in R. If the total count is sufficiently below T, lines 10-14 allocate the slack in proportion to the counts; however, we ensure that the difference between higher and lower thresholds is at least $\frac{\delta T}{m}$, using extra variables s_i, to ensure that the total amount of slack allocated stays within the permitted bounds. Lines 3–4 deal with the case when the count first exceeds T, and from that point on, we switch to proportionally increasing counts (line 15) as before.

LEMMA 5. *The modified adaptive thresholding assignment algorithm presented in Figure 3 satisfies the δ-deficient thresholded count constraints.*

PROOF. Consider the case when $\hat{N} < (1-\delta)T$. If a remote site i does not belong to R ($i \notin R$), $t_{iH} = \frac{\delta T}{m}$. In line 11, the rest of the available slack, $T - S$, is proportionally divided to the rest of the sites $\in R$. t_{min} denotes the minimum of the slack values. If $t_{min} < \frac{\delta T}{m}$, then all sites $\in R$ are allocated a slack of $\frac{\delta T}{m}$ in line 13 of the algorithm. Hence $N < \sum_{i=1}^{m} t_{iH} = \hat{N} + \sum_{i=1}^{m} \frac{\delta T}{m} < T$. If $t_{min} > \frac{\delta T}{m}$, then the slacks are proportionally allocated to the sites. Hence $N < \sum_{i=1}^{m} t_{iH} = T$, because the algorithm allocates the slack in the system to the sites. Thus, if $\hat{N} < (1-\delta)T$, $N < T$. When $\hat{N} \geq (1-\delta)T$, the total count exceeds $(1-\delta)T$ and the algorithm follows the proportional spread threshold assignment in the static scheme, and the proof is the same as the previous Lemma. Thus, the modified algorithm satisfies the δ-deficient error constraints. □

THEOREM 8. *The total number of messages from all remote sites to the coordinator using the modified adaptive algorithm is $O(\frac{m}{\delta}(m + \log(\frac{N-T}{m}))$ when $N > T$, and $O(\frac{m^2 N}{\delta T})$ when $N \leq T$.*

PROOF. We split our analysis into two parts, first when the total count is less than T, and the second when it exceeds T. In the first part, the algorithm ensures that the "slack" in each threshold, i.e. $t_{iH} - t_{iL}$ is always at least $\frac{\delta T}{m}$. Thus, there can be at most $\frac{mN}{\delta T}$ threshold violations before the count reaches T, simplifying to $\frac{m}{\delta}$ when N first exceeds T. Each threshold violation causes at most $O(m)$ messages to be sent, to inform the sites of their new high thresholds t_{iH}. When the count is above T, the algorithm mimics the proportional threshold assignment case in Section 4.2, and adapting Lemma 2, the number of messages between remotes sites and the central site to go from T to N is $O(\frac{m}{\delta}\log\frac{N-T}{m})$. The result follows by summing these two bounds. □

Note that one can easily force $\Omega(m^2 + \frac{m}{\delta}\log\frac{N-T}{m})$ messages by first making one site have count $\frac{T}{m}$, then setting $\frac{m}{2}$ counts $N_i = \frac{\delta T}{m}$ (to set up the adaptive thresholds). Then for each of the same $\frac{m}{2}$ sites in turn, set their local count to $N_i = \frac{T}{m}$: each of these settings causes $\theta(m)$ messages, over $\frac{m}{2}$ sites gives the $\Omega(m^2)$ bound. Using the remaining $\frac{m}{2}$ sites (currently with zero local count each), one can then elicit the $\frac{m}{2} + \frac{m}{2}\frac{\log\frac{2(N-T)}{m}}{\log(1+\delta)} = \Omega(\frac{m}{\delta}\log(\frac{N-T}{m}))$ cost from the proportional threshold settings. However, in general, we expect to do much better than this worst case bound, since the analysis is somewhat pessimistic.

6. EXTENSIONS

Negative Updates. Thus far we have assumed that all updates received at remote sites are non-negative. However, a simple observation is that our static protocols remain correct when negative updates are permitted. Instead of checking for thresholds being exceeded, we must check that the upper threshold remains an upper bound, and also that the lower threshold remains a lower bound. Similarly, our adaptive protocols can also handle negative updates with minor modifications. The analysis in previous sections that relates the cost of the protocol to the value of the global count no longer applies: positive and negative updates can cause a lot of communication but leave the global count quite low, thus the communication bound cannot still hold. Indeed, if the updates cause counts to repeatedly cross the same threshold boundaries (in the static case), then the best bound we can state is one that is linear in the number of updates. In the adaptive case, this adverse outcome can be avoided.

