Distributed Online Tracking

Mingwang Tang[1], Feifei Li[1], Yufei Tao[2]
[1]University of Utah [2]Chinese University of Hong Kong
{tang, lifeifei}@cs.utah.edu taoyf@cse.cuhk.edu.hk

In online tracking, an observer S receives a sequence of values, one per time instance, from a data source that is described by a function f. A tracker T wants to continuously maintain an approximation that is within an error threshold of the value $f(t)$ at any time instance t, with small communication overhead. This problem was recently formalized and studied in [32, 34], and a principled approach with optimal competitive ratio was proposed. This work extends the study of online tracking to a distributed setting, where a tracker T wants to track a function f that is computed from a set of functions $\{f_1, \ldots, f_m\}$ from m distributed observers and respective data sources. This formulation finds numerous important and natural applications, e.g., sensor networks, distributed systems, measurement networks, and pub-sub systems. We formalize this problem and present effective online algorithms for various topologies of a distributed system/network for different aggregate functions. Experiments on large real data sets demonstrate the excellent performance of our methods in practice.

Categories and Subject Descriptors

H.2.m [**Information Systems**]: Database Management – *Miscellaneous*

Keywords

Distributed online tracking, online tracking, tracking

1. INTRODUCTION

The increasing popularity of smart mobile devices and the fast growth in the deployment of large measurement networks generate massive distributed data continuously. For example, such data include, but are not limited to, values collected from smart phones and tablets [3], measurements from large sensor-based measurement networks [12, 22, 31], application data from location based services (LBS) [28], and network data from a large infrastructure network.

Tracking a user function over such distributed data *continuously* in an online fashion is a fundamental challenge. This is a useful task in many applications. A concrete example is the MesoWest project [18]. It collects many aspects of weather data from distributed measurement stations. Users and scientists in MesoWest would like to continuously track the aggregate values (e.g., maximum or minimum) of atmospheric readings, e.g. temperature, from a number of stations in vicinity. Another example is the SAMOS project (Shipboard Automated Meteorological Oceanography System) [27], which collects marine meteorological and near-surface oceanographic observations from distributed research vessels and voluntary ships at sea.

Similar examples can be easily found in location based services and other distributed systems. This problem is also useful in the so called publish/subscribe systems [4, 13], where a subscriber (tracker) may register a function (also known as a query) with a publisher (observer). Data continuously arrive at the publisher. The publisher needs to keep the subscriber informed about the value of her function f, when f is continuously applied over the *current data value*. When a subscriber's function of interest depends on data values from multiple publishers, it becomes a distributed tracking problem.

It is always desirable, sometimes even critical, to reduce the amount of communication in distributed systems and applications, for a number of reasons [1, 5–7, 12, 22, 23, 25, 31]. Many devices rely on on-board battery and incur high power consumption when they communicate, e.g., in sensors and smart phones. Hence, reducing the number of messages they need to send helps extend their battery time. Another reason is to save the network bandwidth. From the user's point of view, less communication often leads to economic gains, e.g., most smart phones have a monthly budget for their data plan, or for nodes in remote areas in a large measurement network, communication via satellites come with a high price tag. From the network infrastructure's point of view (e.g., ISP such as Comcast), too much communications from any application could significantly congest their network and slow down the performance of the network (keep in mind that there could be many user applications running at the same time that share the available network bandwidth).

To achieve exact *continuous online tracking* of arbitrary functions, the solution is to ask all stations to always send readings back to a central coordinator (the tracker), from which various functions can be easily computed and then tracked. This baseline approach, unfortunately, generates excessive communications: every new reading from any station must be forwarded to the tracker to ensure the correctness of the output values of the function being tracked.

SIGMOD'15, May 31–June 4, 2015, Melbourne, Victoria, Australia.

http://dx.doi.org/10.1145/2723372.2737790.

But the good news is, in many application scenarios, exact tracking is often unnecessary. Users are willing to trade-off accuracy with savings in communication. In some applications, approximation is often necessary not just for reducing communication, but also for policy constraints, e.g, due to privacy concerns in location based services [2] and law requirements.

To formalize this accuracy and communication trade-offs, we refer to a distributed site that continuously receives data from a data source as an *observer*, and the centralized site that wants to track a function (or multiple functions) computed over data from multiple, distributed data sources as the *tracker*. Without loss of generality, we assume that the tracker is tracking only one function, which if f. Clearly, f's output is a function of time, and denoted as $f(t)$ for a time instance t. More precisely, it is a function of multiple data values at time instance t, one from each observer. Based on the above discussion, producing the exact values of $f(t)$ continuously for all time instances is expensive. Thus, the tracker's goal is to maintain an approximation $g(t)$, which is his best knowledge of $f(t)$ at any time instance t using small amount of communication (accumulated so far). Focusing on functions that produce a one-dimensional output, we require that $g(t) \in [f(t)-\Delta, f(t)+\Delta]$ for any $t \in [0, t_{\text{now}}]$, for some user-defined error threshold Δ.

Under this set up, when $\Delta = 0$, $g(t)$ always equals $f(t)$ and the baseline exact solution is needed, which is communication-expensive. On the other hand, in the extreme case when $\Delta = +\infty$, $g(t)$ can be a random value, and effectively there will be no communication needed at all. These two extremes illustrate the possible accuracy-communication trade-off enabled by this framework.

Key challenge. It is important to note that our problem is a *continuous online* problem that requires a good approximation for *every time instance*. This is different from many distributed tracking problems in the literature that use the popular *distributed streaming model*, where the goal is to produce an approximation of certain functions/properties computed over the *union of data stream elements seen so far* for all observers, from the beginning of the time till now. It is also different from many existing work on monitoring a function over distributed streams, where the tracker only needs to decide if $f(t)$'s value has exceeded a given (constant) threshold or not at any time instance t.

When there is only one observer, our problem degenerates to a centralized, two-party setting (observer and tracker). This problem has only been recently studied in [32, 34]. where they have studied both one-dimensional and multi-dimensional online tracking in this centralized setting. They have given an online algorithm with $O(\log \Delta)$ competitive-ratio, and shown that this is optimal. In other words, any online algorithm for solving this problem must use at least a factor of $O(\log \Delta)$ more communication than the offline optimal algorithm. Note that, however, this problem is different from the classic problem of two-party computation [30] in communication complexity. For the latter problem, two parties Alice and Bob have a value x and y respectively, and the goal is to compute some function $f(x, y)$ by communicating the minimum number of bits between them. Note that in online tracking, only Alice (the observer) sees the input, Bob (the tracker) just wants to keep track of it. Furthermore, in communication complexity both inputs x and y are given in advance, and the goal is to study the worst-case communication; while in online tracking, the inputs arrive in an online fashion and it is easy to see that the worst-case (total) communication bound for online tracking is meaningless, since the function f could change drastically at each time step. For same reasons, our problem, distributed online tracking, is also different from distributed multi-party computation.

Our contributions. In this paper, we extend the online tracking problem that was only recently studied in [32,34] to the distributed setting with common aggregation functions (e.g., MAX), and investigate principled methods with formal theoretical guarantees on their performance (in terms of communication) when possible. We design novel methods that achieve good communication costs in practice, and formally show that they have good approximation ratios.

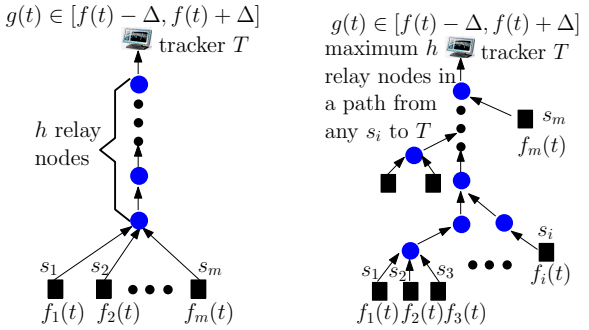
Our contributions are summarized below.

- We formalize the distributed online tracking problem in Section 2 and review the optimal online tracking method from [32, 34] in the centralized setting.
- We examine a special extension of the centralized setting with one observer but many relaying nodes, known as the *chain case*. We study the chain model in Section 3 and design a method with $O(\log \Delta)$ competitive ratio. We also show that our method has achieved the optimal *competitive-ratio* in this setting.
- We investigate the "broom" model in Section 4 by leveraging our results from the chain model, where there are m distributed observers at the leaf-level and a single chain connecting them to the tracker. We design novel method for MAX function and show that our method has very good approximation ratio among the class of online algorithms for the broom model.
- We extend our results to the general tree model in Section 5, which is an extension of the broom model. We again show that our method has good approximation ratio among the class of online algorithms for the general-tree model.
- We discuss other functions and topologies in Section 6.
- We conduct an extensive experiments to evaluate the effectiveness of our methods in practice in Section 7. We used several real data sets and the results have confirmed that our methods are indeed superior compared to other alternatives and baseline methods.

Furthermore, we review the related work in Section 8, and conclude the paper in Section 9. Unless otherwise specified, proofs are provided in Appendix A.

2. PROBLEM FORMULATION AND BACKGROUND

Formally, there are m observers $\{s_1, \ldots, s_m\}$ at m distributed sites, and a tracker T. These m observers are connected to T using a network topology. We consider two common topologies in this work, the *broom* topology and the *general-tree* topology as shown in Figure 1. Observers always locate at the leaves, and the tracker always locates at the root of the tree. Both topologies are constructed based on a *chain topology* as shown in Figure 2(a), and the centralized setting studied in [32, 34] is a special case of the chain topology as shown in Figure 2(b). A relay node does



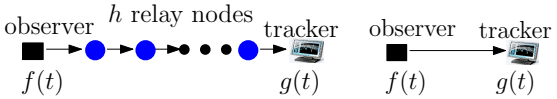
(a) broom model. (b) general-tree.

Figure 1: Track $f(t) = f(f_1(t), f_2(t), \cdots, f_m(t))$.

not directly observe a function (or equivalently data values) that contributes to the computation of f, but it can receive messages from its child (or preceding) node(s), and send messages to its parent (or succeeding) node.

It is important to note that our general-tree topology has *already covered the case* in which an intermediate replay node u may be an observer at the same time, who also observes values (modeled by a function) that contributes to the computation of function f. This is because we can always conceptually add an observer node s directly below (and connected to) such an intermediate node u. Let s report the data values that is observed by u, we can then only view u as a relay node (while making no changes to all other connections to u that already exist). More details on this issue will be presented in Section 6.

That said, in practice, a relay node can model a router, a switch, a sensor node, a computer or computation node in a complex system (e.g., Internet, peer-to-peer network), a measurement station in a monitoring network, etc.



(a) chain topology. (b) centralized setting [32, 34].

Figure 2: Special cases: $g(t) \in [f(t) - \Delta, f(t) + \Delta]$.

Each observer's data value changes (arbitrarily) over time, and can be described by a function. We dub the function at the ith observer f_i, and its value at time instance t $f_i(t)$. The tracker's objective is to continuously track a function f that is computed based on the values of functions from all observers at time instance t, i.e., it's goal is to track $f(t) = f(f_1(t), f_2(t), \ldots, f_m(t))$ continuously over all time instances. Since tracking $f(t)$ exactly is expensive, an approximation $g(t)$ is allowed at the tracker T, subject to the constraint that $g(t) \in [f(t) - \Delta, f(t) + \Delta]$ for any time instance $t \in [0, t_{\text{now}}]$. $\Delta \in \mathbb{Z}^+$ is a user-defined error threshold, that defines the maximum allowed error in approximating $f(t)$ with $g(t)$.

The goal is to find an online algorithm that satisfies this constraint while minimizing the communication cost.

Note that depending on the dimensionality for the outputs of $f(t)$, as well as $f_1(t), f_2(t)$, ..., and $f_m(t)$, we need to track either a one-dimensional value or a multi-dimensional value that changes over time. This work focuses on the one-dimensional case. In other words, we assume that $f(t)$, and $f_1(t), \ldots, f_m(t)$ are all in a one-dimensional space.

2.1 Performance metric of an online algorithm

There are different ways to formally analyze the performance of an online algorithm.

