Distributed Set-Expression Cardinality Estimation

Abhinandan Das[†] [*] Sumit Ganguly[§] [*] Minos Garofalakis[‡] Rajeev Rastogi[‡]

[†] Cornell University
asdas@cs.cornell.edu

[§] IIT Kanpur
sganguly@cse.iitk.ac.in

[‡] Bell Labs, Lucent Technologies
{minos,rastogi}@bell-labs.com

Abstract

We consider the problem of estimating set-expression cardinality in a *distributed streaming* environment where rapid update streams originating at remote sites are continually transmitted to a central processing system. At the core of our algorithmic solutions for answering set-expression cardinality queries are two novel techniques for lowering data communication costs without sacrificing answer precision. Our first technique exploits global knowledge of the distribution of certain frequently occurring stream elements to significantly reduce the transmission of element state information to the central site. Our second technical contribution involves a novel way of capturing the semantics of the input set expression in a boolean logic formula, and using models (of the formula) to determine whether an element state change at a remote site can affect the set expression result. Results of our experimental study with real-life as well as synthetic data sets indicate that our distributed set-expression cardinality estimation algorithms achieve substantial reductions in message traffic compared to naive approaches that provide the same accuracy guarantees.

1 Introduction

The widespread deployment of wireline and wireless networks linking together a broad range of devices has resulted in a new class of *distributed data streaming* applications. In these applications, rapid update streams originating at tens or hundreds of remote sites are continuously transmitted to a central processing system for online querying and analysis. Examples include monitoring of service provider network traffic statistics, telecommunication call detail records, Web usage logs, financial stock tickers, retail chain transactions, weather data, sensor data, and so on.

[*]Work done while visiting Bell Labs, Murray Hill, NJ.

**Proceedings of the 30th VLDB Conference,
Toronto, Canada, 2004**

An important consideration in the above-mentioned monitoring applications is the communication overhead imposed by the distributed query processing architecture on the underlying network. A naive approach in which every stream update is shipped to the central site for processing can lead to inordinate amounts of message traffic, and thus have a crippling effect on the communication infrastructure as well as the central processor. For instance, monitoring flow level information within AT&T's IP backbone using Cisco's NetFlow tool [1] is known to generate in excess of 500 GBytes of data per day [4]. Clearly, transmitting every flow record to the central network operations center of a large ISP can seriously strain its processing and network resources. As another example, consider wireless sensor networks (e.g., for environmental monitoring, inventory tracking, etc.), where sensors have a very limited battery life, and radio communication is much more expensive in terms of power consumption compared to processing. In order to ensure longer lifetimes for sensor nodes, it is critical to reduce the amount of data transmitted, even if that implies additional processing at the sensor nodes [13, 12, 10].

Fortunately, for many distributed stream-oriented applications, exact answers are not required and approximations with guarantees on the amount of error suffice. Thus, it is possible to trade answer accuracy for reduced data communication costs. For example, consider the problem of detecting distributed denial-of-service (DDoS) attacks by analyzing network flow information collected from an ISP's border routers. In a typical DDoS attack scenario, hundreds of compromised "zombie" hosts flood a specific victim destination with large numbers of seemingly legitimate packets. Furthermore, in order to elude source identification, attackers typically forge, or "spoof", the IP source address of each packet they send with a randomly-chosen address [11]. Consequently, a possible approach for detecting DDoS attacks is to look for sudden spikes in the number of distinct IP source addresses observed in the flows across the ISP's border routers. Clearly, our DDoS monitoring application does not require IP source address counts to be tracked with complete precision. Approximate counts can be equally effective for the purpose of discerning DDoS activity as long as errors are small enough so as to not mask abrupt changes. Thus, depending on the accuracy requirements of the DDoS application, routers only need to transmit a subset of flow records to the central monitoring site.

As another example, consider a Web content delivery service such as that provided by *Akamai* (`www.akamai.com`). In this case, Web sites are replicated at a large number of geographically distributed servers, and users accessing a Web site are automatically redirected to the geographically closest server, or the least loaded server. Here, one might often be interested in tracking (approximately) the number of (distinct) users accessing a Web site (across all servers), the number of users who visit both a Web site A and Web site B, or the number of users who visit Web site A but not B. These statistics can be useful for determining the servers at which to replicate Web sites, deciding which advertisements to display at each Web site, and so on.

The problem of counting the number of distinct IP source addresses or web-site users, as discussed above, are special cases of the more general *set-expression cardinality* estimation problem, which we tackle in this paper. In this more general problem, we are interested in estimating the number of distinct values in the result of an arbitrary set expression over distributed data streams. For example, in the DDoS scenario, we may want to employ the set difference cardinality query $|S - T|$ to detect significant traffic deviations – here, S is the IP source address set for the sliding window spanning the past week (until now) and T is the set of IP source addresses from the week prior to that (e.g., two weeks ago). Similarly, in our Web example, if S and T are the sets of users who visit Web sites A and B, respectively, then the set intersection query $|S \cap T|$ yields the number of users who access both sites A and B.

Prior Work. The tradeoff between answer accuracy and communication overhead for specific classes of continuous queries over distributed update streams was recently studied in [12, 2]. In [12], Olston et al. consider aggregation queries that compute sums and averages of dynamically changing numeric values spread over multiple sources. In their approach, each site is assigned an interval of a certain width such that the sum of site interval widths is less than the application's total error tolerance. Thus, as long as the numeric value at each site stays within the interval for the site, no messages need to be sent by the sites in order to satisfy the application's accuracy requirements. However, in case the value at a site drifts outside the site's interval, the site is required to transmit the value to the central site and make appropriate adjustments to its interval. Reference [2] focuses on the problem of continually tracking the top-k values in distributed data streams; the developed techniques ensure the continuing validity of the current top-k set (at the central site) by installing arithmetic constraints at each site.

Our work is most similar to the above two research efforts, but considers *set-expression cardinality queries* as opposed to the aggregation and top-k queries handled in [12, 2]. As we will see later in the paper, processing set-expression cardinality queries requires substantially new algorithms to be developed for effectively trading off answer accuracy and communication costs in a distributed-streams setting.

Much of the recent work on data streams has focused on developing memory-efficient one-pass algorithms for performing a wide range of computations on a single stream; examples include computing quantiles [9], estimating distinct values [7, 8], set-expression cardinality [6] and frequent stream elements [3]. An exception is [8] where randomized hash-based sampling algorithms are employed to estimate the number of distinct values in a sliding window over distributed streams. However, [8] does not address the issue of optimizing data-shipping costs when guaranteed precision estimates are required to be continually tracked at the central processing site. Our work differs from these existing proposals that are primarily concerned with exploring space-accuracy tradeoffs (mostly for single streams) rather than communication-accuracy tradeoffs in a distributed streams setting.

Our Contributions. In this paper, we focus on the problem of estimating the cardinality of arbitrary set expressions over distributed update streams. Our proposed algorithmic solutions are the *first* to provide provable guarantees on the accuracy of the final set-expression cardinality estimate, while keeping data transmission costs at a minimum. Since set-expression queries subsume the important class of distinct-value queries, our work also constitutes the first attempt at providing low-cost high-quality answers to this latter type of queries in a distributed setting. More concretely, our contributions can be summarized as follows.

• **Distributed Framework for Processing Set-Expression Cardinality Queries.** We develop our solutions in the context of a general framework for guaranteeing precision constraints for set-expression cardinality queries in a distributed setting. In our framework, each site is allocated an error budget which governs when the site communicates stream state information to the central processing site (for estimating set-expression cardinality). Basically, each remote site associates a *charge* with every stream element that is inserted or deleted since stream state was last transmitted to the central site. Only when the sum of element charges at a site exceeds the site's error budget does it communicate the current stream state information to the central site. Our framework allows for flexibility in how elements are assigned charges – methods for computing charges are only required to satisfy certain basic properties needed for correctness in terms of providing the stipulated error guarantees. Obviously, methods that return smaller charges for elements are more desirable since they result in lower communication overhead.

• **Techniques that Incorporate Global Knowledge to Reduce Communication.** In many distributed streaming environments, the frequency distribution of stream elements will be skewed with certain elements occurring more frequently than others. For example, in the flows collected from an ISP's border routers, IP addresses corresponding to popular Web sites like Yahoo, Google, Amazon, etc. will be contained in a disproportionately large number of flows. Now, if such a frequently occurring element is inserted into a stream at a site (where it does not appear previously), then we do not need to charge for it since the element must already be present at the central site, and thus the insert has no effect on the set-expression cardinality at the central site. Similarly, the charge for the deletion of a frequent element can be distributed across all the sites where the element occurs since the element would need to be deleted at all these sites to truly go away. Thus, global knowledge of frequent stream elements can lead to lower overall communication costs due to reduced element charges at each site. We propose protocols for disseminating this global information to the various sites while incurring minimal message overhead.