Sliding Windows. Being able to handle negative updates means that we can apply our methods to other models of computing counts. Typically, we do not want to monitor counts which increase indefinitely. Indeed, in several of the queries outlined in the Introduction,

time windows were implicitly given in the form of "within an hour" or "in a day". There are several models for dealing with such time-windowed queries: (a) *Periodic reset*. After the time period has elapsed, reset all counts to zero, and restart the protocol. (b) *Sliding window*. Ensure that the current count covers exactly the last hour, for example, by keeping track of past updates, and applying updates older than one hour as negative updates. In the case that there is insufficient storage to retain this many updates, then approximate information can be kept, as explained below. (c) *Overlapping window*. A compromise between periodic reset and sliding window is to apply the overlapping window approach: for example, the window consists of one hour's data, and the start of the window is advanced by five minute intervals every five minutes (so the window contains between 1 hour and 1 hour and five minutes of updates). Now we just have to record the sum of updates in each five minutes and apply these as a single negative update when the start of the window is advanced.

Approximate Counts. So far we have assumed that there is sufficient storage capacity at the remote nodes to store all local count values. But in the case when there are very many updates of different values (for example, tracking network activity), we cannot make this assumption. We may use the same (static) thresholds and δ value for all counts to reduce space usage, but still there may be too many counts to store. The natural solution is to adopt an approximate way of storing the counts, such as Lossy Counting [18] or Count-Min sketch [9]. However, using such approximate structures mean that the guarantees that we can give are much weaker: instead of the δ-deficient guarantee, we must now give a guarantee relative to $\delta N_v + \epsilon \sum_v N_v$, since the approximate counting methods return counts of each item with error $\epsilon \sum_v N_v$. Although we can reduce ϵ (at the cost of more space), in general it is not possible to set a non-zero value of ϵ that gives a δ-deficient guarantee. Hence, the result appear more in line with those that follow for Heavy-hitter style problems [8].

Time-dependent Thresholds. Prior work has built models of how data varies with time in order to reduce the communication cost further [8, 7]. We could apply a similar approach to our methods: the result is *time-dependent local thresholds*. Now we would set our thresholds so that they can increase or decrease as time passes, so that we ensure that the total uncertainty still remains within the same bounds. The idea is that the varying thresholds predict where the true count will lie at time t; if this prediction is correct, then no communication cost is incurred. If any (now time dependent) local threshold is broken, then communication is triggered with the coordinator, and the model can be recalibrated with the recent history. We hope to investigate such extensions in future work.

7. EXPERIMENTAL STUDY

In this section we present the experimental evaluation of our static and adaptive algorithms. We also compare the performance of these algorithms with a technique proposed by Olston *et al* in [21] (referred to as the OJW algorithm in the rest of this work) where the authors try to minimize communication overhead while maintaining a certain accuracy for continuous queries over a distributed data stream.

We begin by presenting our experimental setup in Section 7.1. We then experimentally show that our algorithms always satisfy the problem requirements. We also validate our blended approach for static thresholds by comparing the theoretical results from the model with the results from our experiments. Then we present some observations that provide more insights about the usefulness of our algorithms.

7.1 Experimental Setup

To gain a better understanding of our theoretical analysis, we built a simulator with m monitoring nodes and one central node.

Data Sets. Although the definition of thresholded counts problem is applicable in a variety of different scenarios (as pointed out in Section 1), our focus in the experiments is on a distributed network monitoring system. In this scenario, every node monitors traffic on a link for all the registered events and increments the count for all the events that are observed. We define an event as the occurrence of a combination of *destination IP address* and the *destination port number* in a packet seen by a monitoring node.

We use publicly available link traces from NLANR [20] as input to our distributed monitoring system. These traces are for a single ingress link, and we transform this data for our distributed system by assigning a probability distribution for distributing packets randomly to the various monitors. By using different probability distributions, we can simulate various scenarios that can occur in real networks. For example, a skewed probability distribution function represents a scenario where a few nodes (that are monitoring large inter-domain "peering" links) receive large number of events while others do not. Similarly, a uniform distribution represents a scenario where events are equally likely to occur in any of the monitoring nodes. Although we track all the events that occur in the link traces from NLANR, for the ease of illustration, we present the results for tracking one event whose overall count was 960000.

