

Estimating Aggregates on a Peer-to-Peer Network

MAYANK BAWA, HECTOR GARCIA-MOLINA, ARISTIDES GIONIS, RAJEEV MOTWANI
 Computer Science Department
 Stanford University
 Stanford, CA 94305 USA
 {BAWA, HECTOR, ARIS, RAJEEV}@cs.stanford.edu

Abstract—

As Peer-to-Peer (P2P) networks become popular, there is an emerging need to collect a variety of statistical summary information about the participating nodes. The P2P networks of today lack mechanisms to compute even such basic aggregates as MIN, MAX, SUM, COUNT or AVG. In this paper, we define and study the NODEAGGREGATION problem that is concerned with aggregating data stored at nodes in the network. We present generic schemes that can be used to compute any of the basic aggregation functions accurately and robustly. Our schemes can be used as building blocks for tools to collect statistics on network topology, user behavior and other node characteristics.

This is a STUDENT paper intended as a REGULAR presentation.

I. INTRODUCTION

In the last couple of years, Peer-to-Peer (P2P) networks have become widely popular and have successfully defined a new class of highly distributed content-sharing applications over the Internet. Most of the content being shared on such networks is unstructured and is not replicated at any central server. Instead, nodes configure themselves into a network and use a common protocol to cooperatively answer queries against the shared content.

As these networks grow in popularity, there is an emerging need to collect a variety of statistical information about the network. For example, an administrator who supports an application on a P2P network needs information about usage trends to tune his particular application across successive rollouts. Here are a few prototypical questions that he may ask at any point in time which motivate this work:

- How much free space does the network have to offer?
- What is the average lifetime of a node in the network?
- How many documents do nodes share on the average?
- Which node(s) performed the most searches?
- What is the maximum number of downloads that some node supported during a session?

Using ‘search’ Queries: The first generation of P2P networks have focussed on supporting just a simple ‘search’ query. The network responds to such a ‘search’ by returning all content that satisfies the query.

Consider the following ALLREPORT scheme that uses ‘search’ to answer the above questions. The querying node q floods its query in the network. Each node that

satisfies the query sends its data (e.g., its available free space, or the number of documents its shares) directly to q . The querying node q receives all the data and computes an answer for its query.

A moment of reflection reveals that ALLREPORT is a flawed scheme. It causes all data relevant to a query to be accumulated at a single node. Worse, it creates a flood of replies to a single node as (potentially) each node in the network responds to the query.

Instead, we would like to answer the query in a distributed fashion, without overloading any node. In some cases, we may also want to probe a “sample” subset of nodes and then “extrapolate” the answer. In this paper, we present such schemes to solve the following problem:

The NODEAGGREGATION Problem: *Devise a scheme to enable any node in a P2P network to issue a query that computes an aggregate function (MIN, MAX, COUNT, SUM, AVG) over data residing at nodes in the network.*

There are two significant hurdles in designing efficient schemes to compute such aggregates. The first challenge arises from the large degrees of distribution. Popular P2P networks currently boast of tens of thousands of simultaneously active peers. Thus the aggregation has to be performed over such large numbers of nodes. The second challenge is posed by the dynamicity of P2P networks. Nodes may subscribe and unsubscribe from the network during the execution of a query. When nodes subscribe, new data is introduced into the network and when nodes unsubscribe, existing data is removed. Such changes to the data set can take place even while the function is being computed. Hence, the definition of a valid answer for the aggregate function becomes unclear.

In this paper, we discuss and define precise notions of validity that a scheme for answering queries must provide. We show that it is impossible to construct a scheme that computes strongly valid aggregates but that a relaxed notion of validity (*practical validity*) is achievable.

We approach the NODEAGGREGATION problem by considering a simple and efficient scheme that aggregates data by building a spanning tree on nodes in the network. However we show that this simple scheme is unable to provide any guarantees in validity. An alternate scheme is then proposed that allows us to obtain the desired validity, albeit at a higher performance cost. We then combine the

efficiency of the first with the correctness of the latter to propose an efficient and accurate scheme.

The last part of this paper focuses on an interesting sub-problem of NODEAGGREGATION called NODECOUNTING: count the number of nodes that are active in the network. We design schemes that can report size estimates quickly and efficiently, albeit at the cost of accuracy.

For both problems, we present experimental results based on real and synthetic network data sets to demonstrate the validity and efficiency of our schemes.

Organization: Section II defines our model for P2P networks. Section III discusses solution desiderata. Section IV presents computability results. Sections V and VI present NODEAGGREGATION and NODECOUNTING algorithms respectively. Section VII discusses related work. Section VIII concludes the paper.

II. THE P2P NETWORK MODEL

Network Topology: The P2P network is represented as an undirected graph $G = (V, E)$, where V is the set of nodes present in the network, and E is a set of edges (u, v) that describes a symmetric neighbor relation (e.g., from using TCP connections) among nodes $u, v \in V$. We use the notation D for the maximum distance along G from any node to any other.

Each node in the P2P network has a unique identifier (e.g., IP-address). The neighborhood of node u in the P2P network is $N(u) = \{v | (u, v) \in E\}$. Each node u maintains a set of identifiers of its neighbors in $N(u)$. Messages can be sent from a node u to another node v provided u knows the identifier of v . In other words, we assume that the P2P network is an overlay on top of an existing physical network (e.g., Internet) and messages can be sent from a node u to any node v whose identifier (e.g., IP-address) is known by u .

Relaxed Asynchrony: We work with a *relaxed asynchronous* model of distributed systems, i.e., there are known upper bounds on process execution speeds, message transmission delays, and clock drift rates. All of the above bounds can be integrated in a known universal maximum delay Δ between any pair of nodes. Thus a message generated at time t at a source node s will be received by the destination node d by time at most $t + \Delta$. In such a setting nodes can, for example, monitor their neighbors for faults using *heartbeats* sent periodically at intervals of time T_{hb} . If a node n does not receive a heartbeat from its neighbor u within $T_{hb} + \Delta$ time of the last heartbeat, then n knows that there must have been a fault at u .