For an online problem P (e.g., caching), let $\mathcal{I}$ be the set of all possible valid input instances, and $\mathcal{A}$ be the set of all valid online algorithms for solving the problem P. Suppose the optimal *offline* algorithm for P is $\mathsf{offline}$. Given an input instance $I \in \mathcal{I}$, and an algorithm $A \in \mathcal{A}$ (or $\mathsf{offline}$), we denote the cost of running algorithm A on I as $\mathsf{cost}(A, I)$. In our setting, the cost is the total number of messages sent in a topology.

A widely used metric is the concept of *competitive ratio*. Formally, for an algorithm $A \in \mathcal{A}$, the competitive ratio of A [24], denoted as $\mathsf{cratio}(A)$, is defined as:

$$\mathsf{cratio}(A) = \max_{I \in \mathcal{I}} \frac{\mathsf{cost}(A, I)}{\mathsf{cost}(\mathsf{offline}, I)}.$$

Another popular metric is to analyze the performance of an algorithm A compared to other algorithms in a class of online algorithms. Formally, we can define the *ratio* of A on an input instance I as follows:

$$\mathsf{ratio}(A, I) = \frac{\mathsf{cost}(A, I)}{\mathsf{cost}(A^*_I, I)},$$

where A^*_I is the online algorithm from the class $\mathcal{A}$ that has the lowest cost on input I, i.e., $A^*_I = \text{argmin}_{A' \in \mathcal{A}} \mathsf{cost}(A', I)$.

Lastly, we can quantify an algorithm A's performance by considering its worst case *ratio*, i.e.,

$$\mathsf{ratio}(A) = \max_{I \in \mathcal{I}} \mathsf{ratio}(A, I).$$

Note that the definitions of $\mathsf{ratio}(A, I)$ and $\mathsf{ratio}(A)$ are inspired by the classic work that has motivated and defined the concept of "instance optimality" [14]. In fact, if $\mathsf{ratio}(A)$ is a constant, then indeed A is an *instance optimal online algorithm*.

Clearly, we always have, for any online problem P and its online algorithm A, $\mathsf{cratio}(A) \leq \mathsf{ratio}(A)$.

2.2 State-of-the-art method

Prior work has studied the online tracking problem in the *centralized, two party setting* [32, 34], as shown in Figure 2(b). They studied both one-dimensional tracking and multi-dimensional tracking, defined by the dimensionality of the output value for the function $f(t)$ at the observer. Since we focus on the one dimension case, here we only review the one-dimension tracking method from [32, 34]. Finding a good online algorithm for this seemingly very simple set up turns out to be a very challenging problem.

Consider the simple case where the function takes integer values at each time step, i.e., $f : \mathbb{Z} \to \mathbb{Z}$, and the tracker requires an absolute error of at most Δ. The natural solution is to let the observer first communicate $f(t_0)$ to the tracker at the initial time instance t_0; then every time $f(t)$ has changed by more than Δ since the last communication, the observer updates the tracker with the current value of $f(t)$. However, this natural solution has an unbounded competitive ratio compared with the offline optimal method. Consider the case where $f(t)$ starts at $f(0) = 0$ and then oscillates between 0 and 2Δ. The above algorithm will communicate for an infinite number of times while the offline optimal solution only needs to send one message: $g(0) = \Delta$.

This example demonstrates the hardness of the online tracking problem. For functions in the form of $f : \mathbb{Z} \to \mathbb{Z}$, Yi and Zhang proposed the method in Algorithm 1, and showed the following results.

Algorithm 1: OptTrack (Δ) (from [32, 34])

```
let S = [f(t_now) − Δ, f(t_now) + Δ] ∩ ℤ
while S ≠ ∅ do
    let g(t_now) be the median of S;
    send g(t_now) to tracker T; set t_last = t_now;
    wait until |f(t) − g(t_last)| > Δ;
    S ← S ∩ [f(t) − Δ, f(t) + Δ]
```

Theorem 1 *(from [32, 34]) To track a function $f : \mathbb{Z} \to \mathbb{Z}$ within error Δ, any online algorithm has to send $\Omega(\log \Delta \cdot \mathsf{OPT})$ messages in the worst case, where OPT is the number of messages needed by the optimal offline algorithm. And,* OptTrack *is an $O(\log \Delta)$-competitive online algorithm to track any function $f : \mathbb{Z} \to \mathbb{Z}$ within Δ. Furthermore, if f takes values from the domain of reals (or any dense set), the competitive ratio of any online algorithm is unbounded.*

Theorem 1 establishes the optimality of the OptTrack method, since it shows that any online algorithms for centralized online tracking (between two nodes) has a competitive ratio that is at least $\log \Delta$, and OptTrack's competitive ratio $O(\log \Delta)$ has met this lower bound.

Note that the negative results on real domains and other dense domains do not rule out the application of OptTrack in practice on those cases. In practice, most functions (or data values for a sequence of inputs) have a fixed precision, e.g., any real number in a 64-bit machine can be described by an integer from an integer domain with size 2^{64}.

To the best of our knowledge, and as pointed out in [32, 34], no prior work has studied the distributed online tracking problem as we have formalized earlier in this section.

3. THE CHAIN CASE

We first examine a special case that bridges centralized and distributed online tracking. Considering the tree topology in Figure 1, it is easy to observe that each observer is connected to the tracker via a single path with a number of relay nodes (if multiple paths exist, we simply consider the shortest path). Hence, online tracking in the chain topology as shown in Figure 2(a) is a basic building block for the general distributed online tracking problem. We refer to this problem as the *chain online tracking*.

The centralized online tracking as reviewed in Section 2.2 and shown in Figure 2(b) is a special case of chain online tracking, with 0 relay node.

Baseline methods. For a chain topology with h relay nodes, a tempting solution is to distribute the error threshold Δ equally to all relay nodes and apply $(h + 1)$ independent instances of the OptTrack algorithm. Suppose we have h relay nodes $\{n_1, \ldots, n_h\}$, an observer s, and a tracker T. Let $n_0 = s$ and $n_{h+1} = T$, for every pair of nodes $\{n_{i-1}, n_i\}$ for $i \in [1, h + 1]$, we can view n_i as a tracker and its preceding node n_{i-1} as an observer, and require that n_i tracks n_{i-1}'s function within an error threshold of $\frac{\Delta}{h+1}$.

Let $y_i(t)$ be the function at n_i for $i \in [1, h + 1]$, then $y_i(t)$ is the output of running OptTrack with an error threshold $\frac{\Delta}{h+1}$, where n_{i-1} is the observer, $y_{i-1}(t)$ is the function to be tracked, and n_i is the tracker. Since $n_0 = s$ and $y_0(t) = f(t)$, we have two facts:

(1) $y_1(t) \in [f(t) - \frac{\Delta}{h+1}, f(t) + \frac{\Delta}{h+1}]$ for any time instance t.

(2) $y_i(t) \in [y_{i-1}(t) - \frac{\Delta}{h+1}, y_{i-1}(t) + \frac{\Delta}{h+1}]$ for any $i \in [2, h+1]$ and any time instance t.

Since the tracker T is simply node n_{h+1}, thus, $g(t) = y_{h+1}(t)$. Using the facts above, it is easy to verify that $g(t)$ will be always within $f(t) \pm \Delta$ as required. We denote this method as ChainTrackA (chain-track-average).

We can generalize ChainTrackA to derive other similar methods. Instead of distributing the error threshold uniformly along the chain, one can distribute a random error threshold Δ_i to node n_i for $i = 1, \ldots, h + 1$, subject to the constraint that $\sum_{i=1}^{h+1} \Delta_i = \Delta$. We denote this method as ChainTrackR (chain-track-random). Using a similar argument, ChainTrackR also ensures that $g(t)$ at T is always within $f(t) \pm \Delta$.

Unfortunately, these seemingly natural solutions do not perform well, even though they are intuitive extensions of the optimal online tracking method between two nodes to multiple nodes.

Given any valid algorithm A for chain online tracking, let $y_i(t)$ be the best knowledge of $f(t)$ at node n_i at any time instance t, for $i = 1, \ldots, h + 1$. The first observation on a chain topology is given by the following lemma.

Lemma 1 *For an algorithm A (either online or offline) that solves the chain online tracking problem, we must have $y_i(t) \in [f(t) - \Delta, f(t) + \Delta]$ for any $i \in [1, h + 1]$ in order to reduce communication while ensuring correctness. This holds for any $t \in [0, t_{now}]$.*

Lemma 1 formalizes a very intuitive observation on a chain topology. This result helps us arrive at the following.

Lemma 2 *Both* ChainTrackA *and* ChainTrackR*'s competitive ratios are $+\infty$ for the chain online tracking problem.*

Proof. We prove the case for ChainTrackA. The proof for ChainTrackR is similar and omitted for brevity.

Consider a function f at the observer s (which is node n_0) whose values always change no more than Δ around a constant a. In other words, $f(t) \in [a - \Delta, a + \Delta]$ for any time instance t.

By the construction of ChainTrackA, we must have:

(1) $y_i(t) \in [f(t) - \frac{i}{h+1}\Delta, f(t) + \frac{i}{h+1}\Delta]$ for any $i \in [1, h]$;

(2) $g(t) = y_{h+1}(t) \in [f(t) - \Delta, f(t) + \Delta]$.

Consider an adversary Alice that tries to explore the worst case for ChainTrackA. Suppose that t_0 is the initial time instance. Alice first sets $f(t_0) = a - \Delta$. It takes $h + 1$ messages to let n_{h+1} learn a valid value for $g(t_0)$ at time t_0. By the facts above, it must be $y_i(t_0) \in [a - \Delta - \frac{i}{h+1}\Delta, a - \Delta + \frac{i}{h+1}\Delta]$ for any $i \in [1, h]$.

Alice then sets $f(t_1) = a + \Delta$. Now $y_i(t_0)$ is more than Δ away from $f(t_1)$ for any $i \in [1, h]$. By Lemma 1, such $y_i(t_0)$'s are not allowed, hence, any node n_i cannot simply set $y_i(t_1) = y_i(t_0)$. Instead, every node n_i needs to receive an update message to produce a valid tracking value $y_i(t_1)$. This leads to h messages. Again, based on the design of ChainTrackA, $y_i(t_1) \in [a + \Delta - \frac{i}{h+1}\Delta, a + \Delta + \frac{i}{h+1}\Delta]$.

Alice sets $f(t_2) = a - \Delta$, by a similar argument, this will again trigger h messages. She repeatedly alternates the subsequent values for f between $a + \Delta$ and $a - \Delta$. ChainTrackA pays at least h messages for any $t \in [t_0, t_{now}]$, which leads to $O(ht_{now})$ messages in total. However, the offline optimal algorithm on this problem instance only needs to set $g(t_0) = y_{h+1}(t_0) = a$ at t_0, which takes $h+1$ messages, and keeps all subsequent $g(t_i)$ same as $g(t_0)$.

Hence, $\mathsf{cratio}(\text{ChainTrackA}) = ht_{now}/(h+1) = t_{now} = +\infty$. □

Optimal chain online tracking. Recall that the centralized, two-party online tracking (simply known as online tracking) is a special case of chain online tracking with no relay nodes, i.e., $h = 0$. The OptTrack method in Algorithm 1 achieves an $O(\log \Delta)$-competitive ratio for the online tracking problem. Furthermore, it is also shown that $O(\log \Delta)$ is the lower bound for the competitive ratio of any online algorithms for online tracking [32,34]. Yet, when generalizing it to chain online tracking with either ChainTrackA or ChainTrackR, the competitive ratio suddenly becomes unbounded. The huge gap motivates us to explore other alternatives, which leads to the optimal chain online tracking method, ChainTrackO (chain-tracking-optimal).

This algorithm is shown in Algorithm 2, and its construction is surprisingly simple: allocate all error threshold to the very first relay node!