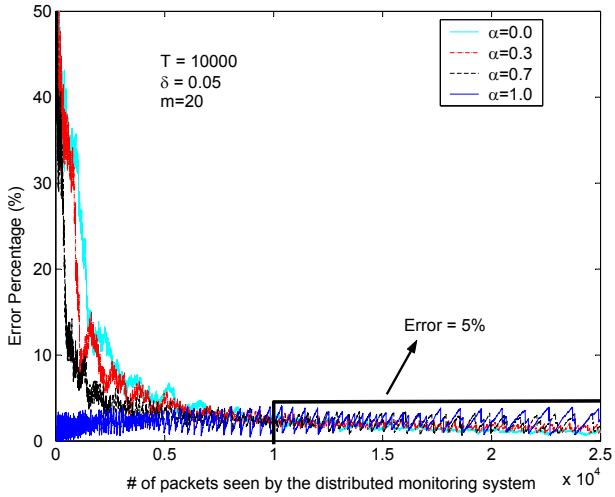
Implementation Issues. We implemented the static and adaptive algorithms described earlier in Sections 4 and 5. Since the OJW algorithm was not proposed to address the thresholded counts problem, we need to set certain parameters of the algorithm in [21] to apply it to our problem. The main issues are:

- The OJW algorithm assumes that a single node can monitor all the updates for a given object/event and a single query can include multiple objects. Since we are interested in tracking the same objects/events in multiple monitors, we treat each item in each site as a separate object/event that is the subject of a single query.
- In the thresholded counts problem definition, the error values are relative, i.e., the maximum error allowed for an event in the system depends on its current count. The original OJW algorithm uses absolute errors (i.e., the total error in the system is required to be below a certain constant value), so to apply to our problem, we set the maximum allowed error for each count to be fixed at δT, divided evenly between all sites where it can occur.

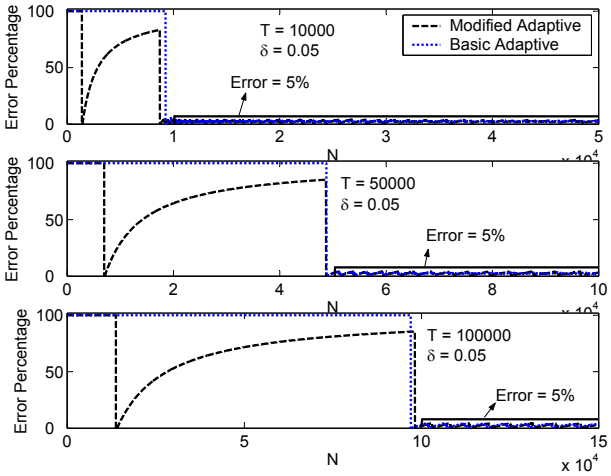
These parameter settings of the OJW algorithm are our best effort to make the algorithm apply to our δ-deficient thresholded count problem. They ensure that the algorithm generates results that are correct according to our problem definition (and the algorithm falls into our class of adaptive algorithms); however, we will see that the cost is much higher than our algorithms that were designed for this problem.

7.2 Performance Accuracy

Count Accuracy. In Figure 4(a) we examine the total error in the distributed monitoring system as packets arrive at various monitoring nodes while using the blended static threshold assignment. We set the values of T, δ, and m to be 10000, 5%, and 20 respectively. When the count of the event is less than T the error in the system can be as high as 50%, but after the count *exceeds the value of* T the error is always less than the value specified by δ (indicated by



(a) N Vs Error percentage in the static case



(b) N Vs Error percentage in the adaptive case

Figure 4: Testing accuracy for static and adaptive cases

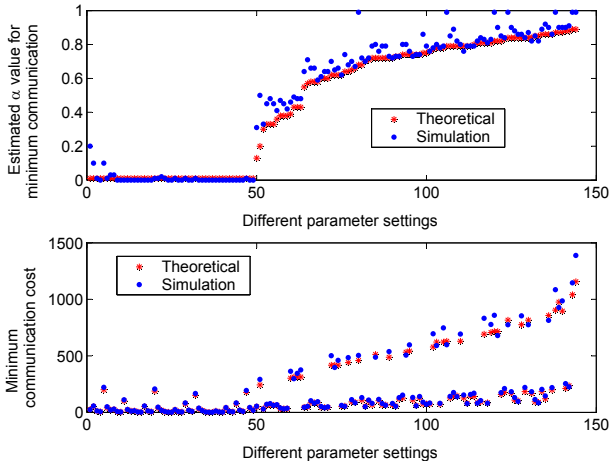


Figure 5: Comparing the optimal theoretical α values with the results obtained from simulation. The x-axis represents different combinations of T, δ, and N. Each combination is referred to as a "parameter setting" and represents a point on the x-axis.