Dynamic Graphs: A P2P network is dynamic if nodes can subscribe or unsubscribe from the network at will. In addition to well behaved unsubscribes, we assume that the network can tolerate a limited number of faults. For example, nodes may simply withdraw from the network without following the unsubscribe protocol. Henceforth, they do not participate in the network protocol and can only rejoin the network by following the *subscribe* protocol.

In real life such behavior might occur when a host computer is switched off. Nodes that exhibit such behavior are called *faulty nodes*. Nodes monitor their neighbors for faults using heart-beats. In case a node n detects a fault at node u , n removes u from its set of neighbors $N(u)$ and recovers from the unsubscribe of u .

Distributed State: Each node stores data on which an aggregate function is to be computed. Solutions that require all data to be stored at a single node are precluded. **Communication Failures:** The presence of communication failures is disregarded. Such an assumption rests on the wide deployment of the TCP/IP and the success of the Internet routing protocols. Thus, we assume that messages are reliably delivered and in order. However, we will be concerned about *overlay network partitions*, where the graph G becomes disconnected due to node unsubscribes or faults, making it impossible for some nodes to communicate with each other along the overlay.

III. SOLUTION DESIDERATA

Generality: The same basic scheme should enable us to compute each of the aggregate functions.

Flexibility: Different P2P protocols result in different topologies. The scheme should work accurately and efficiently regardless of the underlying topology.

Termination: The scheme must guarantee termination with respect to any number of faults that occur during its execution. No dead-locks should occur. We say that an algorithm satisfies *wait-free termination* if in any execution an answer is eventually returned to the querying node.

Correctness: Suppose that a query to compute an aggregate function $F(\cdot)$ was issued by q at time t_0 for which an answer was returned at time t_1 ¹. Let V denote set of nodes in the network at time t_0 , S the set of nodes that subscribed and U the set of nodes that unsubscribed or faulted in the interval $[t_0, t_1]$. Then, an algorithm may be required to ensure one of the following guarantees:

- *Strong validity:* The answer returned is $F(V)$.
- *Relaxed validity:* The answer returned lies in the range $[\min\{F(V-U), F(V \cup S)\}, \max\{F(V-U), F(V \cup S)\}]$.
- *Practical validity:* Let $G_t(V, E)$ be the network topology at time t . Let C be defined as follows: For each node v in C , there is at least one path from q to v along the topology $G_t(V, E)$ that exists throughout the time interval $t \in [t_0, t_1]$. Let I be defined as follows: For each node v in I , there is some path from q to v along $G_t(V, E)$ that was present at some point $t \in [t_0, t_1]$, although not necessarily throughout that interval. Then, the answer returned lies in the range $[\min\{F(C), F(I)\}, \max\{F(C), F(I)\}]$.

We will prove in Section IV that there is no algorithm in the P2P network model that can guarantee either *strong* or *relaxed* validity. The impossibility proofs will also provide an intuition for computability and motivate the definition of *practical* validity.

¹ All times are reported with respect to q .

While the above specify validity of answers, randomized NODEAGGREGATION algorithms will approximate such an answer with varying degrees of accuracy. For such algorithms we must specify bounds on their deviation from validity. Let $[L, U]$ denote a range of valid answers. Then, a randomized algorithm may be required to ensure one of the following guarantees:

- *Approximate accuracy*: For some $0 < \epsilon < 1$, the answer returned is between $(1 - \epsilon)L$ and $(1 + \epsilon)U$.
- *Probabilistic approximate accuracy*: For some $0 < \epsilon < 1$ and $0 < \delta < 1$, the answer returned is approximate accurate with probability at least $(1 - \delta)$.

Performance Measures: We will consider three measures of performance while evaluating the efficiency of an algorithm: *communication*, *computation* and *time*. For randomized algorithms both the communication and computation costs are measured in terms of expected values.

The *communication cost* associated with a NODEAGGREGATION algorithm is the sum of sizes of messages sent between any node pairs (u, v) during its execution. We note that all our algorithms involve messages of a fixed constant size. Hence we will report communication cost as the number of messages sent.

The *computation cost* is a measure of the processing steps incurred by the algorithm. The P2P network consists of processes that are executed independently on each node. The computation cost on a node v is the number of steps taken by such a process on node v . The computation cost of an algorithm is defined as the *maximum* computation cost among all the nodes in the network.

The total elapsed time from query to answer report measures the time needed for both computation and communication among nodes. However, the *computation costs* have already been captured above. For our purposes, the *time cost* refers to the message-passing time. The measure for the time cost of the algorithm then is the length of the longest chain of messages that occurs before the algorithm terminates, where each message in the chain except the first is generated by the receipt of the preceding one.

Assumptions: Some of our schemes will make the following two assumptions about the P2P network.

First, we assume that an upper bound to the diameter of the network can be estimated. This assumption can be justified by the *small world phenomenon* [9] exhibited by information networks and P2P systems: in such systems the diameter grows extremely slowly with the size of the network². Thus a crude overestimate is feasible.

Second, we assume that a value $r > \log n$, where n is the number of nodes in the network, can be estimated. This assumption can be justified by the slow growth of logarithmic function with n . A gross overestimate (say $r = 30$) will be adequate for practically all networks.

²The diameter of a social network was shown to be 6 in 1967 [12], of WWW to be 19 in 1999 [2] and of Gnutella to be 12 in 2001 [16].

IV. COMPUTABILITY RESULTS

The dynamicity of a P2P network implies that the set of nodes that are reachable from q can change even while the aggregate is being computed. The resulting change in data values has implications on the correctness of aggregates and forces negative results that we discuss next.

The following discussion is based on the notion of *graph connectivity*. We say that nodes u and v are *connected* in the graph $G = (V, E)$ if there is a path from u to v along the edges in E . We say that nodes u and v are *c-connected* if u and v remain connected after removing any $c - 1$ nodes in the network, and c is the *largest* such number that provides this property.