Algorithm 2: ChainTrackO (Δ, h)

1 Let the tracking output at a node n_i be $y_i(t)$.
2 Run OptTrack (Δ) between observer s and the first relay node n_1, by running n_1 as a tracker.
3 **for** *any node* n_i *where* $i \in [1, h]$ **do**
4 Whenever $y_i(t)$ has changed, send $y_i(t)$ to node n_{i+1} and set $y_{i+1}(t) = y_i(t)$.

Basically, ChainTrackO ensures that $y_1(t)$ is always within $f(t) \pm \Delta$ using the OptTrack method. For any other relay node n_i for $i \in [2, h+1]$, it maintains $y_i(t) = y_{i-1}(t)$ at all time instances t. The tracker T maintains $g(t) = y_{h+1}(t)$ (recall node n_{h+1} is the tracker node). In other words, $g(t) = y_{h+1}(t) = y_h(t) = \cdots = y_2(t) = y_1(t)$ for any t.

Lemma 3 ChainTrackO*'s competitive ratio is* $O(\log \Delta)$ *for chain online tracking with* h *relay nodes.*

Proof. While running algorithm OptTrack between the observer s and node n_1, we define a round as a time period $[t_s, t_e]$, such that $S = [f(t_s) - \Delta, f(t_s) + \Delta]$ at t_s and $S = \emptyset$ at t_e from line 1 and line 2 in Algorithm 1. By the proof of Theorem 1 in [32,34], we know that OptTrack will communicate $O(\log \Delta)$ messages in a round. Thus, by the construction of Algorithm 2, ChainTrackO has to communicate $O(h \log \Delta)$ messages in this round.

For a round $[t_s, t_e]$, consider any time instance $t \in [t_s, t_e]$. Lemma 1 means that $y_i(t) \in [f(t) - \Delta, f(t) + \Delta]$ for any $i \in [1, h+1]$ in the offline optimal algorithm. Suppose node n_i receives no message in this round, then it must be the case that:

$$y_i(x) \in \cap_{t=t_s}^{x} [f(t) - \Delta, f(t) + \Delta] \text{ for any } x \in [t_s, t_e]. \quad (1)$$

Consider the set $S(x)$ at node n_1 at time x, where $S(x)$ is the set S at time x in Algorithm 1. By the construction of Algorithm 1, $S(x) = \cap_t [f(t) - \Delta, f(t) + \Delta]$ for a subset of time instances t from $[t_s, x]$ (only when $|f(t) - g(t)| > \Delta$, $S \leftarrow S \cap [f(t) - \Delta, f(t) + \Delta]$). Clearly, we must have:

$$\cap_{t=t_s}^{x} [f(t) - \Delta, f(t) + \Delta] \subseteq S(x) \text{ for any } x \in [t_s, t_e]. \quad (2)$$

By the end of this round, $S(t_e)$ becomes $\emptyset$ by the definition of a round, which means that $\cap_{t=t_s}^{x} [f(t) - \Delta, f(t) + \Delta]$ has become $\emptyset$ at some time $x \leq t_e$ by (2). But this means that (1) cannot be true. Hence, our assumption that node n_i receives no message in this round must be wrong. This argument obviously applies to any node n_i for $i \in [1, h+1]$, which implies that an offline optimal algorithm must have sent at least $h + 1$ messages in this round.

Thus, $\mathsf{cratio}(\text{ChainTrackO}) = ((h+1) \log \Delta)/(h+1) = \log \Delta$. □

ChainTrackO is optimal among the class of online algorithms that solve the chain online tracking problem, in terms of its competitive ratio. Specifically, $O(\log \Delta)$ is the lower bound for the competitive ratio of any online algorithms for chain online tracking. Let $\mathsf{C\text{-}OPT}(h)$ be the number of messages sent by the offline optimal algorithm for a chain online tracking problem with h relay nodes.

Lemma 4 *Any online algorithms for chain online tracking of* h *relay nodes must send* $\Omega(\log \Delta \cdot \mathsf{C\text{-}OPT}(h))$ *messages.*

4. THE BROOM CASE

A base case for distributed online tracking is the "broom" topology as shown in Figure 1(a). A broom topology is an extension of the chain topology where there are m observers (instead of only one) connected to the first relay node. Similarly as before, we denote the ith relay node as n_i, and n_1 is the first relay node that connects directly to m observers. In fact, a broom topology reduces to a chain topology when $m = 1$.

Since there are m functions, one from each observer, an important distinction is that the function to be tracked is computed based on these m functions. Specifically, the goal is to track $f(t)$ where $f(t) = f(f_1(t), \ldots, f_m(t))$ for some function f at T subject to an error threshold Δ. Clearly, the design of online tracking algorithms in this case will have to depend on the function f. We *focus on the max aggregate function* in this work, and discuss other aggregate functions in Section 6. Hence, in subsequent discussions, $f(t) = \max(f_1(t), \ldots, f_m(t))$ and $g(t)$ must be in the range $[f(t) - \Delta, f(t) + \Delta]$ at T, for any time instance t.

A baseline. A baseline method is to ask T to track each function $f_i(t)$ within $f_i(t) \pm \Delta$ for $i \in [1, m]$ using a function $g_i(t)$. The tracker computes $g(t) = \max(g_1(t), \ldots, g_m(t))$ for any time instance t. For the ith function, this can be done by using the ChainTrackO method for a chain online tracking instance, where the chain is the path from observer s_i to tracker T. Given that $g_i(t) \in [f_i(t) - \Delta, f_i(t) + \Delta]$ for all $i \in [1, m]$, it is trivial to show that $g(t) \in [f(t) - \Delta, f(t) + \Delta]$. We denote this approach as the m-Chain method.

Improvement. Recall that ChainTrackO allocates all error threshold to the first relay node n_1 in its chain; all other relay nodes simply forward every update arrived at n_1 (from observer s) to T. Hence, in the m-Chain method, it is n_1 that actually tracks $g_1(t), \ldots, g_m(t)$ and n_1 simply passes every update received for $g_i(t)$ through the chain to T. This clearly generates excessive communication. In light

of this, we consider a class A_{broom} of online algorithms for broom online tracking as follows:

1. Every node u in a broom topology keeps a value $y_u(t)$ which represents the knowledge of u about $f(t)$ in the subtree rooted at u at time t. For a leaf node u (an observer), $y_u(t)$ is simply its function value $f_u(t)$.

2. Each leaf node u's function is tracked by its parent v within error Δ using $g_u(t)$, i.e., $|g_u(t) - f_u(t)| \leq \Delta$ for every time instance t. Note that $g_u(t)$ does not need to be $f_u(t)$. Specifically, a leaf u sends a new value $g_u(t)$ to its parent v *only when* $|g_u(last) - f_u(t)| > \Delta$, where $g_u(last)$ is the previous update u sent to v.

Note that in *both broom and tree models*, we *do not* analyze the competitive ratio (cratio) of their online algorithms. The reason is that in a broom or a tree topology, since the offline optimal algorithm offline can see the entire input instance in advance, offline can "communicate" between leaf nodes for free. These are observers that are distributed in the online case. As a result of this, there always exists an input instance where the performance gap between an online algorithm and offline is infinitely large.

Hence, in the following discussion, we will analyze the performance of an online algorithm using the concept of ratio as defined in Section 2.1 with respect to the class A_{broom}.

In a broom topology, we use $y_i(t)$ to denote $y_u(t)$ for a node u that is the ith relay node n_i.

Lemma 5 *Any algorithm $A \in A_{broom}$ must track functions $f_1(t), \ldots, f_m(t)$ with an error threshold that is exactly Δ at the first relay node n_1 in order to minimize* ratio(A).

Lemma 5 implies that $y_u(t)$ at every relay node u must be tracked by its parent node exactly, since all error thresholds have to be allocated to the first relay node n_1.

That said, whenever $y_i(t)$ changes, the message will be popped up along the chain to the tracker T. Formally,

Lemma 6 *Whenever $y_i(t) \neq y_i(t-1)$ at node n_i for any $i \in [1, h]$, any $A \in A_{broom}$ must send an update from n_i to n_{i+1}, and this update message must be $y_i(t)$.*

Proof. This lemma is an immediate result of the above discussion. □

Lemmas 5 and 6 do *not* imply that A_{broom} does not have many choices, because there are still many possible tracking strategies between n_1 and the m leaf nodes (observers). But these two lemmas do help us reach the following theorem.

Theorem 2 *For any algorithm A in A_{broom}, there exists an input instance I and another algorithm $A' \in A_{broom}$, such that* cost(A, I) *is at least h times worse than* cost(A', I)*, i.e., for any $A \in A_{broom}$,* ratio$(A) = \Omega(h)$.

Theorem 2 implies that there does not exist an "overall optimal" algorithm A in A_{broom} that always achieves the smallest cost on all input instances from $\mathcal{I}$. Such an algorithm A would imply ratio$(A) = 1$, which contradicts the above.

Next, we present an online algorithm whose performance is close to the lower bound established by Theorem 2.

The BroomTrack **method.** We design the BroomTrack algorithm in Algorithm 3. Similarly as before, n_{h+1} refers to the tracker T and $g(t) = y_{h+1}(t)$.

Algorithm 3: BroomTrack (Δ, m, h)

```
run m instances of OptTrack (Δ), one instance per
  pair (s_i, n_1). note that s_i is the observer at the ith leaf
  node and n_1, the first relay node, behaves as a tracker
  in OptTrack, for i ∈ [1, m];
let g_j(t) be the tracking result at n_1 at time t for f_j(t);
for any time instance t do
    if no update in any OptTrack instances at n_1
    then set y_1(t) = y_1(t − 1);
    else
        set y_1(t) = max(g_1(t), g_2(t), ..., g_m(t));
    for i = 1, ..., h do
        if t = 0 or y_i(t) ≠ y_i(t − 1) then
            send y_i(t) to n_{i+1} and set y_{i+1}(t) = y_i(t);
        else set y_i(t) = y_i(t − 1);
```

The idea of Algorithm 3 is inspired by the same principle we explored in the design of ChainTrackO for chain online tracking, which is to ask the first relay node to do all the tracking, and the remaining relay nodes simply "relay" the updates sent out by n_1. Specifically, n_1 tracks each function f_i from observer s_i *with error threshold* Δ, and monitors the maximum value among these tracking results; n_1 takes this value as $y_1(t)$, his tracking result for $f(t)$. Other than the first time instance $t = 0$, only a change in this value, when $y_1(t) \neq y_1(t-1)$, will cause a communication through the entire chain, to send $y_1(t)$ to the tracker T and set $g(t) = y_1(t)$. Otherwise, every node in the chain, including the tracker, simply sets $y_u(t) = y_u(t-1)$ without incurring any communication in the chain (from n_1 to T).

The correctness of BroomTrack is obvious: $|g_i(t) - f_i(t)| \leq \Delta$ for any i and t. And, at the tracker T, for any time instance t, $g(t) = y_1(t)$ and $y_1(t) = \max(g_1(t), \ldots, g_m(t))$ immediately lead to $|g(t) - f(t)| \leq \Delta$, for $f(t) = \max(f_1(t), \ldots, f_m(t))$.

Theorem 3 *With respect to online algorithms in A_{broom},* ratio(BroomTrack) $< h \log \Delta$.

Proof. Given an input instance $I \in \mathcal{I}$, we denote M_i as the number of messages between s_i and n_1 for tracking function f_i up to t_{now} by algorithm BroomTrack. Thus, n_1 will receive $\sum_{i=1}^{m} M_i$ messages from $s_1, \ldots, s_m$. In the worst case, all of them get propagated up from n_1 to the root. So cost(BroomTrack, I) $\leq h \sum_{i=1}^{m} M_i$.