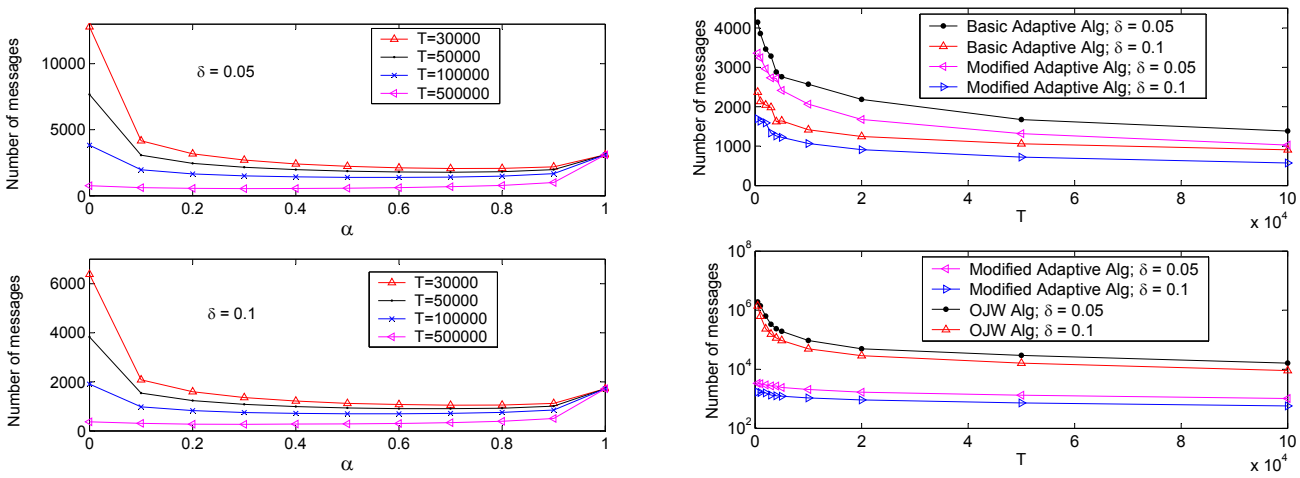
the heavy line on the figure). Different parameter settings yielded similar results.

The results for the same experiment performed with the adaptive algorithms (but varying values of T) are shown in Figure 4(b). The distinctive shape of the curve for the modified adaptive algorithm is explained by the different parts of the algorithm: the initial high error is due to allocating $t_{iH} = \frac{T}{m}$ in the initial phase of the algorithm. The error drops to zero when the central node polls all the monitoring nodes and hence has accurate count information. The error gradually increases when nodes are allocated adaptive thresholds, which allows the total error to grow (within the allowed bounds), until the count reaches $(1-\delta)T$. Finally, the algorithm switches to proportionally growing thresholds, which keep the fractional error within the necessary bounds. Thus we observe that the total error in the system is less than the value specified by δ after the total count exceeds T. Meanwhile, the basic adaptive algorithm has consistently higher error for $N < T$, but also higher communication cost.

Setting α for the Static Algorithm. To validate our theoretical results, we compare the optimal value of α obtained from our theoretical model using a gradient descent approach to the ideal value of α obtained from experiments. For this experiment, we use a uniform distribution to send a given packet from the input file to the monitoring nodes. This is because the static threshold assignment algorithms have a worst case when the packets are uniformly distributed across the remote sites. We repeat the experiment 100 times to ensure that the outcome is not biased by outliers (while generating uniform distribution), and the experimental results presented here are the average value from all the 100 runs.

In our experiments we consider several different values for T and δ. The range of values for T was [100,100000] and the range of values δ was [0.01, 0.1]. As we showed in Section 4.3, the total number of messages in a monitoring system using blended static thresholding approach also depends on N_i, the count of the event in the monitor i. Hence in our experiments we also vary the overall count of the event that we track in the range [2500, 960000]. For each combination of values for T, δ, and N (referred to as a *parameter setting*), we vary the value of α from 0 to 1 with increments of 0.001. For every parameter setting we compute the value of α that resulted in the minimum number of communications, both using simulations and the theoretical model.

The comparison of the ideal values of α from the simulations and theoretical model is shown in the top of Figure 5. Although the theoretical results closely match the experimental values in most of the cases, there are a few cases where the difference between the two is significant. However, these have minimal impact on the overall cost, as shown in the lower half of the figure. The discrepancies are mainly due to the fact that we use integer values in our simulator while our theoretical model ignores this condition and considers thresholds to be real values. The difference between the experimental and theoretical results are significant only when the values of both T and δ are small (i.e., when the system requires high accuracy). We see that in most cases, the cost using the theoretically predicted alpha is as good as or better than the value found by simulations, and in only a few cases is there a slight benefit for the empirically found value.