Consider a network G with a node v which is 1-connected to the querying node q . The connectivity definition implies that there exists a node u whose removal disconnects v from q . Notice that q cannot communicate with v if u fails. Hence q will be unable to take into account the data values at v during aggregation. Now consider a network G' that has all the nodes and edges in G except node v and edges that end in v . Imagine that a NODEAGGREGATION algorithm is executed on both networks G and G' , and that in both cases node u has failed. It is easy to argue that no algorithm can distinguish between these two networks. However the correct answer is different in each case. We have thus shown the following.

Theorem IV.1: There is no algorithm in the relaxed asynchronous model with reliable ordered communication that solves the NODEAGGREGATION problem while guaranteeing strong validity and wait-free termination. ■

Notice that we can replace the 1-connected node v with an arbitrarily large 1-connected component where arbitrary subscribes and unsubscribes can take place. This implies that we can strengthen Theorem IV.1 by replacing *strong* validity with any of *relaxed*, *approximate* or *probabilistic approximate* validity. Furthermore, we can easily extend the result to networks with larger connectivity.

Theorem IV.1 motivates the definition of practical validity given in Section III. The intuition is that a scheme that explores the network starting from a node q will be able to reach nodes that are in the same connected component as q , and may or may not reach nodes that were intermittently connected with q .

Let us formalise the ALLREPORT algorithm of Section I. The querying node q sets a timer to expire after an interval of length equal to the maximum round-trip time 2Δ , and initiates a broadcast in the P2P network. The broadcast message contains the identifier of q . Each node j that receives the broadcast message sends the data needed to compute the aggregate function directly to q . The querying node q simply computes the function using the data received and resets the timer to 2Δ to allow for similar reporting messages from the neighbors of j . If the timer expires, and no more reporting messages are received at q , the answer obtained so far is reported. It is easy to see that the final answer will include all the nodes that were constantly connected with q and some of the

nodes that were occasionally connected with q . But this is precisely our definition of *practical* validity.

Theorem IV.2: There is an algorithm in the relaxed asynchronous model with reliable ordered communication that solves the NODEAGGREGATION problem while guaranteeing practical validity and wait-free termination. ■

In the ALLREPORT solution outlined above, the number of messages sent is $O(m+n)$. The answer reported by q satisfies practical validity. However, as discussed earlier, the solution is undesirable because it causes the querying node to become a hot spot. In the remainder of this paper, we study solutions that guarantee practical validity while avoiding such hot spots in the network.

V. NODE AGGREGATION SCHEMES

A. SINGLETREE algorithm

We now explore a simple aggregate computation algorithm that works in two phases. In the first phase, the querying node broadcasts the aggregate query to the network. A spanning tree on the network is constructed during this broadcast phase: each node u in the network identifies as its parent the node w that first transmitted the broadcast message to u .

In the second phase, the answer to the aggregate query is computed in a bottom-up fashion, using the spanning tree created at the broadcast phase. Each node waits until it receives an answer from all of its (active) children. The answers are “combined” and the result is sent to its own parent. For each of the aggregate functions, “combining” the answers means simply to apply the function on these answers. The correctness of the final answer is ensured by the commutativity of the aggregate functions.

The above algorithm is called SINGLETREE algorithm. The problem with SINGLETREE is that it is very sensitive to failures. If a node u fails *after* it has received the broadcast and *before* it has sent the result to its parent, the total amount of information of the subtree rooted at u would be absent from the final answer. In other words, the algorithm does not guarantee practical validity, because the information from any node in the network is pushed to the root through a single path; if there is a failure in the path the information about the node would be lost even if there are alternative paths to the root. More precisely:

Theorem V.1: The SINGLETREE algorithm cannot guarantee practical validity even with 1 node failure. ■

The communication cost is $O(m + nr)$, where r is the size in bits of the aggregating values. This is because the broadcast phase results in $O(m)$ messages followed by a bottom up accumulation phase of $O(n)$ values, each of size r . The computation cost of SINGLETREE is $O(b)$ where $b = \max\{|N(u)|, u \in V\}$ is the maximum degree of a node in the network. The time cost of the algorithm is $O(h)$, where h is the height of the resulting spanning tree, which obviously cannot exceed the network diameter D .

B. PROPAGATE2ALL algorithm

The main reason that SINGLETREE is sensitive to faults is that in every tree there is a unique path from each node to the root. Thus, a single node fault will result in information loss for the whole subtree below the fault, i.e., all the nodes whose path to the root contain the faulty node.

This observation immediately suggests creating more paths between each node u and the root, so that if any node on one of those paths fails, the information from u can still reach the root through an alternate path. In the extreme case every possible path can be tried. This is exactly the motivation behind the PROPAGATE2ALL algorithm.

To ensure termination of the algorithm PROPAGATE2ALL we need to assume that an upper bound to the diameter of the network is known. We denote this by \hat{D} . The execution of PROPAGATE2ALL is initiated by the querying node q which broadcasts the query message and the parameter \hat{D} over the network. The set of actions for each node can be described as follows.

- A node that has not received the broadcast message about a query is *inactive* with respect to that query.
- Upon receiving the broadcast message for a query, the node becomes *active* with respect to that query. It continues the broadcasting process and at the same time it sends its own value to *all* its neighbors. The exact data value it sends depends on the aggregate function specified in the query. We will elaborate this later.
- An active node that receives a data value from a neighbor uses the data to recompute its own value. If the node’s value changes, the node resends its new value to *all* its neighbors.
- Each node stops $2\hat{D}\Delta$ time after it became active.
- The querying node announces its current value as the answer for the aggregate query $2\hat{D}\Delta$ time after the instant it initiated the broadcast.