On the other hand, for any algorithm A ($A \neq$ BroomTrack) in A_{broom}, it takes at least $\frac{M_i}{\log \Delta}$ messages between s_i and n_1 to track $f_i(t)$ on the input I by Theorem 1. Further, A needs to propagate at least one message through the chain at the first time instance. Thus, cost$(A, I) \geq h + \frac{\sum_{i=1}^{m} M_i}{\log \Delta}$. Hence, for any $I \in \mathcal{I}$, the following holds:

$$\text{ratio}(\text{BroomTrack}, I) \leq \frac{h \sum_{i=1}^{m} M_i}{h + \frac{\sum_{i=1}^{m} M_i}{\log \Delta}} < h \log \Delta.$$

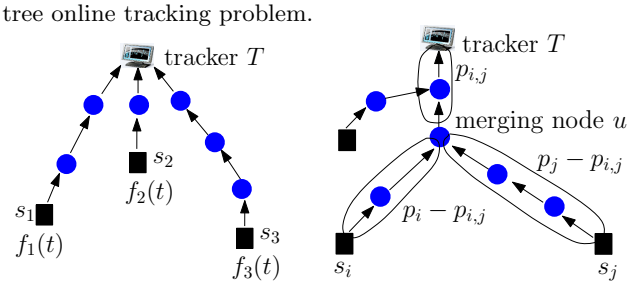
Hence, ratio(BroomTrack) $< h \log \Delta$. □

Similarly, we can show that m-Chain's ratio is $O(h \log \Delta)$ with respect to online algorithms in A_{broom}.

Corollary 1 ratio(m-Chain) $= O(h \log \Delta)$.

5. THE GENERAL TREE CASE

In a general tree topology, every leaf node is still an observer, but it is no longer necessary for all leaf nodes to appear in the same level in the tree. Furthermore, they do not need to share a single chain to the tracker T. We still assume that there are m observers and the tracker T locates at the root. Similarly as before, every non-leaf node (except the root node) is considered a relay node. Using a simililar definition as that for the class A_{broom} in the broom model, we can define A_{tree} as the class of online algorithms for the tree online tracking problem.



(a) simple tree. (b) general tree.

Figure 3: Tree online tracking.

A trivial case is shown in Figure 3(a). Any leaf node (an observer) is connected through a path to the tracker at the root, and no two paths share a common node except the root node. Every such path is a chain, hence, we can run the m-CHAIN method in this case. It is easy to show that m-CHAIN method is *an instance optimal* online method for this simple case, with respect to the class A_{tree}. In other words, its ratio is $O(1)$ with respect to A_{tree}.

So in the general case, an observer at a leaf node is still connected to the tracker through a single path. But a path may join another path at a non-root node u. We call such node u a "merging node". Let p_i be the ith path connecting observer s_i to T. A path p_i may join another path p_j at a merging node u, as illustrated in Figure 3(b). The common part of p_i and p_j is a chain from u to T, and is denoted as $p_{i,j}$ for any such i, j. Note that a path p_i may join with multiple, different paths at either one or more merging node(s), as shown in Figure 3(b). We use $(p - p')$ to denote the *subpath in a path* p that is *not a part of* another path p'.

When two paths p_i and p_j shares a merging node u, they form a *generalized broom model* in the sense that u connects to s_i and s_j through two separate chains, $(p_i - p_{i,j})$ and $(p_j - p_{i,j})$ respectively, and u itself connects to T through a single chain (that is $p_{i,j}$). When both s_i and s_j are directly connected under the merging node u, paths p_i and p_j become exactly a broom model (with two observers).

Given this observation, inspired by Theorem 2, we first have the following negative result.

Corollary 2 *There is no instance optimal algorithm for* A_{tree}.

Also inspired by the above observation, we can extend the idea behind BROOMTRACK to derive the TREETRACK method for tree online tracking. It basically runs a similar version of BROOMTRACK on all generalized broom models found in a tree topology. This algorithm is shown in Algorithm 4 and its correctness is established by Lemma 7.

Lemma 7 *Consider a node u. suppose $y_1, \ldots, y_\ell$ are the most recent updates of its ℓ child nodes. let z be the most recent update sent from u to its parent node. Define $y = \max\{y_1, \ldots, y_\ell\}$. If $y \neq z$, then u must send an update to its parent node, and this update message must be y.*

Algorithm 4: TREETRACK (Δ, a general tree R)

```
for any non-leaf node u in R with an observer s_i as a
    leaf node directly connected under u do
    run an instances of OPTTRACK (Δ) between u and
    s_i, where u is the tracker and s_i is the observer;
for any non-leaf node u in R at any time instance do
    let y_1, ..., y_ℓ be the most recent updates of its ℓ
    child nodes;
    let z be the most recent update of u to its parent;
    set y = max{y_1, ..., y_ℓ};
    if y ≠ z then
        send y as an update to u's parent node in R
g(t) at tracker T is the maximum value among the most
    recent updates T has received from all its child nodes.
```

Lastly, if we denote h_{max} as the max length (the number of relay nodes in a path) of any path in a general tree, we can show the following results for TREETRACK.

Corollary 3 ratio(TREETRACK) $= O(h_{max} \log \Delta)$ *with respect to* A_{tree}.

6. OTHER FUNCTIONS AND TOPOLOGIES

6.1 Other functions for f

It is trivial to see that all of our results hold for min as well. That said, for any *distributive aggregates* on any topology, our methods can be extended to work for these aggregates. A *distributive aggregate* can be computed in a divide-and-conquer strategy, i.e., $f(f_1(t), \ldots, f_m(t)) = f(f(f_{a_1}(t), \ldots, f_{a_i}(t)), f(f_{a_{i+1}}(t), \ldots, f_{a_m}(t)))$, where $\{(a_1, \ldots, a_i), (a_{i+1}, \ldots, a_m)\}$ represents a (random) permutation and partition of $\{1, \ldots, m\}$. Other than max and min, another examples is sum. In online tracking, since we assume that the number of observers is a constant, hence, tracking average is equivalent to tracking sum (because the count is a constant).

So consider sum as an example, we can extend m-CHAIN to both broom and tree models, by allocating Δ/m error for each chain. BROOMTRACK and TREETRACK can be extended as well, by: (1) allocating error thresholds *only* to a chain connecting to a merging node; (2) and making sure that the sum of error thresholds from *all chains to all merging nodes* equals Δ. The rest of the algorithm is designed in a similar fashion as that in BROOMTRACK and TREETRACK, respectively. It is easy to see that, no matter there are how many merging nodes, there are exactly m such chains. So a simple scheme is to simply allocate the error threshold Δ equally to any chain connecting to a merging node. Of course, the ratio of these algorithms needs to be analyzed with respect to the class of online algorithms for sum, which will be different from that in the max case. Such analyses are beyond the scope of this paper, and will be studied in the full version of this work.

More interestingly, our methods can be extended to work with *holistic aggregate* (aggregates that cannot be computed distributively) in certain case, such as *any quantiles* in a *broom* topology. Specifically, we can modify both m-CHAIN and BROOMTRACK for max to derive similar m-CHAIN and

BROOMTRACK methods for distributed online tracking with any quantile function in a broom topology. Suppose $f(t) = \phi(f_1(t), \ldots, f_m(t))$, where $\phi(S)$ represents an ϕ-quantile from a set S of one dimension values for any $\phi \in (0, 1)$. When $\phi = 0.5$, we get the median function.

The only change needed for m-CHAIN is to change $f = \max$ to $f = \phi$-quantile at the tracker, when applying f over $g_1(t), \ldots, g_m(t)$.

For BROOMTRACK, the only change we need to make is in line 6 in Algorithm 3, by replacing max with ϕ-quantile, i.e., $y_1(t) = \phi(g_1(t), \ldots, g_m(t))$. The following lemma ensures the correctness of these adaptions.

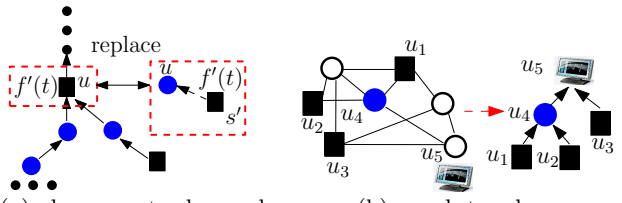
Lemma 8 *Let $y_1(t) = \phi(g_1(t), \ldots, g_m(t))$ and $f(t) = \phi(f_1(t), \ldots, f_m(t))$. If $|g_i(t) - f_i(t)| \leq \Delta$ for all $i \in [1, m]$, then it must be $|y_1(t) - f(t)| \leq \Delta$.*

The proof is provided in Appendix A. Furthermore, using similar arguments, we can show a similar lower bound and upper bound, as that for the max case, on the ratio of these algorithms with respect to the class of online algorithms for tracking a quantile function in the broom model.

For the general-tree model, the m-CHAIN method still works for tracking any quantile function (since the tracker T tracks $f_1(t), \ldots, f_m(t)$ within error Δ with $g_1(t), \ldots, g_m(t)$). However, the TREETRACK method no longer works. The fundamental reason is that one cannot combine quantiles from two subtrees to obtain the quantile value of the union of the two subtrees. A similar argument holds against combining the tracking results from two subtrees in the case of quantile online tracking.

6.2 Other topologies

As we already mentioned in Section 2, the general tree topology can be used to cover the cases when a relay node also serves as an observer at the same time. The idea is illustrated in Figure 4(a). A conceptual observer s' can be added as a leaf-level child node to such a relay node u. Our algorithms and results are carried over to this case. The only difference is that there is no need to run OPTTRACK between s' and u. Instead, u gets $g'(t) = f'(t)$ for free, where $f'(t)$ is the function at s' (the function at u when he acts as an observer). This is useful when intermediate nodes in a network or a distributed system also observe data values of interest to the computation of f being tracked.



(a) observer at relay node. (b) graph topology.

Figure 4: Other topologies.

Lastly, our results from the general tree topology also extend to a graph. On a graph topology G, a distributed online tracking instance has a tracker T at a node from the graph, and m observers on m other nodes on the graph. We can find the shortest path from an observer s_i to T on the graph G, for each $i \in [1, m]$, where the length of a path is determined by the number of graph nodes it contains. Then, a general tree topology can be constructed (conceptually) from these m shortest paths by merging any common graph node from these paths into a single parent node with proper child nodes. T is the root node, and each observer is a leaf node. An example is shown in Figure 4(b).

6.3 Asynchronous updates

So far in our discussion, we assumed that observers receive *synchronous updates* at all time instances, i.e., each observer receives an update on the value of its function at each time instance. In some applications, this may not be the case and observers may receive *asynchronous updates*, in which not only the data arrival rates for different observers may be different, but also the data arrival rate for a single observer may also change over time. Asynchronous updates seem to present new challenges, nevertheless, all our methods can handle asynchronous updates gracefully, in the same way as they do for synchronous updates.

Consider any time instance t for an observer s and its function f, and assume that there are x values, $\{v_1, \ldots, v_x\}$ arriving at s at t. Recall that in our solution framework, u and s simply runs the centralized tracking algorithm OPTTRACK with error parameter Δ [32,34] (say we consider max as an example).

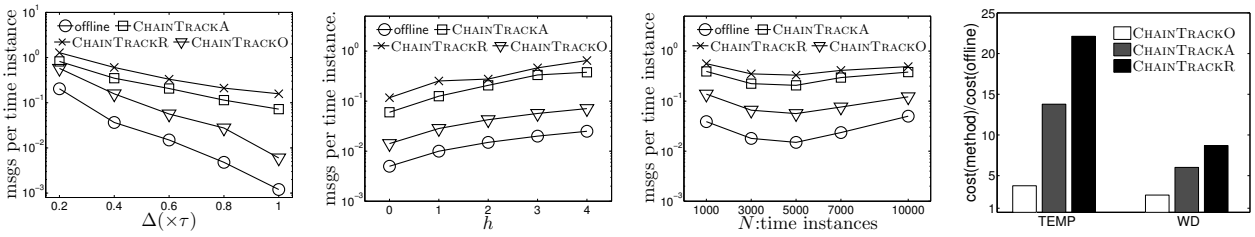
If we treat these values arriving at t as x updates in x time instances $t.1, \ldots, t.x$, and simply continue to run our tracking method between s and u, then u always has an approximation g that is within $f \pm \Delta$ *continuously*. This holds for any observer even if they have different arrival rates. The key observation is that when u receives an update from one observer s_i, but not from another observer s_j, either because s_j didn't receive an update for its function or it did receive an update but OPTTRACK didn't trigger an update message to u, u still holds a valid approximation for each observer's function, guaranteed by the independent centralized tracking (OPTTRACK) instance between u and any observer attached to u!