• **Techniques that Exploit Set-Expression Semantics to Reduce Communication.** We develop schemes that exploit the semantics of set expressions to obtain further reductions in element charges. For example, in the expression $S \cup T$, if an element e is already present in stream S, then inserts and deletes of e from T have no effect on the set-expression result, and thus we do not need to charge for them. We propose a logic-based approach where we capture the conditions for a change in the set-expression result in a boolean formula. Models for the boolean formula then represent scenarios for result changes and are used to compute element charges. Finally, in order to address the (provably required) exponential time complexity of model enumeration, we develop an efficient heuristic for computing element charges whose running time is polynomial in the number of streams.

• **Experimental Results Validating our Approach.** We present the results of an experimental study (with a real-life TCP traffic data set and multiple synthetic data sets) that demonstrate the effectiveness of our distributed algorithms for estimating set-expression cardinality. Our results indicate that, compared to obvious approaches, our estimation algorithms can lead to reductions in communication costs ranging from a factor of 2 (for the real-life data set) to more than 6 (for synthetic data sets) while guaranteeing high precision for the returned estimates.

Note that while our primary focus in this paper is on estimating set-expression *cardinality*, our techniques are quite powerful, and can also be used to approximate set expression *results* (i.e., sets of data elements) at the central site. This can be used by the coordinator to run other potentially complex queries on top of it, which could be more useful than just cardinality queries. For example, in the DDoS scenario, the results could be filtered to identify malicious *hosts*, or in the *Akamai* example, to identify *users* corresponding to certain traffic patterns.

2 System Model

In this section, we describe our distributed update-stream processing architecture and formally define the set-expression cardinality estimation problem addressed in this paper. Consider a distributed environment with $m+1$ sites and n update streams. Stream updates arrive continuously

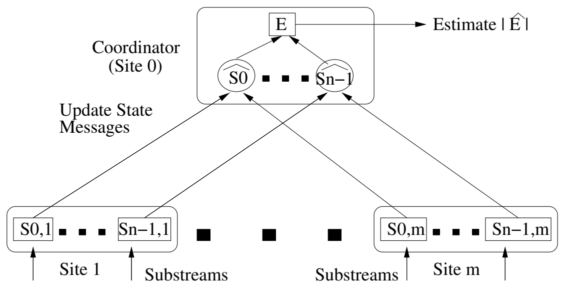


Figure 1: Distributed Stream Processing Model.

at *remote* sites $1, \ldots, m$, and site 0 is a special *coordinator* site that is responsible for generating answers to user (set-expression cardinality) queries. We adopt a similar model to [2, 12] where there is no direct communication among remote sites ; instead, as illustrated in Figure 1, each remote site exchanges messages only with the coordinator, providing it with state information for streams at the site. Note that this distributed communication model is representative of a large class of real-life applications including network monitoring where a central Network Operations Center (NOC) is responsible for processing network traffic statistics collected at the switches and routers distributed across the network.

At each remote site j, the n update streams render n distinct multi-sets $S_{0,j}, \ldots, S_{n-1,j}$ of elements from the integer domain $[M] = \{0, \ldots, M-1\}$. Each stream update at remote site j is a triple of the form $< i, e, \pm v >$, where i identifies the multi-set $S_{i,j}$ being updated, $e \in [M]$ is the specific data element whose frequency changes, and $\pm v$ is the net change in the frequency of e in $S_{i,j}$, i.e., "$+v$" ("$-v$") denotes v insertions (resp., deletions) of e. We assume that all deletions in our update streams are legal; that is, an update $< i, e, -v >$ can only be issued if the net frequency of e in $S_{i,j}$ is at least v. Note that delete operations help to substantially enrich our streaming model; for example, with deletions, we can easily handle sliding window queries by simply issuing a delete operation for each expired stream update that is no longer in the window of interest. For each $i = 0, \ldots, n-1$, let $S_i = \cup_j S_{i,j}$. Thus, S_i reflects the global state of the i^{th} update stream, while each multi-set $S_{i,j}$ captures the local state of stream i at site j. In the remainder of the paper, we will loosely refer to S_i and $S_{i,j}$ as streams even though the intended meaning is the current states of the underlying streams.

Our focus is on the problem of answering set-expression cardinality queries over the underlying collection of distributed update streams. Specifically, given a set expression E over streams $S_0, \ldots, S_{n-1}$ (with the standard set operators $\cup, \cap$, and $-$ as connectives), we seek to estimate $|E|$, the number of distinct elements in E. For example, $|S_0 \cap S_1|$ is the number of distinct elements in the intersection of streams S_0 and S_1. If for $m = 2$ remote sites, $S_{0,1} = \{a\}, S_{0,2} = \{a, b\}, S_{1,1} = \{b\}$ and $S_{1,2} = \{c\}$, then

$S_0 = \{a, b\}$ and $S_1 = \{b, c\}$. Thus, $E = S_0 \cap S_1 = \{b\}$ and $|E| = 1$.

The problem of estimating $|E|$ at the coordinator is complicated in our setting because the substreams $S_{i,j}$ that comprise each distributed stream S_i are distributed across the remote sites. Accurately tracking $|E|$ by having remote sites continuously ship every stream update to the coordinator is clearly impractical for high data rate streams. Consequently, in order to reduce the burden on the communication infrastructure, we allow $|E|$ to be approximated, but enforce a bound on the error in the final estimate. Specifically, for a prespecified error tolerance ϵ, we seek to compute an estimate $\hat{X}$ for $X = |E|$ (at the coordinator) such that $X - \epsilon \leq \hat{X} \leq X + \epsilon$. The ϵ error parameter provides system designers with a useful knob that enables them to trade accuracy for efficiency. Essentially, the larger the error tolerance of an application, the smaller the communication overhead required to ensure that the estimate $\hat{X}$ meets the ϵ accuracy guarantee.

3 Estimating Single Stream Cardinality

We begin by describing our distributed algorithm for the case when the expression E whose cardinality we wish to estimate is a single stream S_i (which is the union of substreams $S_{i,j}$ at remote sites). Thus, we are basically looking to estimate the number of distinct elements in stream S_i. Our scheme for the distinct elements estimation problem illustrates the key concepts underlying our approach as well as the overall structure of our distributed solutions. In the next section, we will generalize our solution for a single stream to handle arbitrary set expressions.

3.1 Overview

Our objective is to be able to continuously estimate $|E|$ at the coordinator with ϵ accuracy. To achieve this, we distribute the error tolerance of ϵ among the m remote sites. We denote the error budget allocated to site j by ϵ_j; thus, $\sum_j \epsilon_j = \epsilon$. While there are multiple ways to allocate error budgets to sites [12], a simple approach is to allocate these proportional to the stream update rates at the sites. The error parameter ϵ_j essentially dictates when site j sends the current states of substreams $S_{i,j}$ at site j to the coordinator. We denote by $\hat{S}_{i,j}$ the most recent state of substream $S_{i,j}$ communicated (by site j) to the coordinator. In addition to $S_{i,j}$, site j also stores in its local memory, the transmitted states $\hat{S}_{i,j}$ for substreams at the site. For each stream S_i, the coordinator constructs the global state $\hat{S}_i$ by taking the union of all the local substream states $\hat{S}_{i,j}$ received from the remote sites. Thus, $\hat{S}_i = \cup_j \hat{S}_{i,j}$. Now let $\hat{E}$ be the result of evaluating expression E on the states $\hat{S}_i$ instead of S_i. The coordinator estimates the cardinality of set expression E as $|\hat{E}|$.

We would like to emphasize here that if remote sites have limited memory, then our scheme can be modified to store a *compact sketch synopsis* for each substream (instead of the complete substream state). Due to space constraints, we describe details of our sketch-based distributed algorithm in [5], and assume that each remote site keeps track of substream states in our current presentation.

In order to guarantee that the estimate $|\hat{E}|$ is correct, we need to ensure that $|E| - \epsilon \leq |\hat{E}| \leq |E| + \epsilon$. A simple approach (based on adapting the scheme of [12]) for ensuring this for $E = S_i$ is as follows. At each remote site j, if either of $|S_{i,j} - \hat{S}_{i,j}|$ or $|\hat{S}_{i,j} - S_{i,j}|$ exceeds ϵ_j, then site j sends the most recent state $S_{i,j}$ to the coordinator. One can easily show that this simple scheme guarantees that at all times, $|E - \hat{E}| \leq \epsilon$ and $|\hat{E} - E| \leq \epsilon$, and is thus correct. For instance, consider an element e in $E - \hat{E}$. The element must belong to $S_{i,j} - \hat{S}_{i,j}$ at some site j, and since $|S_{i,j} - \hat{S}_{i,j}| \leq \epsilon_j$, it must be counted against the error budget ϵ_j at site j. As a result, since $\sum_j \epsilon_j = \epsilon$, we get that $|E - \hat{E}| \leq \epsilon$. Further, since $|E| - |\hat{E}| \leq |E - \hat{E}|$, we obtain that $|E| - |\hat{E}| \leq \epsilon$. Similarly, it is possible to show that $|\hat{E}| - |E| \leq \epsilon$, and thus the estimate $|\hat{E}|$ is within ϵ error of $|E|$.