(a) Communication cost as α varies in static case

(b) Communication case of the adaptive algorithms

Figure 6: Communication Cost of Static and Adaptive Algorithms

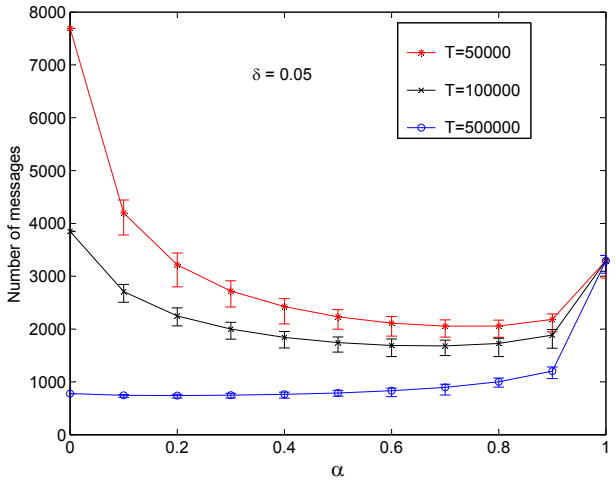


Figure 7: Variation in communication cost with varying α in the static model over 500 repetitions with random incoming packet distribution.

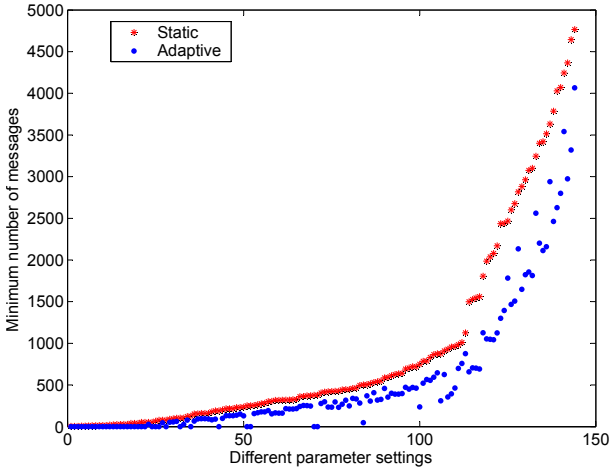


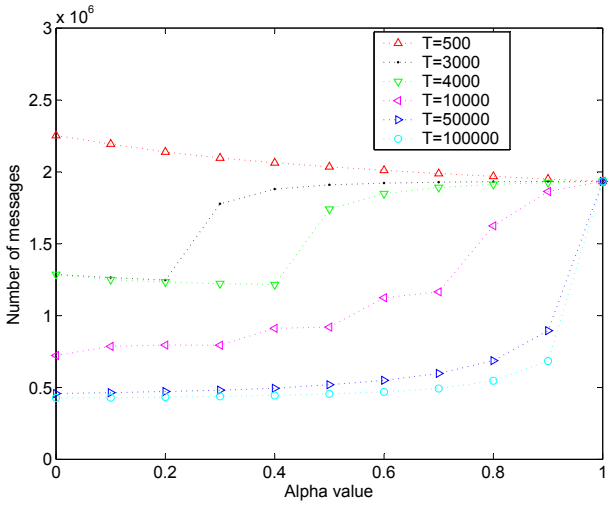
Figure 8: Comparing cost of adaptive and static threshold settings. Each combination of T, δ, and N_i is referred to as a "parameter setting" and represents a single point on the x-axis.

7.3 Communication Cost

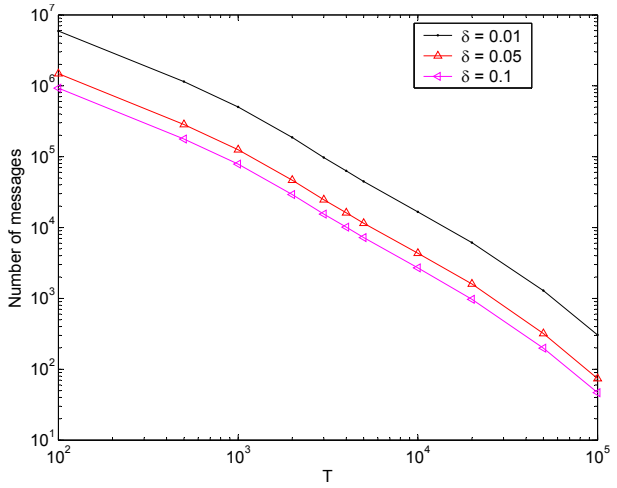
Uniformly Distributed Events. Figure 6(a) examines the impact of α on the number of messages exchanged in a monitoring system using static thresholds. The total count, N, of the event that is tracked is 960000. For a small value of T (i.e., a high value of the ratio $\frac{N}{T}$), as the value of α increases, the number of messages exchanged in the system decreases. In other words, the optimal value of α is close to 1. For a larger value of T (i.e., a small value of $\frac{N}{T}$), the optimal value of α is closer to 0. In essence, as the ratio $\frac{N}{T}$ decreases, the optimal value of α moves towards 0. However, there is a broad range of settings of α which achieve similarly low costs, showing that an approximate value of α will often suffice. Lastly, in line with expectations, decreasing δ increases total cost.