The intuition behind the algorithm becomes apparent when one thinks about computing aggregates MIN and MAX: Say for the MIN, each node sends its value to all of its neighbors. A node keeps the minimum value it has seen so far and when it receives a value less than its current value, it resends the information to all of its neighbors. In particular for MIN and MAX we have:

Theorem V.2: Assume that PROPAGATE2ALL with value \hat{D} is used by the querying node q for computing MIN or MAX aggregates. Then the answer of the query would be correct over all nodes u that are constantly connected with q over a path of length at most \hat{D} . ■

Corollary V.1: The PROPAGATE2ALL algorithm guarantees *practical* validity for MIN or MAX aggregates when used with value $\hat{D} \geq D$. ■

The hurdle in using this algorithm for the aggregate functions COUNT and SUM is that the value from each node travels to q through multiple paths. If the intermediate nodes keep adding all the values they receive, the final answer would be an overestimate of the correct answer.

We now propose an approach that allows us to use

PROPAGATE2ALL for COUNT and SUM while ensuring correctness of result. In particular, we adapt the algorithm (FM) proposed by Flajolet and Martin [5] to estimate the number of distinct items in a multiset. We first describe FM and then its adaptation for NODEAGGREGATION.

The FM algorithm takes as input a set V of size n and a sequence M of values drawn from V , and estimates the number of distinct elements in M . It takes a user parameter $r > \log n$ and produces an estimate using a vector B of r bits. Before counting, the algorithm generates a random function $map : V \mapsto [0, r]$ with an exponential distribution: half the elements in V are mapped to 0, a quarter to 1, one-eighth to 2, etc. At the start of counting, B is initialized to 0. The algorithm makes a single pass over M and for each element $v \in M$, the corresponding $h = map(v)$ bit in B is set to 1. At the end of the pass, the lowest-order bit z in B that is still 0 is identified. The procedure is repeated c times, and the corresponding values z_1, \dots, z_c are recorded. Finally, the average value $\bar{z} = \sum_1^c z_i/c$ is computed and $2^{\bar{z}}$ is returned as the answer.

Theorem V.3: For every $c > 2$, given a sequence M of elements from a universe V of size n , the FM algorithm estimates the number of distinct elements in M such that the probability that the estimate has multiplicative error greater than c is at most $2/c$ [3]. ■

We now adapt the FM algorithm for computing a COUNT aggregate with PROPAGATE2ALL:

The querying node includes parameters $r > \log n$ and c in the broadcast message. When a node v changes its state from inactive to active, v decides if it has data that must be included in the count. If v decides to contribute to the count, v generates its $h = map(v)$ value by tossing upto r coins and recording the index of the first toss h that resulted in heads. It then generates a bitvector B with all bits 0 and sets the h^{th} lowest-order bit to 1. If v decides not to contribute to the count, it merely generates a bitvector B with all bits 0. Each node now propagates its bitvector to its neighbors as specified by PROPAGATE2ALL. For each bitvector that v receives, v performs a logical OR on its own bitvector, and propagates the result if its bitvector changes. The querying node (after time $2\hat{D}\Delta$) identifies the lowest-order bit z that is still 0 in its bitvector.

The above procedure has to be repeated c times. Instead of a single bitvector, each node generates c separate bitvectors B_1, \dots, B_c . These bitvectors are passed around as part of a single message as before. At the end, the querying node identifies the lowest-order bits z_1, \dots, z_c that are still 0 in its bitvectors B_1, \dots, B_c , computes the average value $\bar{z} = \sum_1^c z_i/c$ and reports an estimate of $2^{\bar{z}}$.

Combining Theorem V.3 with the fact that the count information from each node reaches the querying node through all possible paths, we get:

Theorem V.4: Assume that the algorithm PROPAGATE2ALL is executed for the functions COUNT with value $\hat{D} \geq D$. Then, for every $c > 2$, the probability that the algorithm fails to guarantee practical validity within a factor of c is at most $2/c$. ■

For the SUM aggregate assume that the values we add are in the range $[0 \dots R]$. Let $s \in [0 \dots R]$ be the value of node v . Computing SUM is equivalent to computing COUNT, where v pretends that it has s different distinct values. The maximum value of the sum is nR , so we can use the COUNT algorithm with bitvectors of size $r > \log(nR) = \log n + \log R$. A naive simulation of FM would require node u to flip s different coins for each bit, but in fact we can do it with only one biased coin flip. Notice that after running FM s times and OR-ing bitvectors, the probability that the leftmost bit remains 0 is $Pr[b_1 = 0 \text{ after } s \text{ trials}] = \prod_{k=1}^s Pr[b_1 = 0 \text{ in the } k\text{-th trial}] = (1/2)^s$. Similarly, for the j -th bit, $Pr[b_j = 0 \text{ after } s \text{ trials}] = (1 - 1/2^j)^s$. So v can simulate FM for SUM with just one biased coin flip per bit.

Every node v can cause an update to the value of a specific node u . On the other hand, an update on u caused by v can happen at most once, hence the computation cost of PROPAGATE2ALL is $O(n)$. For the communication cost, notice that any update on the value of a node u causes a retransmission to all of u 's neighbors. Therefore, the total communication cost is $O(\sum_{u \in V} n|N(u)|r) = O(nmr)$, assuming messages of size r . The time cost is $O(\hat{D})$.

C. MULTIPLE TREES algorithm

An obvious disadvantage of PROPAGATE2ALL is the high computation and communication cost that it incurs. Another disadvantage is that it assumes a knowledge of D . In this section we describe an algorithm, called MULTIPLE TREES, which overcomes these disadvantages.

Informally, MULTIPLE TREES creates k (2 or 3) independent spanning trees rooted at the querying node, so that the information from each node reaches the root through many different paths.

Unfortunately, MULTIPLE TREES does not satisfy practical validity. In practice MULTIPLE TREES can be expected to be much more robust to failures than SINGLE TREE without incurring any significant increase in costs. Thus, MULTIPLE TREES can be thought of as the ‘‘middle path’’ between SINGLE TREE and PROPAGATE2ALL.

The broadcast can be viewed as a BFS in the network graph, so each node v is associated with its level $l(v)$, which is the length of its shortest path to the root and will be computed during the BFS. Initially, the root q has $l(q) = 0$ and every other node v has $l(v) = \infty$.