Intuitively, the simple scheme described above associates a charge $\phi_j(e)$ with each element e at every remote site j, and if the total of these charges exceed ϵ_j, then the remote site communicates state information to the coordinator. More formally, let $\phi_j^+(e) = 1$ if $e \in (S_{i,j} - \hat{S}_{i,j})$, $\phi_j^-(e) = 1$ if $e \in (\hat{S}_{i,j} - S_{i,j})$, and $\phi_j^+(e) = \phi_j^-(e) = 0$, otherwise. As a result, $\sum_e \phi_j^+(e) = |S_{i,j} - \hat{S}_{i,j}|$ and $\sum_e \phi_j^-(e) = |\hat{S}_{i,j} - S_{i,j}|$. Thus, there is a message exchange between site j and the coordinator if either $\sum_e \phi_j^+(e) > \epsilon_j$ or $\sum_e \phi_j^-(e) > \epsilon_j$.

In the simple scheme, element charges are computed based entirely on the local state information available at each site. We next show that by exploiting global knowledge about element e, we can reduce the charge $\phi_j(e)$ for e, and as a consequence, the overall message traffic between remote sites and the coordinator. The key observation we make is that in many stream-oriented domains, there will be a certain subset of globally "popular" elements. For instance, in an IP network monitoring scenario, destination IP addresses corresponding to popular Web sites like Yahoo, Amazon, Google etc. will frequently appear in the flow records collected from network routers. An important characteristic of each such globally popular element is that, at any given point in time, it will appear in substreams at multiple sites although the exact sites that contain the element may vary over time.

Now suppose that for a popular element e, each remote site (approximately) knows the number of substream states $\hat{S}_{i,j}$ that contain e. Specifically, for stream S_i, let $\theta_i(e) > 1$ be a lower bound on the number of sites for which e appears in the $\hat{S}_{i,j}$ states communicated to the coordinator. Then, even if element e is newly inserted into $S_{i,j}$ at site j (that is, $e \in (S_{i,j} - \hat{S}_{i,j})$), we should not charge for it since e is already in $\hat{S}_i$ and, thus, cannot possibly be in $S_i - \hat{S}_i$. Similarly, if e is deleted from $S_{i,j}$ at site j (that is, $e \in (\hat{S}_{i,j} - S_{i,j})$), then in order for e to be deleted from S_i and thus be in $\hat{S}_i - S_i$, e must be deleted from $S_{i,j}$ at least $\theta_i(e)$ sites. Thus, it suffices to charge $\phi_j^-(e) = 1/\theta_i(e)$ (instead of 1) for the local delete of e at each site j. This way, if local deletions at the $\geq \theta_i(e)$ sites cause e to be globally deleted (that is, $e \in (\hat{S}_i - S_i)$), then the cumulative charge $\sum_j \phi_j^-(e)$ for e across the sites is at least 1. As a result, since $\sum_e \phi_j^-(e) \leq \epsilon_j$ at each site j, this total charge of 1 is counted against the various ϵ_js, and correctness is not compromised.

3.2 Distributed Algorithm

We are now ready to describe the details of our distributed scheme for producing a correct cardinality estimate $|\hat{E}|$ at the coordinator. For each element $e \in \hat{S}_i$, the coordinator maintains a count $C_i(e)$ of the number of remote sites whose states $\hat{S}_{i,j}$ contain the element e. Elements whose counts $C_i(e)$ exceed a threshold τ are considered to be *frequent*, and added to a frequent element set F_i for stream S_i. The coordinator also uses the count $C_i(e)$ for each element $e \in F_i$ to compute a lower bound threshold $\theta_i(e)$ such that the invariant $C_i(e) \geq \theta_i(e)$ always holds. It continuously communicates changes in the frequent element sets F_i and the threshold values $\theta_i(e)$ to the remote sites so that these can be used to compute element charges $\phi_j(e)$ at the sites (as described in the previous subsection). Thus, in order to keep the message overhead under control, the coordinator does not send exact element counts $C_i(e)$ to remote sites, but rather disseminates the thresholds, as described in the paragraph below. Each remote site j keeps track of the sum of local element charges $\sum_e \phi_j(e)$ in variable Φ_j. Further, when Φ_j becomes greater than ϵ_j, it sends the deltas $\Delta_i^+ = S_{i,j} - \hat{S}_{i,j}$ and $\Delta_i^- = \hat{S}_{i,j} - S_{i,j}$ that capture the local state changes for substream $S_{i,j}$ since site j last transmitted state information to the coordinator. (Note that the deltas are *sets* and not multi-sets).

Coodinator Actions. Figure 2 depicts the actions performed by the coordinator when it receives the deltas Δ_i^+ and Δ_i^- for substream $S_{i,j}$ from site j. The coordinator employs the received deltas to first update element counts $C_i(e)$ and the stream state $\hat{S}_i$ stored at the coordinator. (Recall that the sets $\hat{S}_i$ are used to generate the final estimate $|\hat{E}|$.) It then uses the new counts $C_i(e)$ to adjust the frequent element set F_i, and the threshold values $\theta_i(e)$ for frequent elements. It also informs all the remote sites of changes to F_i and $\theta_i(e)$ by sending them "make frequent" and "adjust threshold" control messages, which trigger the remote sites to apply the same changes to their local copies of F_i and $\theta_i(e)$. The control messages thus ensure that the values of F_i and $\theta_i(e)$ are synchronized between the coordinator and remote sites.

The correctness of our distributed scheme hinges on the fact that for each element $e \in F_i$, the threshold value $\theta_i(e)$ is always a lower bound on the number of sites j for whom e is in the local state $\hat{S}_{i,j}$ sent to the coordinator. Thus, our scheme for modifying F_i and $\theta_i(e)$ needs to preserve

```
Procedure COORDINATOR($i, \Delta_i^+, \Delta_i^-$)
Input: Newly inserted ($\Delta_i^+$) and deleted ($\Delta_i^-$) elements for some
          substream $S_{i,j}$.
begin
1.  foreach element $e \in \Delta_i^-$ do {
2.      $C_i(e) := C_i(e) - 1$;
3.      if ($C_i(e) = 0$) then $\hat{S}_i := \hat{S}_i - \{e\}$;
4.      if ($e \in F_i$ and $C_i(e) < \tau$) {
5.          $F_i := F_i - \{e\}$;
6.          Send "make infrequent" control msgs for $e$ to all remote sites;
7.      }
8.      else if ($e \in F_i$ and $C_i(e) < \theta_i(e)$){
9.          $\theta_i(e) := \theta_i(e)/2$;
10.         Send "adjust threshold" control msgs with new threshold
11.             $\theta_i(e)$ for $e$ to all remote sites;
12.     }
13. }
14. foreach element $e \in \Delta_i^+$ do {
15.     $C_i(e) := C_i(e) + 1$;
16.     if ($C_i(e) = 1$) then $\hat{S}_i := \hat{S}_i \cup \{e\}$;
17.     if ($e \notin F_i$ and $C_i(e) \geq 2\tau$) {
18.         $F_i := F_i \cup \{e\}$;
19.         $\theta_i(e) := \tau$;
20.         Send "make frequent" control msgs for $e$ to all remote sites;
21.     }
22.     else if ($e \in F_i$ and $C_i(e) \geq 4\theta_i(e)$){
23.         $\theta_i(e) := 2\theta_i(e)$;
24.         Send "adjust threshold" control msgs with new threshold
25.             $\theta_i(e)$ for $e$ to all remote sites;
26.     }
27. }
end
```

Figure 2: Coordinator Actions for Processing Remote Deltas.

the invariant $C_i(e) \geq \theta_i(e)$ while controlling the number of messages between the coordinator and remote sites. Clearly, to maintain the invariant, the coordinator needs to send messages to all sites every time the count $C_i(e)$ drops below the current threshold $\theta_i(e)$ for an element $e \in F_i$. Consequently, in order to prevent minor fluctuations in the value of $C_i(e)$ from generating excessive amounts of control message traffic, our strategy is to try and keep a sufficient gap between $C_i(e)$ and $\theta_i(e)$. Thus, for instance, if $C_i(e)$ becomes less than $\theta_i(e)$, then we simply halve the value of $\theta_i(e)$. Similarly, we double the value of $\theta_i(e)$ only when $C_i(e)$ exceeds $4\theta_i(e)$, and (conservatively) consider an element to be frequent only if $C_i(e)$ exceeds 2τ.