7. EXPERIMENT

All algorithms were implemented in C++. We tested various distributed topologies, and executed all experiments in a Linux machine with an Intel Core I7-2600 3.4GHz CPU and 8GB memory. Every single communication between *two directly connected nodes* u and v contributes one message.

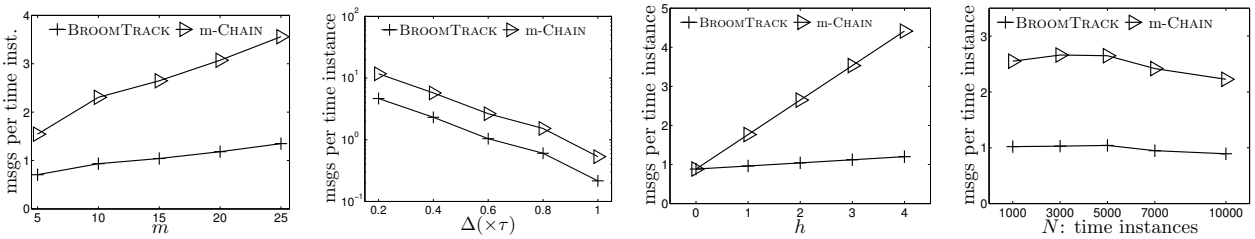
Data sets. We used two real data sets. The first data set is a temperature data set (TEMP) from the MesoWest project [18], a national environmental measurement network. It contains temperature measurements from Jan 1997 to Oct 2011 from 26,383 distinct stations across the United States. We randomly select a subset of stations as the observers and treat readings from a station as the values of the function for that observer.

The second data set is wind direction measurements (WD) from the SAMOS project [27], a weather observation network based on research vessels and voluntary ships at sea. Raw readings from the research vessel Wecoma were obtained which consist of approximately 11.8 million records observed during a 9 month interval in 2010. Wecoma reports a measurement as frequent as 1 second, but may report no readings for a number of seconds between two measurements. We partition these records by a week interval, which leads to 36 chunks. Then we randomly select a subset of data chunks and treat the readings from each chunk as the values of an observer's function.



(a) vary Δ. (b) vary h. (c) vary N. (d) cost(method)/cost(offline)

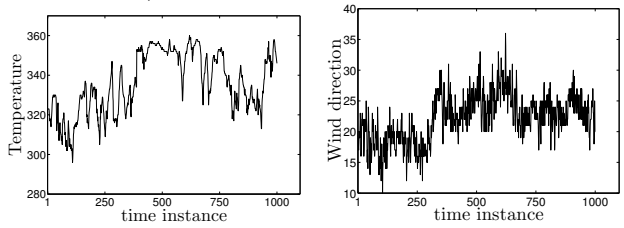
Figure 5: Performance of chain tracking methods on TEMP.



(a) vary m (b) vary Δ. (c) vary h. (d) vary N.

Figure 6: Performance of broom tracking methods on TEMP.

In all data sets, values at each observer are sorted by their timestamps (from MesoWest or SAMOS). By default, at each time instance, we feed each observer one reading as its observed function value at that time instance, which leads to synchronous updates across observers, i.e., each observer receives a single update at each time instance. These two data sets provide quite different distributions: we plot the function values of the function from an observer using a small sample (1000 time instances) in Figure 7.



(a) TEMP (b) WD

Figure 7: $f_1(t)$ for TEMP and WD, for $t \in [1, 1000]$.

Setup. We use TEMP as the default data set. The default values of key parameters are: the number of time instances $N = 5000$; the number of relay nodes in a chain or a broom topology by default is $h = 2$. The default aggregate function f is max. For any function f_i, we compute its standard deviation (std) with respect to $t \in [1, N]$. We set $\tau = \text{avg}(\text{std}(f_1), \ldots, \text{std}(f_m))$ for $f = \max$. We then set the default Δ value to 0.6τ. For the broom model, we set the default number of observers $m = 15$, i.e., number of leaf nodes connecting to the first relay node.

Note that leaf nodes can sit on different levels in a general tree topology. To produce a tree topology, each child node of an internal node with fanout F becomes a leaf node with probability p. We stop expanding nodes when they reach the tree level that equals the tree height H. We set $F = 3$, $p = 0.5$ and $H = 4$ as default values when generating a general tree topology.

For each experiment, we *vary the values for one parameter while setting the other parameters at their default values.* We report the *average total number of messages per time instance* in a topology for different methods. This is denoted as "msgs per time instance" in our experimental results.

7.1 Chain model

Figure 5(a) shows the communication cost when we vary Δ from 0.2τ to τ. Clearly, the communication cost reduces for all methods as Δ increases, since larger Δ values make a tracking algorithm less sensitive to the change of function values. ChainTrackO outperforms both ChainTrackA and ChainTrackR for all Δ values by an order of magnitude. Compared with the cost of the offline optimal method, denoted as offline, ChainTrackO performs well consistently. Averaging over the whole tracking period, offline needs 0.015 message per time instance and ChainTrackO takes only 0.056 message per time instance when $\Delta = 0.6\tau$.

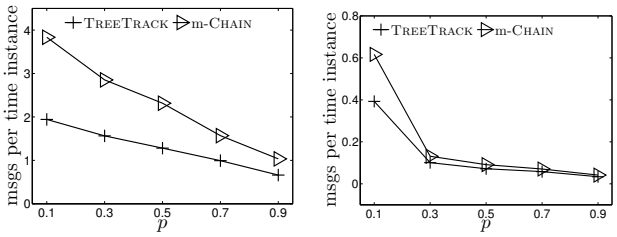
Figure 5(b) shows the communication cost as h increases from 0 to 4. Note that when $h = 0$ the chain model becomes the centralized setting (one observer connects to the tracker directly). Not surprisingly, all methods need more messages on average as the chain contains more relay nodes while h increases. Among the three online algorithms, ChainTrackO gives the best performance for all h values. Meanwhile, we verified that its competitive ratio is indeed independent of h, by calculating the ratio between the number of messages sent by ChainTrackO and offline.

We then vary the number of time instances N from 1000 to 10,000 in Figure 5(c). We observe that the communication cost of all methods first decreases and then increases around $N = 5000$. This is explained by the dynamic nature of functions values over time, due to the real data sets we have used in our experiments.

Figure 5(d) shows the ratio between the cost of a method and the cost of offline, on both TEMP and WD data sets. Clearly, on both data sets, ChainTrackO has significantly outperformed both ChainTrackA and ChainTrackR. The cost of ChainTrackO is very close to the cost of offline.

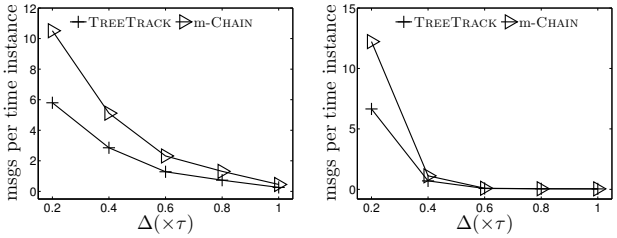
7.2 Broom model

Figure 6(a) shows the communication cost as we vary m the number of observers in a broom topology from 5 to 25. We see that BroomTrack outperforms m-Chain for all m values and the gap enlarges for larger m values. In particular, when $m = 15$ m-Chain takes on average 2.64 mes-



(a) TEMP (b) WD

Figure 8: General tree: vary p.



(a) TEMP (b) WD

Figure 9: General tree: vary Δ.

sages per time instance while BroomTrack takes only 1.04 messages per time instance, in a broom topology with 15 observers and 2 relay nodes. Note that if we were to track function values exactly, this means that we will need 45 messages per time instance! In the following, we use $m = 15$ as the default value in a broom topology.

Figure 6(b) shows the communication cost when we change Δ from 0.2τ to τ. For all Δ values, BroomTrack has outperformed m-Chain by more than 3 times consistently. Again, for similar reasons, a larger Δ value always leads to less communication.

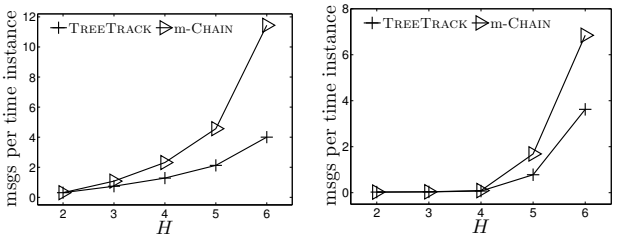
We change h, the number of relay nodes in a broom topology, from 0 to 4 in Figure 6(c). When $h = 0$, there is no relay node and all observers are directly connected to the tracker itself. Therefore, BroomTrack and m-Chain give the same communication cost when $h = 0$. We see that m-Chain suffers from the increase of h much more significantly compared to BroomTrack, and BroomTrack scales very well against more relay nodes. In particular, its number of messages per time instance only increases slightly from 0.88 to 1.20 when h goes from 0 to 4. Again, if we were to track all functions exactly, when the broom has 4 relay nodes and 15 observers, we will need 75 messages per time instance!

We vary N from 1000 to 10,000 in Figure 6(d). It shows that the average number of messages per time instance is quite stable and only decreases slightly when N goes beyond 5000 for both BroomTrack and m-Chain methods. This is caused by the change in the distribution of function values with respect to the time dimension.

7.3 General tree topology

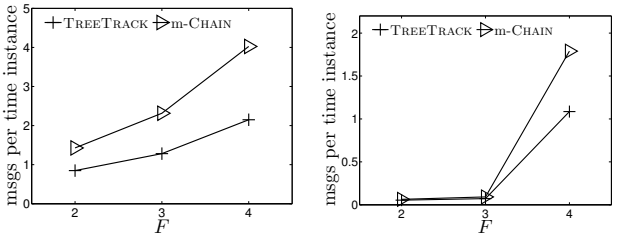
We first vary p, when generating a tree topology, from 0.1 to 0.9 in Figure 8. The corresponding number of observers in the trees that were generated ranges from 27 to 7. Note that larger p values tend to produce less number of observers, since more nodes become leaf nodes (observers) early and they stop generating subtrees even though the height of the tree has not been reached yet in the tree generation process. Not surprisingly, the communication cost of both method reduces as p increases on both data sets. Averaging over the whole tracking period, when $p = 0.5$ TreeTrack takes 1.28 messages per time instance while m-Chain takes 2.21 messages per time instance on TEMP data set. This particular tree has 15 leaf nodes (observers) and 22 nodes in total. In the same tree topology, if we were to track function values exactly, we will need 41 messages per time instance! On WD data set, both methods need even less number of messages per time instance as shown in Figure 8(b). We set $p = 0.5$ as the default value in general-tree topologies.

Figure 9 shows the communication cost when we vary Δ from 0.2τ to τ. Again, TreeTrack outperforms m-Chain for all Δ values on both TEMP and WD data sets. In particular, these results show that TreeTrack is very effective in tracking changes of function values in a tree. For example, Figure 9(b) shows that TreeTrack takes on average 0.07 message per time instance when $\Delta = 0.6\tau$ on WD data set.



(a) TEMP (b) WD

Figure 10: General tree: vary H.



(a) TEMP (b) WD

Figure 11: General tree: vary F.

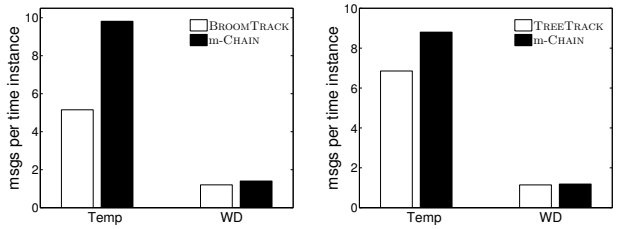
Next, we grow the size of a tree by increasing H, the height of a tree, from 2 to 6 in Figure 10. Note that the number of nodes in a general tree increases exponentially in terms of H. Therefore, both methods show an increase in communication cost as H increases on both data sets. Nevertheless, we still see a significant performance improvement of TreeTrack over m-Chain as H increases on both WD and TEMP data sets. Also note that even though the communication cost increases as a tree grows higher, TreeTrack is still very effective. For example, when $H = 6$, we have a general tree of 91 nodes and 61 of them are leaf nodes (observers). Tracking these functions exactly would require more than 300 messages per time instance. But in this case, TreeTrack has sent on average only 3.6 messages and 4 messages on WD and TEMP data sets respectively.