The k independent spanning trees are created as follows. As before, the querying node q initiates the aggregation query with a broadcast, with the value of k and $l(q) = 0$ contained in the broadcast message. When a node v receives the query from a node u with level $l(u)$, v checks if its own level $l(v) = \infty$. If so, v sets its own level $l(v) = l(u) + 1$ and forwards the query with level $l(v)$ to all its neighbors. Otherwise, v stores the level of u as $l(u)$. We say that a node is *finished* if its l -value and the l -values of all its neighbors are less than ∞ . Define $P(u) = \{v \in N(u) \text{ such that } l(v) < l(u)\}$ to be the set of *potential parents* of a node u . Every finished node

picks uniformly at random k parents from the set $P(u)$, one parent for each tree, and in this way k independent spanning trees are formed.

The algorithm proceeds by computing the aggregate function in a bottom up fashion on the k spanning trees. Each node waits until it receives the data values from all of its children in all of the trees. Then the node aggregates the data values and sends the results in all of its parents in the k trees. The data values and the aggregation procedure used by each node is the same as in PROPAGATE2ALL. Thus, for MIN and MAX we use just the values over which the functions are computed. For COUNT and SUM we use the bitvectors as specified by the FM scheme.

The computation cost of MULTIPLE TREES is $O(k + b)$ since each node needs to update its value b times and pass it to at most k parents. The k messages sent by every node, causes the communication cost to be $O(m + knr)$, where r is the length value used (exact value or bitvector size for the probabilistic schemes). The time cost is $O(D)$.

D. Discussion

Flexibility: All the algorithms are independent of the underlying network topology. PROPAGATE2ALL requires a knowledge of an upper bound on the network diameter.

Generality: SINGLE TREE can be used to compute any aggregation function. PROPAGATE2ALL and MULTIPLE TREES compute exact answers for MIN and MAX, but provide approximate answers for COUNT, SUM and AVG as they rely on probabilistic counting techniques.

Hot Spots: None of the algorithms create hot spots since each node receives messages only from its neighbors.

Termination: All of the proposed algorithms guarantee wait-free termination.

Correctness: SINGLE TREE is very sensitive to faults. PROPAGATE2ALL guarantees approximate practical validity. For MULTIPLE TREES it is not possible to obtain a guarantee of practical validity. However, it is expected to perform better than SINGLE TREE.

Performance: The performance of the three algorithms (SINGLE TREE (ST), MULTIPLE TREES (MT) and PROPAGATE2ALL (P2A)) is summarized in the following table. Recall that b is the maximum degree of a node and D is the network diameter.

Measure	ST	MT	P2A
Comp. cost	$O(b)$	$O(k + b)$	$O(n)$
Time cost	$O(D)$	$O(D)$	$O(\hat{D})$
Comm. cost	$O(m + nr)$	$O(m + knr)$	$O(mnr)$

E. Experimental Evaluation

In this section, we present results from our simulation studies of the various NODEAGGREGATION algorithms on dynamic P2P networks. We show the interplay of various constraints (*validity, accuracy, and costs*).

The topology of the network G was established before the start of the simulation. A GNUTELLA topology with

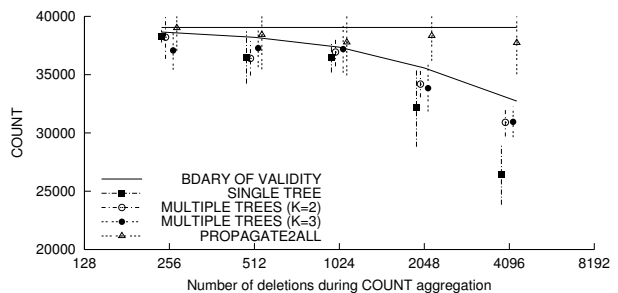


Fig. 1. Accuracy for COUNT query on the GNUTELLA topology

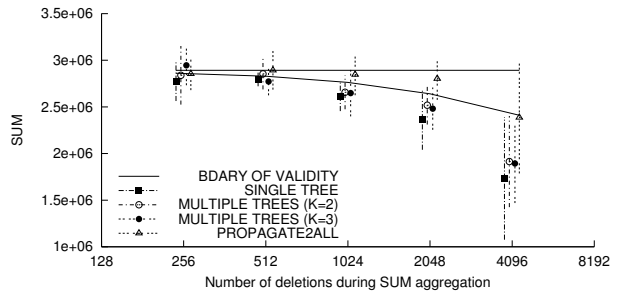


Fig. 2. Accuracy for SUM query on the GNUTELLA topology

$n = 39,046$ nodes is a real-life network obtained from a snapshot of the Gnutella network [1].

Each node stores a data item that has to be aggregated by the algorithm. The data values are drawn from a ZIPF distribution in the range $[1, 500]$ and assigned to nodes at random. We note that a ZIPF distribution has been observed in practice for quantities (e.g., the number of documents at a node) that a network administrator would be interested in aggregating.

We model node departures and failures by removing a total of r randomly selected nodes from the network. The nodes are removed at a uniform rate during the execution of a NODEAGGREGATION algorithm. The value of r is varied from 256 to 4096 to investigate the effect of network dynamicity on our algorithms. Note that practical validity ordains that the aggregate function be computed over data values at nodes that are *always* connected to the querying node. Thus nodes that join the network *after* the issue of a query do not affect the validity of the reported value. Hence we do not model node arrivals in the reported set of experiments.

As a frame of reference, an ORACLE was devised that could observe all events occurring in the network. The ORACLE detected the reachability of each node in the network from the querying node q . Let C be the set of nodes that were always connected to q . Let I be the set of nodes that were intermittently connected to q . The ORACLE computed $F(I)$ and $F(C)$ as the lower and upper bounds of practical validity for each run of a NODEAGGREGATION algorithm. Clearly, such an ORACLE is not feasible in practice.