An additional mechanism that we found to be effective for keeping the volume of control messages low (in our experimental study reported in Section 5) is to double $\theta_i(e)$ only after the count $C_i(e)$ is somewhat stable (that is, has stayed above $3\theta_i(e)$ for a certain time period after crossing $4\theta_i(e)$). Using this strategy, we found that the number of control messages is relatively insensitive to the value of the threshold parameter τ. Finally, observe that while increasing $\theta_i(e)$ is not required for preserving the invariant $C_i(e) \geq \theta_i(e)$, larger $\theta_i(e)$ values are key to reducing the charges $\phi_j(e)$ that sites incur for elements.

```
Procedure REMOTE$(i, e, j)$
Input: Update stream $S_i$, element $e$ and site $j$.
begin
1.  $old^+ := \phi_j^+(e)$;
2.  $old^- := \phi_j^-(e)$;
3.  $[\phi_j^+(e), \phi_j^-(e)]$ := COMPUTECHARGE$(e, j, E)$;
4.  $\Phi_j^+ := \Phi_j^+ + (\phi_j^+(e) - old^+)$;
5.  $\Phi_j^- := \Phi_j^- + (\phi_j^-(e) - old^-)$;
6.  if $(\Phi_j^+ > \epsilon_j \vee \Phi_j^- > \epsilon_j)$ {
7.      for $l := 1$ to $n$ do {
8.          $\Delta_l^+ := S_{l,j} - \hat{S}_{l,j}$;
9.          $\Delta_l^- := \hat{S}_{l,j} - S_{l,j}$;
10.         $\hat{S}_{l,j} := S_{l,j}$;
11.     }
12.     Send "update state" message with triples $< l, \Delta_l^+, \Delta_l^- >$
13.         for all substreams $S_{l,j}$ to the coordinator;
14.     foreach element $e$, $\phi_j^+(e) := \phi_j^-(e) := 0$;
15.     $\Phi_j^+ := \Phi_j^- := 0$;
16. }
end

Procedure COMPUTECHARGE$(e, j, E = S_i)$
Input: Element $e$ for whom to compute charge at site $j$,
        expression $E$.
Output: Charges $\phi_j^+(e)$ and $\phi_j^-(e)$.
begin
1.  $\phi^+ := \phi^- := 0$;
2.  if $(e \notin F_i)$ {
3.      if $(e \in (S_{i,j} - \hat{S}_{i,j}))$ then $\phi^+ := 1$;
4.      else if $(e \in (\hat{S}_{i,j} - S_{i,j}))$ then $\phi^- := 1$;
5.  }
6.  else    /* $e \in F_i$ */
7.      if $(e \in (\hat{S}_{i,j} - S_{i,j}))$ then $\phi^- := 1/\theta_i(e)$;
8.  return $[\phi^+, \phi^-]$;
end
```

Figure 3: Remote Site Actions for Computing Charges.

Remote Site Actions. Figure 3 depicts the actions taken by remote site j when an element e is inserted into or deleted from $S_{i,j}$ (due to a stream update), or the frequent set F_i or threshold value $\theta_i(e)$ gets modified (due to a "make frequent" or "adjust threshold" control message for e from the coordinator). Essentially, remote site j computes new charges $\phi_j^+(e)$ and $\phi_j^-(e)$ for e, and appropriately adjusts the total site charges Φ_j^+ and Φ_j^-. Further, if either of these charges exceeds ϵ_j, the deltas for all substreams $S_{l,j}$ are sent to the coordinator; thus, $\hat{S}_{l,j} = S_{l,j}$, and consequently, all charges $\phi_j(e)$ are reset to 0. (Note that sending the deltas for all 'other' substreams to the coordinator is not required when the expression $E = S_i$ since there is only 1 substream at each site, but is needed for the more general set expressions considered in the next section.)

Procedure COMPUTECHARGE in Figure 3 is tailored for the single stream case (that is, $E = S_i$). Later in the paper, we will present alternate charge computation procedures that apply to general set expressions. In a nutshell, COMPUTECHARGE associates a charge of 1 for non-frequent elements that are newly inserted into or deleted from $S_{i,j}$ since the last message to the coordinator. For frequent elements $e \in F_i$, charge $\phi_j^+(e) = 0$ if e is newly inserted, and charge $\phi_j^-(e) = 1/\theta_i(e)$ if e is locally deleted.

Correctness Argument. For ease of exposition, in the arguments pertaining to the correctness of our distributed scheme, we assume that all message transmissions and the actions they trigger are performed instantaneously. While this is clearly not a realistic assumption, our scheme can be extended to simulate such an instantaneous execution (at a logical level) by having sites send special acknowledgements for messages once all the actions triggered by the messages have completed. Details can be found in [5].

The charges $\phi_j^+(e)$ and $\phi_j^-(e)$ computed by COMPUTECHARGE can be shown to satisfy the following two invariants:

$$\text{For each } e \in E - \hat{E}, \quad \sum_j \phi_j^+(e) \geq 1 \tag{1}$$

$$\text{For each } e \in \hat{E} - E, \quad \sum_j \phi_j^-(e) \geq 1 \tag{2}$$

Thus, our distributed scheme is correct because it can be shown (see [5]) that Equation (1) implies that $|E| - \epsilon \leq |\hat{E}|$ and Equation (2) implies that $|\hat{E}| \leq |E| + \epsilon$.

Using Sketch Synopses to Reduce Space/Communication. The space usage of our distributed algorithm can be reduced by storing a compact sketch synopsis for each substream $S_{i,j}$ instead of the entire substream state. Our scheme would then provide probabilistic as opposed to deterministic error guarantees. For instance, we can maintain a (delete-resistant) distinct sample [7] for each substream, and use the substream samples in place of the substream states in our distributed scheme. Due to lack of space, we defer the details of our distinct sample-based estimation algorithms and hash-based techniques for obtaining delete-resistant distinct stream samples to the full paper [5].

4 Estimating Cardinality of Arbitrary Set Expressions

In this section, we generalize our single stream solution (described in the previous section) to tackle the problem of estimating (to within ϵ absolute error) the cardinality of an arbitrary set expression E involving the distributed update streams $S_0, \ldots, S_{n-1}$. Our distributed scheme for general set expressions is identical to the scheme for single streams except for the charging procedure COMPUTECHARGE. Thus, as before, for each stream S_i, the coordinator maintains the states $\hat{S}_i$, the frequent sets F_i, and the threshold values $\theta_i(e)$ for the number of sites j whose shipped state $\hat{S}_{i,j}$ contains element e. The cardinality estimate at the coordinator is $|\hat{E}|$, where $\hat{E}$ is the result of evaluating expression E using $\hat{S}_i$ instead of S_i. The coordinator processes the deltas from a remote site for an arbitrary stream S_i as described in procedure COORDINATOR (see Figure 2). Similarly, site j executes the actions described in procedure REMOTE (see Figure 3) every time there is a change in the substream state $S_{i,j}$, the frequent set F_i, or the local threshold value $\theta_i(e)$.

In the charging procedure for the single stream case, we charged 1 for inserts and deletes of elements $e \notin F_i$, and if $e \in F_i$, inserts were free and deletes were charged $1/\theta_i(e)$. However, when E contains multiple streams, computing the charge $\phi_j(e)$ for an element e is more involved since e may be concurrently inserted/deleted from more than one substream $S_{i,j}$ at site j. A straightforward approach that overcomes this complication is to set the charges $\phi_j^+(e) = \phi_j^-(e) = 1$ if for any of the substreams $S_{i,j}$, either $e \in (S_{i,j} - \hat{S}_{i,j})$ or $e \in (\hat{S}_{i,j} - S_{i,j})$. However, while this straightforward scheme is obviously correct, it is too conservative, and may end up overcharging in many situations. This, in turn, could lead to frequent state transmission messages from remote sites to the coordinator.

Example 4.1 Consider distributed streams S_1, S_2 and S_3, and let expression $E = S_1 \cap (S_2 - S_3)$. For element e at site j, let $e \in \hat{S}_{3,j}$ and $e \in S_{3,j}$. Clearly, $e \in \hat{S}_3$ and $e \in S_3$, and thus $e \notin \hat{E}$ and $e \notin E$. As a result, even if $e \in (\hat{S}_{1,j} - S_{1,j})$ or $e \in (S_{2,j} - \hat{S}_{2,j})$, we should not charge for element e at site j since e cannot possibly be in either $E - \hat{E}$ or $\hat{E} - E$; thus, based on the semantics of expression E, setting the charges $\phi_j^+(e) = \phi_j^-(e) = 0$ will still ensure correctness. ∎

In the following subsections, for an arbitrary set expression E, we focus on the problem of computing the minimum possible charges $\phi_j^+(e)$ and $\phi_j^-(e)$ for a *fixed* element e at site j by leveraging the semantics of expression E. Our proposed charging schemes ensure that charges $\phi_j^+(e)$ and $\phi_j^-(e)$ satisfy Equations (1) and (2) (from Section 3.2), and thus provide an accuracy guarantee of ϵ for the final estimate $|\hat{E}|$. Our first charging method, presented in Section 4.1, is based on enumerating models for a boolean formula corresponding to expression E, and thus has an exponential time complexity. In Section 4.2, we develop a heuristic that at the expense of overcharging in some situations (described later), is able to eliminate model enumeration altogether, and bring down the time complexity so that it is polynomial in the number of streams.