In Figure 6(b) we compare the performance of the basic adaptive, modified adaptive, and the OJW algorithms. The top graph in the figure compares the total number of messages exchanged in the system (from the monitors to the central node and vice versa) using basic adaptive and modified adaptive algorithms. The modified algorithm outperformed the basic algorithm by an appreciable factor in all our experiments. The bottom graph in the figure compares the performance of OJW algorithm with modified adaptive algorithm. Note that the y-axis in this graph is in log scale. The graph shows that the modified algorithm performs at least two orders of magnitude better than the OJW algorithm confirming that the existing techniques are insufficient for this problem.

Randomly Distributed Events. Previous results were based on selecting which node to update uniformly. In Figure 7 we explore the effect of using random distributions to update different nodes. Random distributions are created by generating random probabilities associated with each of the monitoring nodes. These probabilities are used to send the updates from the input trace to the monitoring nodes. Note that a different random distribution is generated for every simulation run. We repeat the simulation 500 times to ensure that we capture a variety of different random distributions. In Figure 7 we plot the average number of messages exchanged due to



(a) Number of messages Vs. α for static thresholds using one hour's data from a live network.



(b) Number of messages Vs. T for adaptive thresholds using one hour's data from a live network.

Figure 9: Experiments on real network data

these random distributions. The error bars in the figure represent the range of values for the number of messages generated by the 500 runs of the simulator. We can see that the effect of using random distributions is relatively small. As before, the optimal value of α that results in the minimum number of messages in the system decreases as the ratio $\frac{N}{T}$ decreases. Note that the total number of messages in the best case (1000-2000) is approximately 0.1% of the total number of updates. Hence we observe a thousand-fold reduction in cost compared to the cost of sending every update to the central site.

Comparing Costs of Static and Adaptive Algorithms. Figure 8 compares the blended static thresholding algorithm and the modified adaptive algorithm in terms of the number of messages in the monitoring system for different parameter settings. For both the algorithms, we vary the values of T, δ, and N in the range [100,100000], [0.01, 0.1], and [2500, 960000] respectively. In the static algorithm, we used the empirically determined optimal value of α for the given parameter setting. We can see that the performance of the adaptive algorithm is always slightly better than the static algorithm. However, which method is best will depend on the scenario in which they are being applied: every message in the static algorithm is only a few bytes long (to indicate the current threshold being used by the site), while the messages are longer in adaptive algorithms since the central site must give more information (the type of message, the new threshold being sent, etc.). In power constrained sensor networks, the energy consumption of the adaptive algorithms may therefore be higher, whereas in more traditional wired networks, the size of message headers will make the difference in size of the messages insignificant.

Experiments on Real Network Data. The results from our experiments presented until now tracked a single event. In order to explore a more realistic and practical scenario, we obtained complete network packet traces from a research network. The network consists of several routers and we obtained anonymized traces of all the packets that entered the network at each of the routers for one hour on Aug 15, 2005.[4] Our network monitoring system architecture consisted of monitoring nodes, one collocated with every router in the network. We used the traces from the collocated router as the input to the monitoring node. The monitoring nodes tracked *all* the incoming events for one hour, approximately 8 million in total.

Figure 9(a) shows the number of messages required by the static monitoring system to track all the events with δ-accuracy. We can see that when the value of T is small, a high value of α results in minimum communication overhead, and at higher values of T, the best value of α reduces. Figure 9(b) shows the number of messages in the monitoring system using adaptive thresholds. Comparing Figures 9(a) and 9(b) we can see that, at large values of T, the adaptive algorithm performs significantly better than the static algorithm. Since Figure 9(b) is plotted on a log-log scale, it shows an approximately linear relation between the logarithm of the number of messages and $\log T$, implying an inverse polynomial dependency on T. This agrees with our analysis of the adaptive algorithm, suggesting that the bulk of the cost is due to items whose count $N_v < T$: for these items, the number of messages is proportional to $\frac{N_v}{\delta T}$, which agrees with the observed behavior.