The PROPAGATE2ALL algorithm requires an estimate of the network diameter \hat{D} as a parameter. In our experi-

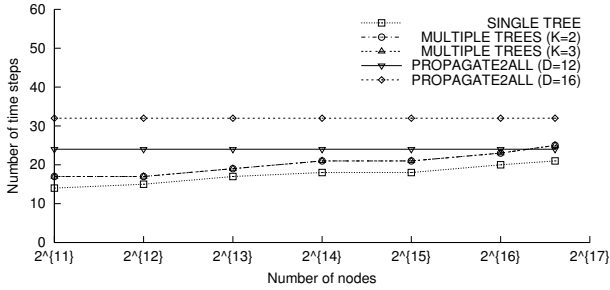


Fig. 3. Time costs for queries on the RANDOM topology

ments, \hat{D} was set to 14 for GNUTELLA topology. We note that the actual diameter of the topology was 10.

Accuracy: Figures 1 and 2 correspond to COUNT and SUM queries over the GNUTELLA topology respectively. The X-axis depicts the number of nodes that were removed during the run of an algorithm. The Y-axis plots the answers for the query. The mean values returned by the various algorithms are shown as points in the graph, with the dotted lines showing the variance observed. The solid lines in the figures show the upper and lower ranges for practical validity as computed by the ORACLE. Since we only delete nodes the upper limit of the practical validity range remains constant.

We observe that all the algorithms report similar mean values for the case of low deletion rates, although the randomized algorithms have higher variance. This is a consequence of FM being a randomized counting scheme. For higher deletion rates, however, the randomized algorithms remain closer to the ORACLE range, with similar variance as before. Notice that PROPAGATE2ALL is always within the limits of practical validity. On the other hand, the accuracy of SINGLETREE (both mean and variance) degrades as the number of deletions increase.

Performance: The performance (time and communication) costs of all our NODEAGGREGATION algorithms are independent of the particular aggregate function being computed and the distribution of values in the network but depend merely on the size of the network. Since we did not have access to Gnutella snapshots of varying sizes, we used synthetic topologies to study performance. A RANDOM topology with n nodes is a topology constructed by connecting two nodes in the network with uniform probability p_e . For our experiments we selected p_e as a function of n to ensure that the average degree of a node is 5.

Figures 3 and 4 show the time and communication costs of the algorithms as a function of the network size. The X-axis depicts the number of nodes in the network. The Y-axis plots the costs (time and messages respectively) incurred in computing the aggregate. The mean values of the costs incurred are shown as points in the graph; the variance is not shown for clarity. Since PROPAGATE2ALL algorithm requires an estimate of the network diameter \hat{D} as a parameter, we simulated the algorithm with different values to study the effects of \hat{D} on the cost metrics. The

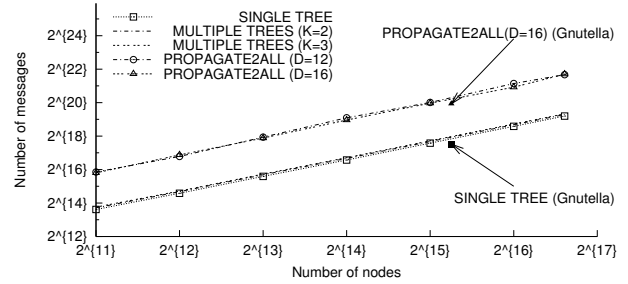


Fig. 4. Communication costs for queries on the RANDOM topology

MULTIPLETREES with $k = 2, 3$ are also shown.

We see that the extra costs incurred by MULTIPLETREES compared to SINGLETREE in terms of both communication and time is small. On the other hand, PROPAGATE2ALL communicates $4\times$ more messages across the network, the price for its improved accuracy.

We also observe that overestimating D hurts only the time costs for PROPAGATE2ALL. The number of messages sent is *independent* of $\hat{D} > D$. The reader may recall that in PROPAGATE2ALL nodes do not send messages to their neighbors unless their own value changes. The values at nodes stabilize within $2D$ time: the first D units are needed for query broadcast, and the next D for values from the farthest nodes to percolate to q . After $2D$ time, no more messages are sent in the network, causing the communication costs to be independent of \hat{D} .

The solid points in Figure 4 depict the costs incurred by SINGLETREE and PROPAGATE2ALL ($D=16$) on GNUTELLA. These are close to costs observed on RANDOM topologies, suggesting that the algorithms will have an analogous scaling behavior on real-life networks.

VI. ESTIMATING SIZE OF THE NETWORK

We now turn to an interesting sub-problem of NODEAGGREGATION: estimate the current size of the P2P network (NODECOUNTING). An accurate estimate would be useful in administering the network and as part of the P2P protocol itself. For example, the protocol in [11] requires such an estimate during the insert of a node in the network. While the general NODEAGGREGATION schemes of Section V can be used for NODECOUNTING, this section focuses on schemes that are fast and efficient at the cost of generality and accuracy.

A. Report to the Querying Node

In the ALLREPORT algorithm outlined in Section IV, the querying node became a hot spot as it has to process $O(n)$ messages. This drawback can be alleviated with the following solution called RANDOMIZEDREPORT.

The querying node q initiates a broadcast of a message that includes a sampling parameter p , and sets its timer to expire after an interval $\hat{D}\Delta$ so that the broadcast message reaches all the nodes in G . Each node that receives the

counting message replies to the querying node with probability p or does not reply to the querying node with probability $1-p$. In either case, the node passes the message to its neighbors as specified by the broadcast protocol. The querying node counts the number of replies received until the timer expires, and then declares the answer after scaling the final count by $\frac{1}{p}$.

This technique reduces the number of replies sent back to the querying node by a factor of p . The computation cost for RANDOMIZEDREPORT is $O(n)$ at the querying node q , and the communication cost is $O(m + np)$. The expected number of replies to the querying node is $O(np)$. Since the nodes choose to reply to q independently at random, the accuracy of the estimate obtained from the above randomized algorithm can be made sufficiently high as we show in Theorem VI.1. In fact, by picking p appropriately we can guarantee probabilistic approximate validity for any choice of the parameters ϵ and δ (Section III).