In the remainder of this section, we will say that a stream S_i has a local state change at site j if either $e \in (S_{i,j} - \hat{S}_{i,j})$ or $e \in (\hat{S}_{i,j} - S_{i,j})$. Similarly, we will say that a stream S_i has a global state change if either $e \in (S_i - \hat{S}_i)$ or $e \in (\hat{S}_i - S_i)$.

4.1 A Model-Based Charging Scheme

Our charging procedure first constructs a boolean formula Ψ_j that captures the semantics of expression E and local stream constraints at each site j. It then defines the charge $\phi_j(e)$ at site j in terms of the charges for models M that satisfy Ψ_j.

4.1.1 Constructing Boolean Formula Ψ_j

For each stream S_i, let p_i and $\hat{p}_i$ be boolean variables with semantics $e \in S_i$ and $e \in \hat{S}_i$, respectively. We construct two boolean formulae Ψ_j^+ and Ψ_j^- over the variables p_i and $\hat{p}_i$. Intuitively, Ψ_j^+ and Ψ_j^- specify the conditions that stream states S_i and $\hat{S}_i$ must satisfy for $e \in (E - \hat{E})$ and $e \in (\hat{E} - E)$, respectively. The formulae also capture constraints on S_i and $\hat{S}_i$ due to local knowledge at site j of the substream states $S_{i,j}$, $\hat{S}_{i,j}$, and threshold values θ_i. For example, if $e \in S_{i,j}$, then it must be the case that $e \in S_i$ (since $S_i = \cup_j S_{i,j}$), and thus, variable p_i must be true.

The formulae Ψ_j^+ and Ψ_j^- are built using the following three formulae: (1) an *Expression* formula F_E representing the logic of expression E, (2) *State* formulae $\hat{G}_j, G_j$ that model the local knowledge that site j has about stream states S_i and $\hat{S}_i$, and (3) a *Threshold* formula H that captures the constraints due to the thresholds θ_i for each stream S_i. We describe each of them below.

Expression Formula. The expression formula F_E is constructed recursively as follows.

1. For every stream S_i in E, we replace its occurrence by the boolean variable p_i.
2. The expression $E_1 \cup E_2$ is translated as $F_{E_1} \vee F_{E_2}$.
3. The expression $E_1 \cap E_2$ is translated as $F_{E_1} \wedge E_{E_2}$.
4. The expression $E_1 - E_2$ is translated as $F_{E_1} \wedge (\neg F_{E_2})$.

For example, the set expression $E = S_1 \cap (S_2 - S_3)$ is translated into the boolean formula $F_E = p_1 \wedge (p_2 \wedge \neg p_3)$. It is easy to see that element $e \in E$ iff F_E is true for the stream states S_i. For instance, $e \in S_1 \cap (S_2 - S_3)$ iff $e \in S_1 \wedge (e \in S_2 \wedge e \notin S_3)$. Formula $\hat{F}_E$ is constructed similarly, except that variables p_i are replaced by $\hat{p}_i$.

State Formula. The state formulae G_j and $\hat{G}_j$ are conjunctions of a subset of the boolean variables p_i and $\hat{p}_i$, respectively. Essentially, if $e \in S_{i,j}$, then variable p_i is added to G_j. Thus, G_j captures the constraints on streams S_i for whom we can infer that $e \in S_i$ based on local information that $e \in S_{i,j}$ at site j. Similarly, we construct $\hat{G}_j$ by adding variable $\hat{p}_i$ to it if $e \in \hat{S}_{i,j}$. Note that G_j and $\hat{G}_j$ may be different for the various remote sites depending on the substream states at each site.

Threshold Formula. The threshold formula H only applies to boolean variables $\hat{p}_i$. Basically, if $e \in F_i$ for stream S_i, then we add variable $\hat{p}_i$ to H. Thus, H captures the constraints on stream states $\hat{S}_i$ for whom we can deduce that $e \in \hat{S}_i$ from the frequent element sets. Note that formula H is identical at all sites since F_i is the same at all sites.

We now construct the formulae Ψ_j^+ and Ψ_j^- at site j as follows.

$$\begin{aligned}\Psi_j^+ &= (\neg\hat{F}_E \wedge F_E) \wedge (\hat{G}_j \wedge G_j \wedge H)\\ \Psi_j^- &= (\hat{F}_E \wedge \neg F_E) \wedge (\hat{G}_j \wedge G_j \wedge H)\end{aligned}$$

The formulae Ψ_j^+ and Ψ_j^- comprise two parts; the first part, involving F_E and $\hat{F}_E$, captures the conditions for one of

$e \in (E - \hat{E})$ or $e \in (\hat{E} - E)$ to hold. The second part $(\hat{G}_j \wedge G_j \wedge H)$ specifies the constraints on stream states $\hat{S}_i$ and S_i due to local knowledge at site j of substream states and frequent element sets. Thus, for the boolean formula Ψ_j^+, it follows that $e \in (E - \hat{E})$ iff Ψ_j^+ is true for stream states $\hat{S}_i, S_i$. Consequently, if Ψ_j^+ is unsatisfiable, then it is impossible that $e \in (E - \hat{E})$, and so we can set $\phi_j^+(e) = 0$. Similarly, if Ψ_j^- is unsatisfiable, then charge $\phi_j^-(e) = 0$.

Revisiting Example 4.1 where $E = S_1 \cap (S_2 - S_3)$, and element $e \in \hat{S}_{3,j}$ and $e \in S_{3,j}$, we get that

$$\Psi_j^+ = (\neg\hat{p}_1 \vee \neg\hat{p}_2 \vee \hat{p}_3) \wedge (p_1 \wedge p_2 \wedge \neg p_3) \wedge (\hat{p}_3 \wedge p_3)$$

Obviously, Ψ_j^+ is unsatisfiable (due to $\neg p_3 \wedge p_3$), and thus, charge $\phi_j^+(e) = 0$. In the following subsection, we show how models for Ψ_j can be used to compute the charges $\phi_j(e)$ when Ψ_j is satisfiable.

4.1.2 Computing Charges using Formula Ψ_j

Overview. Let us consider the problem of computing the charge $\phi_j^+(e)$. For an arbitrary boolean formula over $\hat{p}_i, p_i$, we define a *model* to be an arbitrary subset of $\cup_i\{p_i, \hat{p}_i\}$. Each model M basically assigns *truth* values to variables $p_i, \hat{p}_i$ with variable p_i ($\hat{p}_i$) being assigned *true* iff $p_i \in M$ (resp., $\hat{p}_i \in M$); otherwise, p_i (resp., $\hat{p}_i$) is assigned *false*. We say that model M satisfies a boolean formula if the formula evaluates to *true* for the truth assignment specified by M. For example, model $\{\hat{p}_1, p_2\}$ satisfies the formula $\hat{p}_1 \wedge p_2$, but the model $\{\hat{p}_1\}$ does not. Now, each model M represents a specific scenario for states $\hat{S}_i, S_i$. Essentially, $e \in S_i$ ($e \in \hat{S}_i$) iff $p_i \in M$ (resp., $\hat{p}_i \in M$). Clearly, if $e \in (E - \hat{E})$ for stream states $S_i, \hat{S}_i$, then the model corresponding to these states must satisfy Ψ_j^+. Further, every model M that satisfies Ψ_j^+ represents (from the local viewpoint of site j) a possible scenario for states $\hat{S}_i, S_i$ that is consistent with local substream states at site j, and in which $e \in (E - \hat{E})$.

Our model-based approach assigns a charge $\phi_j(M)$ to each model M that satisfies Ψ_j^+ at site j. Furthermore, since as far as site j is concerned, any of these models can potentially occur and cause $e \in (E - \hat{E})$, we set charge $\phi_j^+(e)$ as follows.

$$\phi_j^+(e) = \max\{\phi_j(M) : \text{Model } M \text{ satisfies } \Psi_j^+\} \quad (3)$$

Recall that for correctness, we require that if $e \in (E - \hat{E})$, then $\sum_j \phi_j^+(e) \geq 1$. Thus, by choosing the charge $\phi_j(M)$ for each model M such that $\sum_j \phi_j(M) \geq 1$ if M were to occur, we can ensure that $\sum_j \phi_j^+(e) \geq 1$ if $e \in (E - \hat{E})$ due to some model M that satisfies Ψ_j^+.

Now let us see how to compute the charge $\phi_j(M)$ for a model M that satisfies Ψ_j^+. Let P be the set of streams S_i such that exactly one of p_i or $\hat{p}_i$ belongs to M, i.e., either $\{p_i, \hat{p}_i\} \cap M = \{p_i\}$ or $\{p_i, \hat{p}_i\} \cap M = \{\hat{p}_i\}$. Thus, P is the set of streams that experience a global state change in model M. In our model-based scheme, site j selects a single "culprit" stream S_i from P using a selection mechanism that satisfies the following property.