We vary the fan-out F from 2 to 4 in Figure 11. Not surprisingly, both m-Chain and TreeTrack show an increasing communication cost as F increases on both data sets since larger F values lead to more nodes in a tree. But the cost of TreeTrack increases much slowly. And in all cases, TreeTrack performs much better than m-Chain, and is still orders of magnitude more effective if we compare its cost against the cost of tracking all functions exactly.

7.4 Other functions

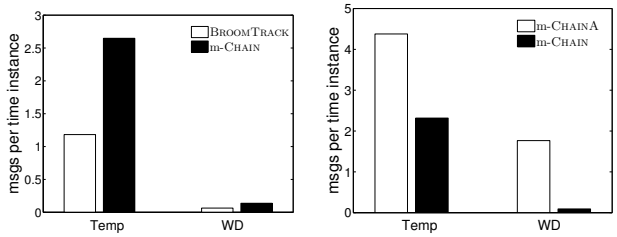
Next, we investigate our online algorithms for tracking sum and median aggregate functions respectively. We eval-

uate their performance using the default parameter values on both data sets, over both broom and tree models. Note that under the default setting, if we were to tracking function values exactly, we will need 45 messages per time instance in the broom instance and 41 messages per time instance in the tree instance.



(a) Broom (b) General tree

Figure 12: Track sum on broom and general tree.



(a) Broom (b) General tree

Figure 13: Track median on broom and general tree.

We first explore the performance of our methods for tracking sum function in Figure 12, for both broom and tree models. In this case, we define $\tau = \text{std}(f(t))$ for $t \in [1, N]$, where we calculate the values of $f(t)$ offline based on functions $f_1(t), \ldots, f_m(t)$, i.e., $f(t) = \sum_{i=1}^{m} f_i(t)$. Our improved methods (BroomTrack and TreeTrack) still outperforms m-Chain in communication cost on both data sets.

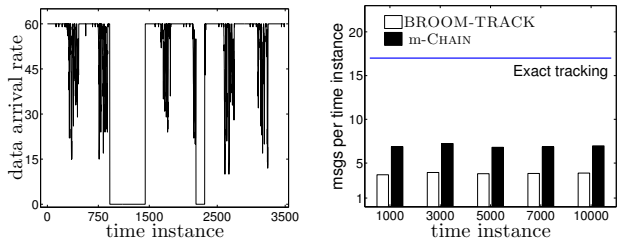
Figure 13 compares the performance of different methods for tracking median function on both broom and tree models. Figure 13(a) shows that BroomTrack outperforms m-Chain in communication cost on both data sets. We evaluate the performance of m-Chain and m-ChainA in Figure 13(b) for general-tree topology since TreeTrack does not work in this case. Here, m-ChainA is a m-chain tracking method that calls ChainTrackA for each chain. It confirms our analysis that allocating tracking error to the first relay node indeed is better than allocating error threshold over different relay nodes in a chain, and m-Chain performs much better than m-ChainA on both data sets.

7.5 Asynchronous updates

By default, we experimented with synchronous updates at all observers, i.e., each observer receives an update per time unit. As discussed in Section 6.3, our methods handle asynchronous updates as well. Next, we show experiments to track the max over observers with different data arrival rates using the WD dataset. We use $m = 15$ observers and default values for other parameters.

In particular, in this experiment, we take *one minute* as a time instance for the SAMOS data set [27]. Each ship in SAMOS collects measurement as frequent as 1 second, but it is also possible that a ship didn't report any measurements for a number of seconds. This leads to different arrival rates for different observers when they measure updates in minute interval. Figure 14(a) shows the data arrival rate over time for an observer over the first 3,500 minutes. Clearly, its data arrival rate fluctuates over time.

Figure 14(b) shows the communication cost over time of BroomTrack and m-Chain under dynamic data arrival rates at 15 observers. We also show msgs per minute for the exact tracking method ($m = 15$ and $h = 2$), which is denoted by the blue line in Figure 14(b). Clearly, both BroomTrack and m-Chain outperform exact tracking, and BroomTrack performs the best over all time instances.



(a) Arrival rate at an observer (b) Communication cost

Figure 14: Broom: max, asynchronous updates.

8. RELATED WORK

Online tracking is a problem that has only been recently studied in [32, 34], which we have reviewed in details in Section 2. Only the centralized setting was studied in [32, 34].

In the popular *distributed streaming model*, the goal is to produce an approximation of certain functions/properties computed over the *union of data stream elements seen so far* for all observers, from the beginning of the time till now, for example, [5–7, 19, 33]). There are studies in distributed streaming model on a time-based sliding window of size W [10,11,17,23,26]. Their objective is to compute an aggregate of interest over the union of all elements (from a single or multiple streams) between 0 and t_{now} or a sliding window of size W (with respect to t_{now}) [5–7, 19, 33].

When a time-based sliding window is used, e.g., [10,11,17], by setting the window size $W = 1$, their problems resemble some similarity to our problem. However, there are two important distinctions. Firstly, prior works for computing aggregates over distributed streams using time-based sliding window focus on computing *aggregates defined over item frequencies* (in fact, this is the case for most streaming algorithms), while our problem is to track an aggregate over *item values*. Secondly, for aggregation over time-based sliding window [10, 11, 17] and streaming algorithms in general, the approximation error is typically quantified by an error parameter $\varepsilon \in (0, 1)$ over the *total frequency* of all item counts in the current window, while an error parameter $\Delta \in \mathbb{Z}^+$ is used in our problem to bound the error in the *output value* of an aggregation over current *item values* from all sites.

In distributed settings with multiple observers, to the best of our knowledge and as pointed out by the prior studies that have proposed the state-of-the-art centralized method [32, 34], a comprehensive study of the distributed online tracking problem with theoretical guarantees is still an open problem. Cormode et al. studied a special instance of this problem, but focused on only *monotone functions* [8, 9]. When f is the top-k function, a heuristic method has been proposed in [1] which provides no theoretical analysis. In contrast, our work focuses on common aggregation functions and principled methods with theoretical guarantees on their communication costs.

Several existing studies explored the problem of threshold monitoring over distributed data [20, 21, 29], where the goal is to monitor continuously if $f(t)$ (a function value computed over the data values of all observers at time instance t) is above a user-defined threshold or not. Note that, in threshold monitoring the tracker only needs to verify if the function value has exceeded a threshold or not, instead of keeping an approximation that is always within $f(t) \pm \Delta$. The geometric-based methods have been further explored to provide better solutions to the threshold monitoring problem [16], and the function approximation problem in the distributed streaming model [15] (which as analyzed above is different from the online tracking problem).

9. CONCLUSION

This paper studies the problem of distributed online tracking. We first extend the recent results for online tracking in the centralized, two party model to the chain model, by introducing relay nodes between observer and tracker. We then investigate both the broom model and the tree model, as well as other different tracking functions. Extensive experiments on real data sets demonstrate that our methods perform much better than baseline methods. Many interesting directions are open for future work, including but not limited to formally analyzing the ratios of our methods when extending them to different aggregates, investigating online tracking with an error threshold that may change over time.

10. ACKNOWLEDGMENT

Mingwang Tang and Feifei Li were supported in part by NSF grants 1251019 and 1200792. Feifei Li was also supported in part by NSFC grant 61428204. The research of Yufei Tao was supported by grants GRF 4168/13 and GRF 142072/14 from HKRGC.

11. REFERENCES

[1] B. Babcock and C. Olston. Distributed top-k monitoring. In *SIGMOD*, pages 28–39, 2003.

[2] A. R. Beresford and F. Stajano. Location privacy in pervasive computing. *IEEE Pervasive Computing*, 2(1):46–55, 2003.

[3] B. Chandramouli, S. Nath, and W. Zhou. Supporting distributed feed-following apps over edge devices. *PVLDB*, 6(13):1570–1581, 2013.

[4] B. Chandramouli, J. Phillips, and J. Yang. Value-based notification conditions in large-scale publish/subscribe systems. In *VLDB*, pages 878–889, 2007.

[5] G. Cormode and M. N. Garofalakis. Sketching streams through the net: Distributed approximate query tracking. In *VLDB*, pages 13–24, 2005.

[6] G. Cormode and M. N. Garofalakis. Streaming in a connected world: querying and tracking distributed data streams. In *EDBT*, 2008.

[7] G. Cormode, M. N. Garofalakis, S. Muthukrishnan, and R. Rastogi. Holistic aggregates in a networked world: Distributed tracking of approximate quantiles. In *SIGMOD Conference*, pages 25–36, 2005.

[8] G. Cormode, S. Muthukrishnan, and K. Yi. Algorithms for distributed functional monitoring. In *SODA*, pages 1076–1085, 2008.

[9] G. Cormode, S. Muthukrishnan, and K. Yi. Algorithms for distributed functional monitoring. *ACM Transactions on Algorithms*, 7(2):21, 2011.

[10] G. Cormode and K. Yi. Tracking distributed aggregates over time-based sliding windows. In *PODC*, pages 213–214, 2011.

[11] G. Cormode and K. Yi. Tracking distributed aggregates over time-based sliding windows. In *SSDBM*, pages 416–430, 2012.

[12] A. Deshpande, C. Guestrin, S. Madden, J. M. Hellerstein, and W. Hong. Model-driven data acquisition in sensor networks. In *VLDB*, pages 588–599, 2004.

[13] Y. Diao, S. Rizvi, and M. J. Franklin. Towards an internet-scale xml dissemination service. In *VLDB*, pages 612–623, 2004.

[14] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. *J. Comput. Syst. Sci.*, 66(4):614–656, 2003.

[15] M. N. Garofalakis, D. Keren, and V. Samoladas. Sketch-based geometric monitoring of distributed stream queries. *PVLDB*, 6(10):937–948, 2013.

[16] N. Giatrakos, A. Deligiannakis, M. N. Garofalakis, I. Sharfman, and A. Schuster. Prediction-based geometric monitoring over distributed data streams. In *SIGMOD*, pages 265–276, 2012.

[17] L.-K. L. Ho-Leung Chan, Tak-Wah Lam and H.-F. Ting. Continuous monitoring of distributed data streams over a time-based sliding window. *Algorithmica*, 62(3-4):1088–1111, 2012.

[18] J. Horel, M. Splitt, L. Dunn, J. Pechmann, B. White, C. Ciliberti, S. Lazarus, J. Slemmer, D. Zaff, and J. Burks. Mesowest: cooperative mesonets in the western united states. *Bull. Amer. Meteor. Soc.*, 83(2):211–226, 2002. http://www.mesowest.org.

[19] Z. Huang, K. Yi, and Q. Zhang. Randomized algorithms for tracking distributed count, frequencies, and ranks. In *PODS*, pages 295–306, 2012.

[20] S. R. Kashyap, J. Ramamirtham, R. Rastogi, and P. Shukla. Efficient constraint monitoring using adaptive thresholds. In *ICDE*, pages 526–535, 2008.

[21] R. Keralapura, G. Cormode, and J. Ramamirtham. Communication-efficient distributed monitoring of thresholded counts. In *SIGMOD*, pages 289–300, 2006.

[22] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. Tinydb: an acquisitional query processing system for sensor networks. *ACM Trans. Database Syst.*, 30(1):122–173, 2005.

[23] A. Manjhi, V. Shkapenyuk, K. Dhamdhere, and C. Olston. Finding (recently) frequent items in distributed data streams. In *ICDE*, pages 767–778, 2005.

[24] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[25] C. Olston, B. T. Loo, and J. Widom. Adaptive precision setting for cached approximate values. In *SIGMOD Conference*, pages 355–366, 2001.