Theorem VI.1: The algorithm RANDOMIZEDREPORT guarantees probabilistic approximate practical validity, provided $p \geq \frac{4}{\epsilon^2 n} \ln \frac{2}{\delta}$. The variance of the estimate is $n(1-p)/p$. ■

Notice that for $p \geq \frac{4}{\epsilon^2 n} \ln \frac{2}{\delta}$ the expected number of responses will be a constant, i.e., it does not depend on n , but only on the precision parameters ϵ and δ . To select a value of p that satisfies the condition in the above theorem, we need to know n . This might seem like a cyclic problem. However, if the querying node has a rough idea on the order of magnitude of the answer (say, an estimate on a *lower bound* on n), it can set the parameter p so that it will achieve an answer within a certain accuracy.

If q sets p too low, the variance of the final answer increases. Similarly, q must have a good estimate \hat{D} of the network diameter. If q underestimates D , then q might terminate its counting process while the broadcast is in progress causing it to underestimate the network size.

The communication cost for RANDOMIZEDREPORT is $O(m + np)$, the computation cost is $O(np)$, and the time cost is $O(D)$. If q overestimates D , the time cost of RANDOMIZEDREPORT becomes T_d .

B. Inverting the Birthday Paradox

We now describe *sublinear* methods for NODECOUNTING. The key idea behind such algorithms is to use random samples for computing an appropriate statistic which depends on the sample space size. The size of the network can then be estimated from the value of the statistic.

The *birthday paradox* states that for \sqrt{n} independent samples from a population of size n , the probability that a pair of samples will have the same value is at least $1/2$ [13]. In a more useful formulation of the paradox, if we sample repeatedly, with replacement and uniformly at random, from a population of size n , the number of trials required for the first repetition of a sampled value has expectation $\sqrt{2n}$ and variance $\Theta(\sqrt{n})$ [13].

We obtain the following simple algorithm schema (BDAYPARADOX) for estimating the size of the network:

Keep sampling random nodes until a node is sampled twice for the first time. Let X_r be the total number of nodes sampled. The estimate for the size of the network is then $\bar{n} = \frac{X_r^2}{2}$.

The sampling of random nodes can be implemented using *random walks*. The walk starts from the querying node q ending at some node n in the topology after r hops. The token-possessing node v is then picked as a sample. A number of such samples can be picked simultaneously by performing the random walks in parallel.

The theory of random walks has been the subject of wide study. A well-known result [13] states that if we perform a random walk of r hops on an undirected graph for sufficiently large r , the probability that we end at a particular node v is proportional to $|N(v)|$. Using this property, and normalizing appropriately by degrees of nodes, we can obtain uniform random samples of network nodes. For example, it can be shown that for an n -node graph that is an *expander*³, we need to take only $r > \log n$ steps before we reach a uniformly random node.

To keep track of the sampled nodes and to identify the first repetition, each sampled node sends a message with its id to q . The querying node q stores the sampled node ids in a hash table of $\Theta(\sqrt{n})$ space, probing and storing a new sample in the hash table in expected $O(1)$ time. The total computation requirements are thus $\Theta(\sqrt{n})$.

By repeating the process a constant number of times we can obtain an estimate which is a close approximation of the true size of the network, with only a small probability of error. In particular, we have the following result:

Theorem VI.2: Let ϵ be a small positive numbers. Let n be the number of nodes in the network and X be an estimate produced by repeating the BDAYPARADOX algorithm $O(\frac{1}{\epsilon^2})$ times. Then, the relative error of the estimate, defined as $\frac{|X_n - n|}{n}$, is less than ϵ with high probability. ■

The communication cost for BDAYPARADOX is $O(\sqrt{n} \log n)$, the computation cost is $O(\sqrt{n} \log n)$ at q , and the time cost is $O(\log n)$. Moreover, it guarantees Approximate Practical Validity with a variance of $O(\sqrt{n})$.

C. Random Increasing Walks

The communication cost of BDAYPARADOX is $O(\sqrt{n} \log n)$. While sublinear in n , the constant factors could make the cost to be close to n for $n \leq 10,000$. The reason is the \sqrt{n} factor which does not decrease fast enough to make the algorithm efficient for small n .

We now present a strategy RANDOMINCWALK with a cost that grows logarithmically with n , giving a practical sublinear strategy. On the flip side, we have been able to analyze the strategy only for *random-graph* network topologies. A *random graph* formed on n nodes when the edge between each (unordered) pair of nodes is present with probability p independent of other edges. Even for relatively small values of $p = c \log n/n$, random graphs

³Several P2P protocols build such a topology [8], [15].

have properties that make them desirable as topologies for P2P networks — a (small) diameter of $O(\log n)$ enables fast search using broadcast, a (high) probability of connectivity enables resilience to malicious attacks [8], etc.

The strategy described in this section involves performing a *random increasing walk* on the topology. A token is passed from one site to another of a higher id until the token cannot be passed any further. The length l of the path traced by the token is recorded, and the process repeated a number of times. In the end, the average \tilde{l} of the path lengths is computed and the estimate $e^{\tilde{l}}$ for n is declared as the answer.

To present the strategy in a simple setting, consider a complete topology, where each sites is connected to every other site. Start with the token at the node with the smallest id value. At any point in time, the token will be at some node v which will pass the token to a neighbor chosen uniformly at random from amongst those which have id greater than that of v itself. Clearly, the process terminates at a node with the largest id value. The following lemma states that the expected length of such a random increasing walk is $O(\ln n)$.

Theorem VI.3: Assume, without loss of generality, that node ids are drawn from the set $\{1, \dots, n\}$. Let random variable L denote the number of steps in a random increasing walk starting at node 1 and reaching node n . Define $A_n = E[L]$; then, $A_n = H_{n-1}$, where $H_0 = 0$ and H_n is the n th Harmonic number for positive integer n . ■

The procedure requires that we start the walk from the node with the smallest id. In general, it is not easy to find and maintain a record of the lowest id in the network. The work-around is simple: along with the token, the querying node passes its own identifier q . Henceforth, each identifier in the network (apart from q) is considered after adding q to the identifier. Effectively, q now becomes the least identifier in the network. Note that the strategy does not impose any range on the site identifiers, merely that they be linearly ordered and unique.