UNIFORM CULPRIT SELECTION PROPERTY:Given a model M and a set P of streams with global state changes in M, every site selects the same culprit stream $S_i \in P$ for M. ∎

Later in this subsection, we will provide one specific culprit selection scheme satisfying the above property that attempts to minimize the magnitude of the charge $\phi_j^+(e)$ at site j. For the selected culprit stream S_i, let charge $\phi(S_i)$ be defined as follows.

$$\phi(S_i) = \begin{cases} 1/\theta_i(e) & \text{if } e \in F_i \\ 1 & \text{otherwise} \end{cases} \quad (4)$$

Intuitively, the reciprocal of this charge $1/\phi(S_i)$ is the minimum number of sites where stream S_i must have local state changes for it to have a global state change. For instance, if $e \in F_i$, then for e to be in $(\hat{S}_i - S_i)$, e must be in $(\hat{S}_{i,j} - S_{i,j})$ for at least $1/\phi(S_i) = \theta_i(e)$ sites. We define the charge $\phi_j(M)$ for model M in terms of the charge $\phi(S_i)$ for the culprit stream S_i.

$$\phi_j(M) = \begin{cases} \phi(S_i) & \text{if } S_i \text{ has a local state change at site } j \\ 0 & \text{otherwise} \end{cases} \quad (5)$$

Thus, we are able to ensure that if model M indeed does occur, then since the culprit stream S_i has a global state change in M, at least the $1/\phi(S_i)$ sites j at which S_i has local state changes, choose $\phi_j(M) = \phi(S_i)$ and thus, $\sum_j \phi_j(M) \geq (1/\phi(S_i))\phi(S_i) \geq 1$.

Correctness Argument. The correctness of our charging scheme follows from the lemma below.

Lemma 4.2 *Let charge $\phi_j^+(e)$ be computed as described in Equations (3), (4) and (5), and the culprit stream S_i for each model M be selected using a scheme that satisfies the uniform culprit selection property. If $e \in (E - \hat{E})$, then $\sum_j \phi_j^+(e) \geq 1$. (An analogous lemma holds for $\phi_j^-(e)$)* □

Culprit Selection. For a model M, a possible culprit selection scheme is as follows: lexicographically order streams $S_i \in P$ based on charge, index pairs $< \phi(S_i), i >$, and choose the smallest stream in the lexicographic ordering as the culprit. In other words, the culprit stream is the stream with the minimum charge $\phi(S_i)$, with ties being broken in favor of the stream with the smallest index. Clearly, since the charge $\phi(S_i)$ for stream S_i is the same across all the sites, our simple culprit selection scheme satisfies the uniform culprit selection property. Thus, due to Lemma 4.2, our charging procedure is correct. Also, observe that since our charging procedure selects the stream with the smallest charge as the culprit for model M, it minimizes the maximum charge incurred for M across the sites.

Example 4.3 Consider distributed streams S_1, S_2 and S_3, and let expression $E = S_1 \cap (S_2 - S_3)$. At some site j, let the substream states be as shown in the table below.

	$i=1$	$i=2$	$i=3$
$\hat{S}_{i,j}$		e	e
$S_{i,j}$	e	e	

Thus, element e is in all substream states except for $\hat{S}_{1,j}$ and $S_{3,j}$. Also, let $e \in F_3$, $e \notin F_1$, $e \notin F_2$ and $\theta_3(e) = 4$; the meaning here is that e is contained in at least 4 substream states for S_3 transmitted to the coordinator. It follows that $\phi(S_1) = \phi(S_2) = 1$ and $\phi(S_3) = 1/4$. Also, the formula Ψ_j^+ for E at site j is

$$(\neg\hat{p}_1 \vee \neg\hat{p}_2 \vee \hat{p}_3) \wedge (p_1 \wedge p_2 \wedge \neg p_3) \wedge (p_1 \wedge \hat{p}_2 \wedge p_2 \wedge \hat{p}_3)$$

Thus, for any model M that satisfies Ψ_j^+, it must be the case that $\{\hat{p}_3, \neg p_3\} \subseteq M$. As a result, $S_3 \in P$ and since the charge $\phi(S_3)$ for S_3 is the smallest, it is chosen as the culprit for all models. Consequently, since S_3 has a local state change at site j, $\phi_j(M) = \phi(S_3) = 1/4$ for all models M that satisfy Ψ_j^+, and thus, the charge $\phi_j^+(e) = 1/4$. Furthermore, since Ψ_j^- is unsatisfiable, charge $\phi_j^-(e) = 0$.

Now suppose that stream S_3 does not have a local state change at site j, that is, e is neither in $\hat{S}_{3,j}$ nor in $S_{3,j}$. Then, since $e \in F_3$, Ψ_j^+ will remain the same as before, and S_3 will still be chosen as the culprit stream for all models M that satisfy Ψ_j^+. However, since S_3 does not have a local state change at site j, $\phi_j(M)$ will be 0 for the models, and thus charge $\phi_j^+(e) = 0$. ∎

Computational Complexity. In order to determine the complexity our model-based approach, we consider the following decision problem for $\phi_j^+(e)$.

PROBLEM (MAXIMUM CHARGE MODEL): Given expression E, site j, element e, and constant k, does there exist a model M that satisfies Ψ_j^+ and for which $\phi_j(M) \geq k$? ∎

The following theorem can be proved using a reduction from 3-SAT.

Theorem 4.4 *The* MAXIMUM CHARGE MODEL *problem is NP-complete.* ∎

From the above theorem, it follows that since $\phi_j^+(e)$ is the maximum charge for models M that satisfy Ψ_j^+, computing $\phi_j^+(e)$ is intractable.

4.2 Heuristic for Charge Computation

Our model-based charging procedure enumerates all models M in the worst case, and thus, has a worst-case time complexity of $O(2^{2n})$. While this may be reasonable for small values of n (e.g., 3 or 4 streams), the model enumeration-based approach will clearly not scale when set expressions involve a moderately large number of streams, a scenario likely in practice. (e.g. in the *Akamai* case). In this section, we present a heuristic solution for computing the charges $\phi_j^+(e)$ and $\phi_j^-(e)$ for an element e at site j. Our heuristic procedure has a time complexity that is polynomial in the number of streams n, and computes identical charge values as the model-based approach as long as every stream appears at most once in the expression E. However, our heuristic may overcharge for element e in certain cases when there are duplicate occurrences of streams in expression E.

Overview. Our model-based charging procedure essentially computes $\phi_j^+(e)$ as the maximum stream charge $\phi(S_i)$ such that (1) S_i has a local state change at site j, and (2) S_i is the culprit stream for some model M that satisfies Ψ_j^+. (Recall that the culprit stream S_i for model M is the stream with the smallest charge, index pair $< \phi(S_i), i >$ from among streams with a global state change in M.) Thus, for a stream S_i, if we can develop a test for quickly determining if S_i is the culprit stream for some model that satisfies Ψ_j^+, then we can speed up the computation of charge $\phi_j^+(e)$. This is the key idea underlying our heuristic.

Let T denote the expression tree for E with leaves and internal nodes corresponding to streams and set operators in E, respectively. For each node V of T, let $E(V)$ be the subexpression for the subtree rooted at node V, and $\hat{F}_{E(V)}$ and $F_{E(V)}$ be the formulae for $E(V)$ as defined in Section 4.1.1. For example, in the expression tree for $E = S_1 \cap (S_2 - S_3)$, the subexpression for the subtree rooted at $V =$"$-$" is $E(V) = S_2 - S_3$, and $F_{E(V)} = p_2 \wedge \neg p_3$. Now, in order to quickly test if a stream S_i is the culprit stream for some model satisfying Ψ_j^+, our heuristic keeps track of culprit streams (for models) at each node of the expression tree using the notion of *charge triples*. Formally, suppose that M is a model that satisfies the local constraints $(G_j \wedge \hat{G}_j \wedge H)$ at site j. At node V in T, we define the charge triple for model M, denoted by $t(M, V)$, as the triple (a, b, x) with the following values:

- If M satisfies $\hat{F}_{E(V)}$, then bit $a = 1$; otherwise, $a = 0$. Similarly, if M satisfies $F_{E(V)}$, then bit $b = 1$; otherwise $b = 0$.

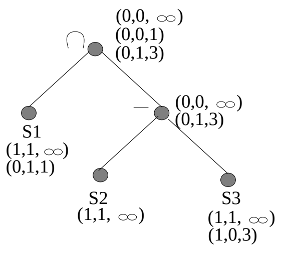
- If none of the streams in V's subtree have a global state change in model M, then $x = \infty$. (The charge, index pair $< \phi(S_\infty), \infty >$ is considered to be greater than $< \phi(S_i), i >$ for all streams S_i.) Otherwise, x is the index of the culprit stream for M in V's subtree; that is, $x = i$, where S_i is the stream with the smallest charge, index pair $< \phi(S_i), i >$ from among streams (in V's subtree) with a global state change in M.