8. CONCLUSIONS AND FUTURE WORK

We have given a number of algorithms to efficiently monitor distributed sets of counts in a continuous fashion, a fundamental problem at the heart of many network and sensor monitoring problems. In our experimental evaluation, we observed that our adaptive algorithms typically outperform those based on maintaining static thresholds. However, the adaptive algorithms may be more expensive in terms of resources required to run and computational power of the participants.

Several problems remain open to study in future work: firstly, to compare the cost of deploying the these algorithms in real network scenarios such as a particular sensor network environment or IP network monitoring setting (so far we have focused on the pervasive cost issues rather than considering any particular situation). In these situations, we can consider other network topologies, such as more hierarchical approaches to avoid overwhelming the central coordinator. It will be of interest to extend our approach to other query types with a similar thresholded nature, such as arithmetic combinations of thresholds ("report $x + y$ when $x + y > T$" or

[4]For anonymity and proprietary reasons, we cannot give further details of the experimental set up.

"report $x * y$ when $x * y > T$") or apply across sites ("report the number of sites observing event E when E is observed by more than n sites").

9. REFERENCES

[1] D. Abadi, D. Carney, U. Çetintemel, M. Cherniack, C. Convey, C. Erwin, E. Galvez, M. Hatoun, A. Maskey, A. Rasin, A. Singer, M. Stonebraker, N. Tatbul, Y. Xing, R. Yan, and S. Zdonik. Aurora: a data stream management system. In *ACM SIGMOD*, 2003.

[2] A. Arasu, B. Babcock, S. Babu, M. Datar, K. Ito, I. Nishizawa, J. Rosenstein, and J; Widom. STREAM: the Stanford Stream Data Manager (demonstration description). In *ACM SIGMOD*, 2003.

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

[4] S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, S. R. Madden, F. Reiss, and M. A. Shah. TelegraphCQ: continuous dataflow processing. In *ACM SIGMOD*, 2003.

[5] M. Cherniack, H. Balakrishnan, M. Balazinska, D. Carney, U. Cetintemel, Y. Xing, and S. Zdonik. Scalable distributed stream processing. In *Conference on Innovative Data Systems Research*, 2003.

[6] J. Considine, F. Li, G. Kollios, and J. Byers. Approximate aggregation techniques for sensor databases. In *IEEE ICDE*, 2004.

[7] G. Cormode and M. Garofalakis. Sketching streams through the net: Distributed approximate query tracking. In *VLDB*, 2005.

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

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

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

[11] C. Cranor, T. Johnson, O. Spatscheck, and V. Shkapenyuk. Gigascope: A stream database for network applications. In *ACM SIGMOD*, 2003.

[12] A. Das, S. Ganguly, M. Garofalakis, and R. Rastogi. Distributed set-expression cardinality estimation. In *VLDB*, 2004.

[13] C. Estan and G. Varghese. New directions in traffic measurement and accounting. In *ACM SIGCOMM*, 2002.

[14] Ahmad *et. al.* Distributed operation in the borealis stream processing engine. In *ACM SIGMOD*, 2005.

[15] A. Jain, J. Hellerstein, S. Ratnasamy, and D. Wetherall. A wakeup call for internet monitoring systems: The case for distributed triggers. In *ACM SIGCOMM Hotnets*, 2004.

[16] N. Jain, P. Yalagandula, M. Dahlin, and Y. Zhang. INSIGHT: A distributed monitoring system for tracking continuous queries. In *Work-in-progress session at ACM SOSP*, 2005.

[17] A. Manjhi, S. Nath, and P. Gibbons. Tributaries and deltas: Efficient and robust aggregation in sensor network streams. In *ACM SIGMOD*, 2005.

[18] G.S. Manku and R. Motwani. Approximate frequency counts over data streams. In *VLDB*, 2002.

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

[20] National Laboratory for Applied Network Research. `http://www.nlanr.net/`.

[21] C. Olston, J. Jiang, and J. Widom. Adaptive filters for continuous queries over distributed data streams. In *ACM SIGMOD* , 2003.

[22] Stanford stream data manager. `http://www-db.stanford.edu/stream/sqr`.

[23] M. Sullivan and A. Heybey. A system for managing large databases of network traffic. In *USENIX*, 1998.

[24] S. Zdonik, M. Stonebraker, M. Cherniack, and U. Cetintemel. The Aurora and Medusa projects. *Bulletin of the Technical Committee on Data Engineering*, pages 3–10, March 2003.