[26] O. Papapetrou, M. N. Garofalakis, and A. Deligiannakis. Sketch-based querying of distributed sliding-window data streams. *PVLDB*, 5(10):992–1003, 2012.

[27] SAMOS. Shipboard Automated Meteorological and Oceanographic System. http://samos.coaps.fsu.edu.

[28] J. H. Schiller and A. Voisard, editors. *Location-Based Services*. Morgan Kaufmann, 2004.

[29] I. Sharfman, A. Schuster, and D. Keren. A geometric approach to monitoring threshold functions over distributed data streams. In *SIGMOD*, pages 301–312, 2006.

[30] A. C.-C. Yao. Some complexity questions related to distributive computing (preliminary report). In *STOC*, pages 209–213, 1979.

[31] Y. Yao and J. Gehrke. Query processing in sensor networks. In *CIDR*, 2003.

[32] K. Yi and Q. Zhang. Multi-dimensional online tracking. In *SODA*, pages 1098–1107, 2009.

[33] K. Yi and Q. Zhang. Optimal tracking of distributed heavy hitters and quantiles. In *PODS*, pages 167–174, 2009.

[34] K. Yi and Q. Zhang. Multidimensional online tracking. *ACM Transactions on Algorithms*, 8(2):12, 2012.

APPENDIX

A. PROOFS

Lemma 1 *For an algorithm A (either online or offline) that solves the chain online tracking problem, we must have $y_i(t) \in [f(t)-\Delta, f(t)+\Delta]$ for any $i \in [1, h+1]$ in order to reduce communication while ensuring correctness. This holds for any $t \in [0, t_{now}]$.*

PROOF. We prove a general fact that holds for any relay node and the tracker. Consider any node n_i for $i \in [1,h]$, and its successor n_{i+1}. We prove that the error (defined as $|y_{i+1}(t) - f(t)|$) at n_{i+1} cannot be lower than that at n_i for any time instance t. This is true because no matter what n_i tells n_{i+1}, n_i can also tell that to itself. In other words, n_i has at least the same amount of information that n_{i+1} does in order to reduce the tracking error.

That said, since we must have $|y_{h+1}(t) - f(t)| \leq \Delta$ at any time instance t at the tracker node n_{h+1}, combined with the fact above, it implies that $|y_i(t) - f(t)| \leq \Delta$ for any t and $i \in [1,h]$. □

Lemma 4 *Any online algorithms for chain online tracking of h relay nodes must send $\Omega(\log\Delta \cdot \mathsf{C\text{-}OPT}(h))$ messages.*

PROOF. Suppose A is an online algorithm for chain online tracking with h relay nodes. The approximations of $f(t)$ at different nodes are $y_1(t), y_2(t), \ldots, y_h(t), y_{h+1}(t)$ respectively (note that n_{h+1} is the tracker T, and $g(t) = y_{h+1}(t)$).

Consider an adversary Alice, who maintains an integer set R such that any $y \in R$ at time instance t is a valid representative for the value of $f(t)$, for $t \in [t_s, t_e]$. Note that a round is defined as a period $[t_s, t_e]$, such that $R = [f(t_s) - \Delta, f(t_s) + \Delta]$ at t_s and $R = \emptyset$ at t_e. Let d_i be the number of integers in R after the i-th update sent out by algorithm A from the observer. Initially, $R = [f(t_s) - \Delta, f(t_s) + \Delta]$ and $d_0 = 2\Delta + 1$.

In each round $[t_s, t_e]$, we will show that there exists a sequence of $f(t)$ values such that A has to send $\Omega(\log\Delta \cdot \mathsf{C\text{-}OPT}(h))$ messages, but the optimal offline method has to send only $\mathsf{C\text{-}OPT}(h)$ messages.

Consider a time instance $t \in [t_s, t_e]$ after the i-th update sent by algorithm A. Let x be the median of R. Without loss of generality, suppose more than $\frac{h+1}{2}$ values among $\{y_1(t), y_2(t), \ldots, y_{h+1}(t)\}$ are in the range $[\min(R), x]$, let $Y_{\leq}(t)$ be the set of such values and $z = \max(Y_{\leq}(t))$.

Alice sets $f(t+1) = z + \Delta + 1$. It is easy to verify that $f(t+1) - y_i(t) > \Delta$ for any $y_i(t) \in Y_{\leq}(t)$. Hence, any node n_i, such that $y_i(t) \in Y_{\leq}(t)$, must receive an update at $(t+1)$ to ensure that $y_i(t+1) \in [f(t+1) - \Delta, f(t+1) + \Delta]$ by Lemma 1. This means that A has to send at least $\frac{h+1}{2}$ messages (the size of $Y_{\leq}(t)$) from t to $t+1$.

After the $(i+1)$-th update sent out by A from the observer, R has to be updated as $R \leftarrow R \cap [f(t+1) - \Delta, f(t+1) + \Delta]$, i.e., $R = [z+1, x + (d_i - 1)/2]$ and its size reduces by at most half at each iteration when A sends out an update from the observer. It's easy to see that $d_{i+1} \geq (d_i - 1)/2$. Using the same argument, we will get a similar result if more than $\frac{h+1}{2}$ values of $\{y_1(t), \ldots, y_{h+1}(t)\}$ are in the range $[x, \max(R)]$. In that case, we set $f(t+1) = z - \Delta - 1$ where $z = \min(Y_{\geq}(t))$.

That said, it takes at least $\Omega(\log\Delta)$ iterations for R to be a constant, since R's initial size is $O(\Delta)$ and its size reduces by at most half in each iteration. When R becomes empty, Alice starts a new round. By then, an offline optimal algorithm must have to send at least one update from the observer to the tracker (as $g(t)$ must hold a value from R to represent $f(t)$ for $t \in [t_s, t_e]$), which takes at least $O(h)$ messages. Hence, in any such round $[t_s, t_e]$, when $R = \emptyset$ at t_e, $\mathsf{C\text{-}OPT}(h) = O(h)$, but A has to send at least $\Omega(\frac{h+1}{2}\log\Delta)$ messages, which completes the proof. □

Lemma 5 *Any algorithm $A \in A_{broom}$ must track functions $f_1(t), \ldots, f_m(t)$ with an error threshold that is exactly Δ at the first relay node n_1 in order to minimize $\mathsf{ratio}(A)$.*

PROOF. Assume that the claim does not hold. This means there exists an algorithm $A \in A_{broom}$ that allows node n_1 to track at least one function $f_i(t)$ using an error threshold δ that is less than Δ. Without loss of generality, consider $i = 1$. In this case, we can show that $\mathsf{ratio}(A) = +\infty$ by constructing an input instance I as follows.

In this instance I, $f_j(t) = -2\Delta$ for $j = 2, \ldots, m$ and any time instance t; $f_1(t) = (-1)^{t \mod 2}\Delta$. In other words, $f_1(t)$ alternates between $-\Delta$ and Δ and all other functions are set to a constant -2Δ. In this case, clearly, $f(t) = \max(f_1(t), \ldots, f_m(t)) = f_1(t)$ for all time instances t. But since the error threshold allocated to n_1 for f_1 is $\delta < \Delta$, A needs to send $+\infty$ number of messages when t goes $+\infty$, no matter how it designs its online tracking algorithm between n_1 and s_1.

But the optimal online algorithm *for this particular instance* only needs to send $(m + h + 1)$ number of messages in the first time instance. This algorithm A_I^* sets $g_1(t_1) = f_1(t_1) + \Delta$, and $g_i(t_1) = f_i(t_1)$ for all $i > 1$ at the first time instance t_1. It then asks each observer s_i to only send an update to n_1 if and only if at a time instance t, $f_i(t)$ has changed more than Δ from its last communicated value to n_1. At any time instance, n_1 sends $y_1(t) = \max(g_1(t), \ldots, g_m(t))$ to T, through the chain defined by $n_2, \ldots, n_h$, if $y_1(t)$ is different from the last communicated value from n_1 to T. It is easy to verify that this algorithm belongs to A_{broom} and it works correctly on all possible instances. On the above input instance, at the first time instance t_1, it will send $g_1(t_1) = 0$, and $g_2(t_1) = \cdots = g_m(t_m) = -2\Delta$ to n_1 from observers $s_1, \ldots, s_m$ (n_1 also forwards only $y_1(t) = 0$ to T). But it will incur no more communication in all subsequent time instances. Hence, its communication cost is $(m + h + 1)$.

Thus, on this input instance I, $\mathsf{ratio}(A, I)/\mathsf{ratio}(A_I^*, I) = +\infty$. As a result, $\mathsf{ratio}(A) = +\infty$.

Note that the optimal algorithm for a particular input instance may perform badly on other input instances. The definition of ratio is to quantify the difference in the worst case between the performance of an online algorithm A against the best possible performance for a valid input instance. □

Theorem 2 *For any algorithm A in A_{broom}, there exists an input instance I and another algorithm $A' \in A_{broom}$, such that $\mathsf{cost}(A, I)$ is at least h times worse than $\mathsf{cost}(A', I)$, i.e., for any $A \in A_{broom}$, $\mathsf{ratio}(A) = \Omega(h)$.*

PROOF. For simplicity and without loss of generality, it suffices to prove the theorem for $m = 2$, i.e., a broom topology that has only 2 observers at the leaf level. We denote the two observers as s_1 and s_2 respectively. For an algorithm $A \in A_{broom}$, we will play as an adversary to construct two bad input instances I_1 and I_2 with respect to A.

Initially, we set $f_1(t_1) = \Delta$. Suppose in algorithm A s_1 sends a value x_1 to n_1. Note that x_1 must be an integer in $[0, 2\Delta]$. When $x_1 > 0$ we'll construct input instance I_1; otherwise we'll construct instance I_2 for $x_1 = 0$.

Time instance	$f_1(t)$	$g_1(t)$	$f_2(t)$	$g_2(t)$	$y_1(t)$
Initialization					
t_1	Δ	x_1	$x_1 + \frac{3}{2}\Delta$	x_2	x_2
t_2	$x_1 + \Delta$	x_1	$\ll 0$	$\ll 0$	x_1
Round					
t_{2i+1}	$x_1 + \Delta$	x_1	$x_1 + \frac{3}{2}\Delta$	x_2	x_2
t_{2i+2}	$x_1 + \Delta$	x_1	$\ll 0$	$\ll 0$	x_1

Table 1: Input instance I_1 and behavior of A.

Table 1 shows the construction of I_1 and the behavior of A on I_1. At time t_1, we set $f_2(t_1) = x_1 + \frac{3}{2}\Delta$ and s_2 sends a value x_2 to n_1. Clearly, x_2 is strictly larger than x_1 since $x_2 \geq x_1 + \frac{\Delta}{2}$. Hence, at node n_1, $g_1(t_1) = x_1$ and $g_2(t_1) = x_2$. We have $y_1(t_1) = \max(g_1(t_1), g_2(t_1)) = x_2$ and it will be propagated all the way up to the tracker at the root node by Lemma 6.

At time t_2, we update $f_1(t_2) = x_1 + \Delta$ and $f_2(t_2)$ to a value that is $\ll 0$. Note that s_1 sends no message to n_1 since $|x_1 - f_1(t_2)| \leq \Delta$; meanwhile, s_2 needs to send an update message that is $\ll 0$ to n_1 because $|g_2(t_1) - f_2(t_2)| > \Delta$ and $f_2(t_2) \ll 0$. Now, $y_1(t_2)$ will be set back to x_1 which will be propagated up to the tracker T.