For example, consider a model M that satisfies $\neg\hat{F}_{E(V)} \wedge F_{E(V)}$ (in addition to local constraints). Then, if the culprit stream S_i for M in V's subtree is defined, the charge triple $t(M, V)$ for M at node V is $(0, 1, i)$; otherwise, $t(M, V) = (0, 1, \infty)$. Our charging heuristic computes, in a bottom-up fashion, a set C of charge triples for each node V of T. Furthermore, it ensures that for every model M that satisfies $(G_j \wedge \hat{G}_j \wedge H)$, the computed set C for node V contains the triple $t(M, V)$. Here, it is important to note that the size of C (in the worst case) is linear in the number of streams n – this is because there are at most $O(n)$ distinct charge triples $t(M, V)$ (one for each combination of a, b and x).

Now, consider the charge triple set C for the root V of T. Clearly, since $E(V) = E$, if a model M satisfies $\Psi_j^+ = (\neg\hat{F}_E \wedge F_E) \wedge (\hat{G}_j \wedge G_j \wedge H)$ and has culprit stream S_i, then triple $t(M, V) = (0, 1, i)$ must be in C. Thus, we can quickly determine if a stream S_i is the culprit stream for some model satisfying Ψ_j^+ by checking if C contains the triple $(0, 1, i)$. Hence, by selecting $\phi_j^+(e)$ to be the maximum stream charge $\phi(S_i)$ such that (1) S_i has a local state change at site j, and (2) triple $(0, 1, i) \in C$, we can ensure that $\phi_j^+(e) \geq \max\{\phi_j(M)$: Model M satisfies $\Psi_j^+\}$ and thus, due to Lemma 4.2, our charging heuristic is correct.

Due to lack of space, we defer the details of our bottom-up charge triple computation algorithm to [5] but illustrate its execution in the following example.

Example 4.5 Consider the distributed scenario described in Example 4.3 involving streams S_1, S_2 and S_3, and expression $E = S_1 \cap (S_2 - S_3)$. Suppose that element e is in all substream states except for $\hat{S}_{1,j}$ and $S_{3,j}$, and also $e \in F_3$ and $\theta_3(e) = 4$. Thus, $\phi(S_1) = \phi(S_2) = 1$ and $\phi(S_3) = 1/4$. The following figure illustrates the charge triple sets computed for the nodes of the expression tree for E by our charging heuristic.



The charge triple set for each leaf S_i is first initialized to contain $t(M, S_i)$ for models M that satisfy local constraints. For example, since e is in $S_{1,j}$ but not in $\hat{S}_{1,j}$, it follows that $p_1 \in G_j$ and thus for models M that satisfy $(G_j \wedge \hat{G}_j \wedge H)$, $p_1 \in M$ but $\hat{p}_1$ may or may not be in M; so the charge triple set for S_1 contains the triples $(1, 1, \infty)$ (for models that contain $\hat{p}_1$) and $(0, 1, 1)$ (for models that do not contain $\hat{p}_1$).

Next, the charge triple (a, b, x) for each internal node V is computed by combining pairs of triples (a_1, b_1, x_1) and (a_2, b_2, x_2) from V's two children. Suppose that *op* is the boolean operation corresponding to the set operation for V; the boolean operations for $\cup, \cap$ and $-$ are $\vee, \wedge$ and $\wedge\neg$, respectively. Then $a = a_1$ *op* a_2, $b = b_1$ *op* b_2 and x is set to one of x_1 or x_2, whichever has the smaller charge, index pair $< \phi(S_{x_i}), x_i >$. For example, the charge triples for node "$-$" of T are generated by combining triples for nodes S_2 and S_3. Triples $(1, 1, \infty)$ and $(1, 1, \infty)$ when combined result in the triple $(0, 0, \infty)$ (since $1 \wedge \neg 1 = 0$). Similarly, combining triples $(1, 1, \infty)$ and $(1, 0, 3)$ results in the triple $(0, 1, 3)$ (since $1 \wedge \neg 0 = 1$, and $< \phi(S_3), 3 >$ is less than $< \phi(S_\infty), \infty >$). Finally, the sets for S_1 and "$-$" are combined to obtain the charge triple set C for the root node "$\cap$", which is then used by our charging heuristic to compute the charges $\phi_j^+(e)$ and $\phi_j^-(e)$. Since C contains the triple $(0, 1, 3)$ and S_3 has a local state change at site j, charge $\phi_j^+(e) = \phi(S_3) = 1/4$. Further, since C does not contain a triple of the form $(1, 0, x)$, $\phi_j^-(e) = 0$. ∎

Correctness Argument. The following lemma establishes the correctness of our charging heuristic.

Lemma 4.6 *Consider a model M that satisfies local constraints $(G_j \wedge \hat{G}_j \wedge H)$ at site j. Then, for an arbitrary node V in T, charge triple $t(M, V)$ is in the set of charge triples for V computed by our heuristic.* ∎

Computational Complexity. The maximum size of a charge triple set for a node is $O(n)$, and thus, the worst-case time complexity of our charging heuristic can be shown to be $O(n^2 s)$, where s is the size of set expression E [5].

The following lemma implies that when E contains no duplicate streams, our heuristic returns the same charge values as the model based approach.

Lemma 4.7 *Let E be a set expression in which each stream appears at most once. For an arbitrary node V in T, charge triple t is in the set of charge triples for V computed by our heuristic* if and only if $t = t(M, V)$ *for some model M satisfying $(G_j \wedge \hat{G}_j \wedge H)$ at site j.* ∎

5 Experimental Study

In this section, we present the results of an empirical study of our distributed set-expression cardinality estimation algorithms with real-life as well as synthetic data sets. The main objective of this study is to gauge the effectiveness of our approximation techniques in cutting down the volume of message traffic. Our results indicate that compared to naive approaches, our estimation algorithms can lead to reductions in communication costs ranging from a factor of 2 (for real-life data sets) to more than 6 (for synthetic data).

5.1 Testbed and Methodology

Algorithms for Query Answering. We implemented our distributed algorithm from Section 3.2 where the coordinator executes the actions in procedure COORDINATOR (see Figure 2) to process substream deltas, and each remote site performs the actions in procedure REMOTE (see Figure 3) to detect error violations. In procedure COORDINATOR, we choose the threshold parameter for considering elements to be frequent as $\tau = 4$. In our experiments, we observed that the conservative policy (described in Section 3.2) of "doubling $\theta_i(e)$ only after the count $C_i(e)$ has stabilized" increases the robustness of our algorithm by making the number of control messages virtually independent of the choice of τ. Further, we employ our expression tree-based charging heuristic procedure to compute element charges at each remote site. We will refer to this implementation of our distributed scheme as *Tree-based algorithm*.

To test the efficacy of our tree-based algorithm, we compare it to a naive algorithm in which the coordinator does

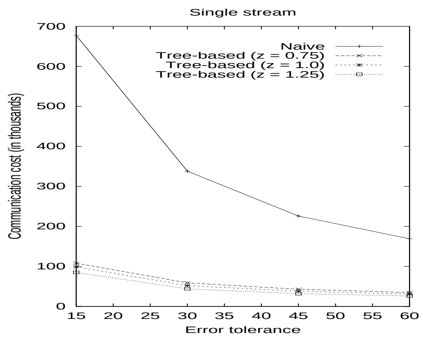


Figure 4: Distinct Values Query.

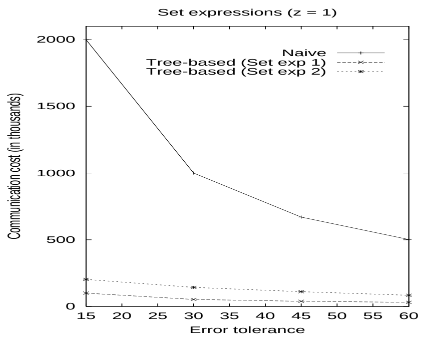


Figure 5: Set Expression Query.

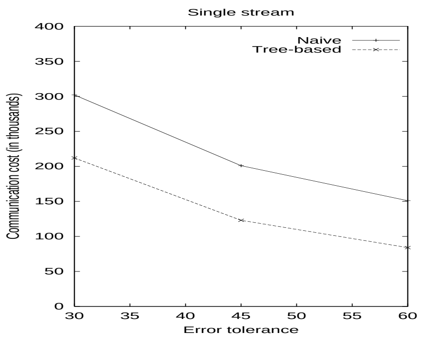


Figure 6: Distinct Values Query.

not send any control messages to remote sites. Instead, each remote site j simply keeps track of the number of elements that have been inserted or deleted from any of the substreams $S_{i,j}$ since stream state information was last communicated to the coordinator. If this element count exceeds the error budget ϵ_j for the site, then it transmits all the substream deltas to the coordinator. Essentially, the naive algorithm adapts the scheme of [12] to our set-expression setting; it considers the charges $\phi_j^+(e)$ and $\phi_j^-(e)$ to be 1 if element e is newly inserted/deleted from any substream at site j completely oblivious of global element frequencies and set-expression semantics.