[25] I. Sharfman, A. Schuster, and D. Keren. A Geometric Approach to Monitoring Threshold Functions Over Distributed Data Streams. In *ACM SIGMOD*, June 2006.

APPENDIX

A. PROOF OF CONVEXITY OF K_N

THEOREM 5. $K_N = m\dfrac{\log(1+\alpha h) - \log(1-\alpha)}{\log(1+\alpha\delta)}$ *is a convex function in* α *in the range* $\alpha \in (0,1)$ *for small values of* δ.

PROOF. We prove that k is convex in α by showing that the second derivative $k'' \geq 0$, where

$$k = \frac{K_N}{m} = \frac{\log(1+\alpha h) - \log(1-\alpha)}{\log(1+\alpha\delta)} = \frac{\log(1+\alpha h) - \log(1-\alpha)}{\alpha\delta}$$

Since $h = \frac{N}{T} - 1$ and $N, T > 0$, $h \in (-1, \infty)$ or $h+1 > 0$. Also, $\alpha, \delta \in (0,1)$, and assuming that δ is very small, we approximate $\ln(1+\alpha\delta)$ as $\alpha\delta$.

Differentiating k, we get the first and second derivates as

$$\delta\alpha\, k' = \frac{h}{1+h\alpha} + \frac{1}{1-\alpha} - \delta k \tag{3}$$

$$\delta\alpha\, k'' = \frac{1}{(1-\alpha)^2} - \frac{h^2}{(1+h\alpha)^2} - 2\delta k' \tag{4}$$

$$= \frac{1}{(1-\alpha)^2} - \frac{h^2}{(1+h\alpha)^2} - \frac{2}{\alpha}\left(\frac{h}{1+h\alpha} + \frac{1}{1-\alpha} - \delta k\right) \tag{5}$$

Using Lemma 6,

$$\ln(1+h\alpha) - \ln(1-\alpha) = \ln\left(\frac{1+h\alpha}{1-\alpha}\right) \geq \frac{2\alpha(h+1)}{2+h\alpha-\alpha}$$

Note that $\frac{1+h\alpha}{1-\alpha} > 1$, since $h+1, \alpha > 0$. Thus,

$$k \geq \frac{1}{\delta\alpha}\frac{2\alpha(h+1)}{2+h\alpha-\alpha} = \frac{2}{\delta}\left(\frac{h+1}{2+h\alpha-\alpha}\right)$$

Substituting this is in Equation(5), we get

$$\begin{aligned}\delta\alpha k'' &\geq \frac{(h+1)(1+2h\alpha-h)}{(1-\alpha)^2(1+h\alpha)^2} - \frac{2}{\alpha}\left(\frac{h+1}{(1+h\alpha)(1-\alpha)} - \frac{2(h+1)}{2+h\alpha-\alpha}\right) \\ &= \frac{(h+1)(1+2h\alpha-h)^2}{(1-\alpha)^2(1+h\alpha)^2(2+h\alpha-\alpha)} \geq 0\end{aligned}$$

Since $k'' \geq 0$, k is convex in α. □

LEMMA 6. $\ln(x) \geq 2\left(\frac{x-1}{x+1}\right)$, *when* $x > 1$

PROOF. Using Taylor's series, we get

$$\ln(\tfrac{1}{x}) = -\ln(x) = \left(\tfrac{1}{x}-1\right) - \tfrac{1}{2}\left(\tfrac{1}{x}-1\right)^2 + \tfrac{1}{3}\left(\tfrac{1}{x}-1\right)^3 - \dots$$

$$\begin{aligned}\text{So } \ln(x) &= \left(1-\tfrac{1}{x}\right) + \tfrac{1}{2}\left(1-\tfrac{1}{x}\right)^2 + \tfrac{1}{3}\left(1-\tfrac{1}{x}\right)^3 + \dots \\ &\geq \left(1-\tfrac{1}{x}\right) + \tfrac{1}{2}\left(1-\tfrac{1}{x}\right)^2 + \tfrac{1}{4}\left(1-\tfrac{1}{x}\right)^3 + \dots \\ &= \left(1-\tfrac{1}{x}\right)\left[1 + \tfrac{1-1/x}{2} + \left(\tfrac{1-1/x}{2}\right)^2 + \dots\right] \\ &= \left(1-\tfrac{1}{x}\right)\left[\tfrac{1}{1-\frac{1-1/x}{2}}\right] = 2\left(\tfrac{1-1/x}{1+1/x}\right) \\ &= 2\left(\tfrac{x-1}{x+1}\right)\end{aligned}$$

□