The rest of I_1 is constructed in rounds. Right before each round, we always ensure that $y_1(t) = x_1$ and $g_2(last) \ll 0$. Each round i contains two time instances t_{2i+1} and t_{2i+2}. In a round i, we keep f_1's value at $x_1 + \Delta$ and alternate the values of f_2 between $x_1 + \frac{3}{2}\Delta$ and some value that is $\ll 0$. Specifically, at time t_{2i+1}, s_2 will send a value x_2 to n_1 and $g_2(t_{2i+1})$ will be set to x_2. But s_1 will not send any message and $g_1(t_{2i+1})$ will still be x_1. Because $|x_2 - (x_1 + \frac{3}{2}\Delta)| \leq \Delta$, clearly, $x_2 > x_1$. Hence, $y_1(t_{2i+1}) = x_2$, and by Lemma 6, $y_1(t_{2i+1}) = x_2$ will be propagated up to the tracker. At the following time t_{2i+2}, s_2 sends another update message to n_1 that sets $g_2(t_{2i+2})$ to be $\ll 0$, and s_1 again sends no update message and $g_1(t_{2i+2})$ is still x_1. Hence, $y_1(t_{2i+2})$ goes back to x_1. Again, this update on $y_1(t)$ will be propagated up to the tracker according to Lemma 6.

In summary, for every time instance in a round, A will incur h messages. Hence, $\mathsf{cost}(A, I_1)$ equals $O(ht_{now})$.

Next, we show the existence of another algorithm A' in A_{broom} which only sends $O(h + t_{now})$ messages on the same input I_1.

Time instance	$f_1(t)$	$g_1(t)$	$f_2(t)$	$g_2(t)$	$y_1(t)$
Initialization					
t_1	Δ	0	$x_1 + \frac{3}{2}\Delta$	$x_1 + \Delta$	$x_1 + \Delta$
t_2	$x_1 + \Delta$	$x_1 + \Delta$	$\ll 0$	$\ll 0$	$x_1 + \Delta$
Round					
t_{2i+1}	$x_1 + \Delta$	$x_1 + \Delta$	$x_1 + \frac{3}{2}\Delta$	$x_1 + \Delta$	$x_1 + \Delta$
t_{2i+2}	$x_1 + \Delta$	$x_1 + \Delta$	$\ll 0$	$\ll 0$	$x_1 + \Delta$

Table 2: A' on input instance I_1.

At time t_1, A' sends $g_1(t_1) = 0$ from s_1 and $g_2(t_1) = x_1 + \Delta$ from s_2 to n_1, for $f_1(t_1) = \Delta$ and $f_2(t_1) = x_1 + \frac{3}{2}\Delta$ respectively. Hence, $y_1(t_1) = x_1 + \Delta$ and it will be propagated up to the tracker at the root.

At time t_2, $f_1(t_2) = x_1 + \Delta$. This forces s_1 to send a new update message to n_1 because $|f_1(t_2) - g_1(t_1)| = |x_1 + \Delta - 0| > \Delta$ (note that while constructing I_1, we assumed that $x_1 > 0$). At this time, A' sends $g_1(t_2) = x_1 + \Delta$ for tracking $f_1(t_2) = x_1 + \Delta$. On s_2, $f_2(t_2) \ll 0$ which also forces an update message $g_2(t_2) \ll 0$ to be sent to n_1. But $y_1(t_2) = \max(g_1(t_2), g_2(t_2))$, which still equals $y_1(t_1) = x_1 + \Delta$! Thus, $y_1(t_2) = x_1 + \Delta$ does not need to sent up to the tracker along the chain.

In subsequent rounds, A' is able to maintain $y_1 = x_1 + \Delta$ with respect to I_1 in each round as shown in Table 2. That said, it only takes two messages for updating $g_2(t)$ at n_1 in each round. Therefore, $\mathsf{cost}(A', I_1) = h + 4 + 2 \times r$ for r rounds, which is $O(h + t_{now})$ since each round has two time instances. This means that $\mathsf{cost}(A', I_1) = O(h + t_{now})$.

Similarly, we can construct a bad input instance I_2 for algorithm A when $x_1 = 0$ as shown by Table 3. And for this input instance, there exists another algorithm A'' in A_{broom} that only takes $(h + 3 + 2 \times r) = O(h + t_{now})$ messages on input I_2 as shown by Table 4.

Time instance	$f_1(t)$	$g_{s_1}(t)$	$f_2(t)$	$g_{s_2}(t)$	$y_1(t)$
Initialization					
t_1	Δ	0	$\Delta + 1$	x_2	x_2
t_2	Δ		$\ll 0$	$\ll 0$	0
Round					
t_{2i+1}	Δ		$\Delta + 1$	x_2	x_2
t_{2i+2}	Δ		$\ll 0$	$\ll 0$	0

Table 3: A on input instance I_2.

Time instance	$f_1(t)$	$g_{s_1}(t)$	$f_2(t)$	$g_{s_2}(t)$	$y_1(t)$
Initialization					
t_1	Δ	1	$\Delta + 1$	1	1
t_2	Δ		$\ll 0$	$\ll 0$	1
Round					
t_{2i+1}	Δ		$\Delta + 1$	1	1
t_{2i+2}	Δ		$\ll 0$	$\ll 0$	1

Table 4: A'' on input instance I_2.

That said, for any algorithm $A \in A_{broom}$, there always exists an input instance I and an algorithm $A' \in A_{broom}$, such that $\mathsf{cost}(A, I) = O(ht_{now})$ and $\mathsf{cost}(A', I) = O(h + t_{now})$. In other words, $\mathsf{ratio}(A) = \Omega(h)$. □

Corollary 2 *There is no instance optimal algorithm for A_{tree}.*

PROOF. Since we have excluded the trivial case in the tree model, a tree topology must have at least two paths p_i and p_j that shares a merging node u. The paths p_i and p_j form a generalized broom model as discussed above. Suppose path $p_{i,j}$ consists of h relay nodes.

By Theorem 2, for any algorithm $A \in A_{tree}$, there always exists an input instance I and another algorithm $A' \in A_{tree}$, such that the cost of A on I on path $p_{i,j}$ is at least h times worse than the cost of A' on I on path $p_{i,j}$. The cost of A on I on path $(p_i - p_{i,j})$ and path $(p_j - p_{i,j})$ is at best the same as the cost of A' on I on these two paths. □

Lemma 7 *Consider a node u. suppose $y_1, \ldots, y_\ell$ are the most recent updates of its ℓ child nodes. let z be the most recent update sent from u to its parent node. Define $y = \max\{y_1, \ldots, y_\ell\}$. If $y \neq z$, then u must send an update to its parent node, and this update message must be y.*

PROOF. Without loss of generality, assume that $y = y_i$. The jth child node of u is referred to as the node j, and its function is denoted as f_j.

We prove the theorem by induction. First, consider the base case when u only has leaf nodes as its child nodes (i.e., u only has observers as its child nodes). We first show that u must send an update to its parent.

case 1: $y > z$: For node i, set f_i to $y+\Delta$. For node $j \neq i$, if $y_j \geq y - 2\Delta$, then set f_j to $y - \Delta$.

case 2: $y < z$: For node i, set f_i to $y-\Delta$. For node $j \neq i$, if $y_j \geq y - 2\Delta$, then set f_j to $y - \Delta$.

With some technicality, we can show that in both cases: (1) $y \in [f(t)-\Delta, f(t)+\Delta]$ where $f(t) = \max(f_1(t), \ldots, f_\ell(t))$; and (2) $z \notin [f(t) - \Delta, f(t) + \Delta]$. Hence, node u must send an update to its parent. Now suppose that node u sends an update y', but $y' \neq y$.

case 1: $y > y'$: For node i, set f_i to $y+\Delta$. For node $j \neq i$, if $y_j \geq y - 2\Delta$, then set f_j to $y - \Delta$.

case 2: $y < y'$: For node i, set f_i to $y-\Delta$. For node $j \neq i$, if $y_j \geq y - 2\Delta$, then set f_j to $y - \Delta$.

It is easy to show that y' is not in $[f_i(t) - \Delta, f_i(t) + \Delta]$, but by construction $f(t) = f_i(t)$. So the update cannot be such an y'.

Now consider the case where u is a node such that the statement holds for all its descendants. We will show that the statement also holds for u. First, we show that u must send an update.

The fact that the statement holds for all the descendants of u implies that each y_j must be the most recent update of some leaf node in the jth subtree of u (rooted at u's jth child node). Let v be the leaf node corresponding to y_i.

case 1: $y > z$. For v, set f_v to $y + \Delta$. For any leaf node $w \neq v$, if $y_w \geq y - 2\Delta$, then set f_w to $y - \Delta$.

case 2: $y < z$. For v, set f_v to $y - \Delta$. For any leaf node $w \neq v$, if $y_w \geq y - 2\Delta$, then set f_w to $y - \Delta$.

Using a similar argument as that in the base case, we can show that u must send an update to its parent. Now suppose that node u sends an update y', but $y' \neq y$.

case 1: $y > y'$. For v, set f_v to $y + \Delta$. For any leaf node $w \neq v$, if $y_w \geq y - 2\Delta$, then set f_w to $y - \Delta$.

case 2: $y < y'$. For v, set f_v to $y - \Delta$. For any leaf node $w \neq v$, if $y_w \geq y - 2\Delta$, then set f_w to $y - \Delta$.

Again by a similar argument as that in the base case, we can show that such an y' cannot be the update message. □

Corollary 3 $\mathsf{ratio}(\textsc{TreeTrack}) = O(h_{max} \log \Delta)$ *with respect to* A_{tree}.

PROOF. Given an input instance I and any algorithm $A \in A_{tree}$, suppose the number of messages between a leaf node (an observer) s_i and its parent node as M_i. It's easy to see that $\mathsf{cost}(\textsc{TreeTrack}, I) \leq h_{max} \sum_{i=1}^{m} M_i$. Meanwhile, any algorithm A' in A_{tree} satisfies $\mathsf{cost}(A', I) \geq h_{max} + \frac{\sum_{i=1}^{m} M_i}{\log \Delta}$ as a direct application of Theorem 1. Thus,

$$\mathsf{ratio}(\textsc{TreeTrack}, I) \leq \frac{h_{max} \sum_{i=1}^{m} M_i}{h_{max} + \sum_{i=1}^{m} M_i / \log \Delta} < h_{max} \log \Delta.$$

Therefore, $\mathsf{ratio}(\textsc{TreeTrack}) = O(h_{max} \log \Delta)$. □

Lemma 8 *Let* $y_1(t) = \phi(g_1(t), \ldots, g_m(t))$ *and* $f(t) = \phi(f_1(t), \ldots, f_m(t))$. *If* $|g_i(t) - f_i(t)| \leq \Delta$ *for all* $i \in [1, m]$, *then it must be* $|y_1(t) - f(t)| \leq \Delta$.

PROOF. We prove this with contradiction, and we illustrate the proof using median ($\phi = 0.5$). Other quantile functions can be similarly proved. To ease the discussion for quantiles, we assume no duplicates. Cases with duplicates can be easily handled with a proper tie-breaker. Let's say $f(t) = f_i(t)$ for some i.

Suppose this is not true, then it must be $|y_1(t) - f_i(t)| > \Delta$. Without loss of generality, let's say $y_1(t) - f_i(t) > \Delta$. This means that $y_1(t) > g_i(t)$, since $|g_i(t) - f_i(t)| \leq \Delta$.

There are two cases. First, consider m is an odd number. Since $f_i(t)$ is the median of $f_1(t), \ldots, f_m(t)$, there must be $\frac{m-1}{2}$ functions $f_{\ell_1}, \ldots, f_{\ell_{(m-1)/2}}$ in $f_1(t), \ldots, f_m(t)$, such that $f_{\ell_i}(t) < f_i(t)$.

Consider $f_{\ell_i}(t)$ for any $i \in [1, (m-1)/2]$, the facts that $|g_{\ell_i}(t) - f_{\ell_i}(t)| \leq \Delta$, $f_{\ell_i}(t) < f_i(t)$ and $y_1(t) - f_i(t) > \Delta$ imply that $g_{\ell_i}(t) < y_1(t)$.

But now there are $(m+1)/2$ functions ($g_{\ell_1}, \ldots, g_{\ell_{(m-1)/2}}$, and $g_i(t)$), from $g_1(t), \ldots, g_m(t)$, that are less than $y_1(t)$, which contracts that $y_1(t)$ is the median of $g_1(t), \ldots, g_m(t)$.

The other case when m is an even number can be argued in the same fashion, as long as median is properly defined (as either the $(m-1)/2$th value or the $(m+1)/2$th value in the sorted sequence). □