In the above two algorithms, we distributed the error tolerance budget ϵ uniformly across the m sites; thus, each $\epsilon_j = \frac{\epsilon}{m}$. Recall that ϵ represents the *absolute error* and not the relative error tolerance. For both real-life as well as synthetic data sets, we found the performance of this uniform distribution policy to be comparable to more sophisticated schemes that allocate error budgets to the various sites proportional to stream update rates.

Data Sets. We experimented with multiple synthetic data sets where we varied the frequency distribution for stream elements and one real-life data set.

•*Synthetic data sets.* Our synthetic data stream generator sequentially outputs 1 million stream updates for the n streams at 16 remote sites. For each update, it randomly selects the substream $S_{i,j}$ to be updated at one of the 16 remote sites. The element e for the update is chosen from the domain $[1000] = \{0, \ldots, 999\}$ following a Zipfian distribution. Essentially, the zipf parameter z provides a knob to control the skew in the frequency with which elements in $[1000]$ are updated. If the selected element e is not present in substream $S_{i,j}$, then the update is treated as an insert operation. Otherwise, the update is either an insert or a delete with a slight bias towards deletes to ensure that elements are continuously inserted and deleted from substreams.

•*Real-life data set.* We used the LBL-TCP-3 data set[1] which is a packet trace containing two hour's worth of all wide-area TCP traffic between the Lawrence Berkeley Laboratory and the rest of the world. We considered 500,000 records from the data set, where each record includes a timestamp, source host and destination host field. Even though the trace was collected at a single site, we treat it as if it were collected in a distributed fashion at 16 sites. Thus, each record corresponds to an insert operation for a single distributed stream at one of the 16 sites and whose arrival time is given by the record timestamp. Further, we delete each record using a sliding window of 2 seconds; that is, we issue a delete for each record exactly 2 seconds after its insertion into the stream.

[1]Available from http://ita.ee.lbl.gov/html/contrib/LBL-TCP-3.html.

Performance Metrics. Similar to [2], we use the number of messages exchanged between the coordinator and the remote sites as a measure of the communication costs incurred by the tree-based and naive algorithms. The rationale for this is that in our study, we found message sizes to be generally small (≤ 200 bytes); as a result, the number of messages is an appropriate metric to compare the performance of the two algorithms.

5.2 Experimental Results

5.2.1 Synthetic Data Sets

In our experiments, we compare the message overhead of the tree-based and naive approaches as the skew z in element update rates and the error tolerance ϵ are varied. In the following, we first consider a single stream scenario where our goal is to estimate the number of distinct values in a single distributed stream. This case essentially allows us to isolate the performance improvements realized by our tree-based algorithm as a result of propagating global frequency threshold information. We then turn our attention to general set expressions to further explore the gains obtained due to exploiting set-expression semantics.

Single Stream Cardinality Estimation. In Figure 4, we plot the communication costs for the tree-based and naive algorithms as the error tolerance ϵ is varied. In Figure 4, we consider three values for z (0.75, 1 and 1.25), but only plot a single curve for the naive scheme since the message traffic does not change much as the element update skew is altered. As expected, in the graph of Figure 4, the messaging overhead for both algorithms decreases as the accuracy requirements are relaxed. Furthermore, for all the error and skew values shown in Figure 4, our tree-based algorithm outperforms the naive scheme by a factor of at least 5. The reason is that as elements are randomly inserted and deleted from the various substreams, a significant fraction of them occur at more than $\tau = 4$ sites, and are thus considered to be frequent. Now, for such frequently occurring elements e, our tree-based algorithm propagates the threshold values $\theta_i(e)$ which ensure that inserts of e are 'free' and deletes are charged $1/\theta_i(e)$. In contrast, the naive algorithm charges 1 for both inserts and deletes, and thus, sends many more "update state" messages to the coordinator.

Note that there is a cost associated with disseminating the θ_i values to remote sites in our tree-based algorithm – on an average, we counted the number of such "adjust threshold" control messages sent by the coordinator to be approximately 18 thousand for the 1 million stream updates. Clearly, this is negligible compared to the savings in state transmission messages obtained due to the smaller charge values at sites. In general, control messages (whose counts have been included in all graphs shown) constituted between 20% and 50% of the total message traffic for our tree-based algorithm.

Set-Expression Cardinality Estimation. Figure 5 depicts the number of messages sent by the tree-based and naive algorithms for two set expressions as the error tolerance ϵ is varied between 15 and 60, and skew z is fixed at 1. The expressions we consider are over 3 streams S_0, S_1 and S_2, with the first being $(S_0 - S_1) \cup S_2$, and $(S_0 \cup S_1) \cap S_2$, the second. In the graph, we only plot one curve for the naive scheme since the communication cost was the same for the two set expressions. This is not surprising since the naive scheme does not really care about the structure of set expressions, and simply charges 1 for each element that is inserted/deleted from any of the streams. On the other hand, our tree-based algorithm, by exploiting the semantics of set expressions (in addition to element frequency threshold information), is able to deliver impressive reductions in the data transmission overhead. For the expression $(S_0 - S_1) \cup S_2$, our tree-based algorithm results in factors ranging from 16 (for $\epsilon = 60$) to 20 (for $\epsilon = 15$) lower communication compared to the naive scheme. For the expression $(S_0 \cup S_1) \cap S_2$, the performance improvement factors are halved (since the set-difference operator provides more opportunities to suppress communication as compared to the set-intersection operator), but still lie between 7 and 10.

5.2.2 Real-life Data Set

We compare the communication costs of the tree-based and naive algorithms for the following query over the distributed TCP trace data: How many distinct destination hosts are contained in the TCP trace records within the most recent 2 second sliding window? As shown in Figure 6, our tree-based algorithm incurs between 35% (for $\epsilon = 30$) and 50% (for $\epsilon = 60$) less communication overhead compared to the naive scheme. The reason for the comparatively modest improvement over the naive scheme in this case is the lesser stability in element counts resulting in lower thresholds valid for short durations of time. Also note that our techniques which exploit set-expression semantics did not come into play. It is interesting to note that for our tree-based algorithm, the number of control messages transmitted is actually quite low and ranges between 5% and 20% of the total message traffic.

6 Concluding Remarks

In this paper, we considered the problem of approximately answering set-expression cardinality queries over distributed streams originating at tens or hundreds of remote sites. We proposed novel algorithms for estimating set-expression cardinality with guaranteed accuracy at a central processing site, while keeping data communication costs between the remote sites and the central processor at a minimum. Our solutions exploit global knowledge of the distribution of frequent elements as well as the semantics of set expressions to reduce data transmission overhead while preserving user-specified error guarantees. We developed protocols for efficiently propagating global frequency information across sites, and devised a logic-based formulation for identifying the element state changes (at a remote site) that can affect the set expression result (at the central site). Through experiments with a real-life TCP traffic data set and multiple synthetic data sets, we demonstrated the effectiveness of our techniques in reducing the volume of message traffic compared to naive approaches that provide the same error guarantees.

References

[1] "NetFlow Services and Applications". Cisco Systems White Paper (`http://www.cisco.com/`), 1999.

[2] B. Babcock and C. Olston. "Distributed Top-K Monitoring". In *SIGMOD*, 2003.

[3] M. Charikar, K. Chen, and M. Farach-Colton. "Finding Frequent Items in Data Streams". In *ICALP*, 2002.

[4] C. Cranor, T. Johnson, O. Spatscheck, and V. Shkapenyuk. "Gigascope: A Stream Database for Network Applications". In *SIGMOD*, 2003.

[5] A. Das, S. Ganguly, M. Garofalakis, and R. Rastogi. "Approximating Set-Expression Cardinality over Distributed Update Streams". Bell Labs Tech. Memorandum, 2003.

[6] S. Ganguly, M. Garofalakis, and R. Rastogi. "Processing Set Expressions over Continuous Update Streams". In *SIGMOD*, 2003.

[7] P. B. Gibbons. "Distinct Sampling for Highly-Accurate Answers to Distinct Values Queries and Event Reports". In *VLDB*, 2001.

[8] P. B. Gibbons and S. Tirthapura. "Distributed Streams Algorithms for Sliding Windows". In *SPAA*, 2002.

[9] A. C. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. J. Strauss. "How to Summarize the Universe: Dynamic Maintenance of Quantiles". In *VLDB*, 2002.

[10] S. Madden, M. J. Franklin, J. H. Hellerstein, and W. Hong. "The Design of an Acquisitional Query Processor for Sensor Networks". In *SIGMOD*, 2003.

[11] D. Moore, G. M. Voelker, and S. Savage. "Inferring Internet Denial-of-Service Activity". In *USENIX Security Symposium*, 2001.

[12] C. Olston, J. Jiang, and J. Widom. "Adaptive Filters for Continuous Queries over Distributed Data Streams". In *SIGMOD*, 2003.

[13] G. Pottie and W. Kaiser. "Wireless Integrated Network Sensors". *Communications of the ACM*, 43(5), 2000.