The same strategy can be applied to a random graph, with the edge probability parameter p_e . The analysis yields a recurrence of the form:

$$A_n = 1 + (1 - (1 - p_e)^n) \sum_{j=1}^{n-1} \frac{A_j}{n-1}$$

If p_e is known, and the average path length L is determined, successive A_i can be generated using the recurrence to find an A_n that closely approximates L . Then, n can be announced as an estimate of the network size.

Estimating p_e accurately is a problem in itself. A simple technique involves gathering a random subgraph from the bigger graph, and deducing p_e from the probability of an edge in the sample subgraph. The problem is that if p_e is small, the sublinearity gains in the counting strategy might be lost as the estimation of p_e becomes expensive.

In general, the form of the recurrence depends on the topology involved. We believe that similar recurrences can be derived for *small-world* and *power-law* topologies.

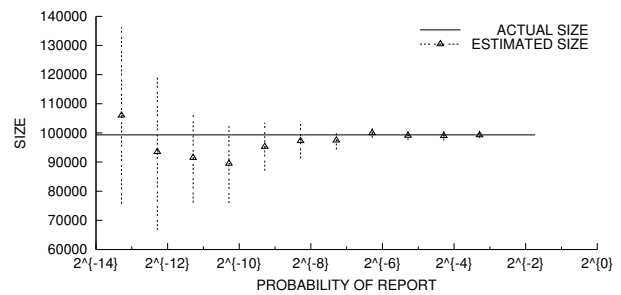


Fig. 5. Accuracy of RANDOMIZEDREPORT

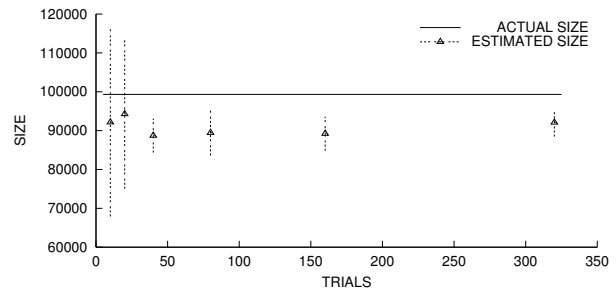


Fig. 6. Accuracy of BDAYPARADOX

D. Experimental Evaluation

We simulated the NODECOUNTING algorithms described in this section over the RANDOM topology discussed in V-E. In this set of simulations, the RANDOM topology is a synthetic network with $n = 100,000$ nodes formed by connecting two nodes in the network with uniform probability p_e . The parameter p_e is set such that the average degree of a node is 5.

Note that all of our algorithms in this section are probabilistic. The main point of our experimental study was to investigate how the quality of the estimation improves as we allow the algorithms to perform more random trials and therefore improve their confidence.

For the RANDOMIZEDREPORT algorithm, we increase the bias p with which a node replies to the querying node. This has the effect of increasing the number of nodes that reply to the querying node. For the BDAYPARADOX and RANDOMINCWALK algorithms, we increase the number of trials that were performed. Each trial in BDAYPARADOX consists of random samplings of nodes until the first repetition of a sampled node. Each trial in the RANDOMINCWALK consists of a random increasing walk on the nodes in the network. Both of them have the effect of increasing the number of messages sent in the network.

Figure 5 shows the performance observed for the RANDOMIZEDREPORT algorithm. The X-axis plots the probability p with which a node replies to the querying node on a logarithmic scale, while the Y-axis plots the number of nodes estimated in the network. We see that the estimate improves dramatically, as p increases. However, the costs have to be borne by the querying node Q which receives these increased messages within a short interval of time.

Figures 6 and 7 show the performance observed for

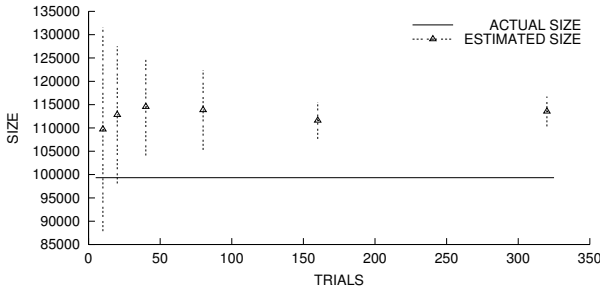


Fig. 7. Accuracy of RANDOMINCWALK

Measure	RR	BP	RIW
Flexibility	High	Medium	Low
Comm. cost	$\approx 400K$	$\approx 100K$	$\approx 1K$

Fig. 8. Comparing NODECOUNTING algorithms.

the BDAYPARADOX and RANDOMINCWALK algorithms respectively. The X-axis plots the number of trials performed while the Y-axis plots the number of nodes estimated in the network. We see that the variance decreases with an increase in the number of trials. The estimates are off from the actual size due to a bias in the estimation. The costs in both these experiments are distributed, with no single node bearing the brunt of the messages.

Figure 8 compares the different NODECOUNTING algorithms discussed in this section. The communication costs reported are for runs that return similar accuracy ($\approx 5\%$ variance) to make comparisons meaningful. The RANDOMIZEDREPORT (RR) is the most flexible as it can be used with any underlying topology. However, it is broadcast-based and incurs a high communication cost ($O(|E|)$ messages). The BIRTHDAYPARADOX (BP) can be used on topologies that allow a random node to be picked efficiently. For a RANDOM topology, its communication costs ($O(\hat{D}\sqrt{|V|})$ messages) are less than RANDOMIZEDREPORT. The RANDOMINCWALK (RIW) is the least flexible, feasible only on a RANDOM topology. However, it has a substantially less cost than the other two.

VII. RELATED WORK

To the best of our knowledge, we are the first to define the NODEAGGREGATION problem in the context of P2P networks. However, aggregation of information is currently an active field of research in sensor networks [6], [10], [18]. The emphasis has been on building energy efficient aggregation schemes but the validity guarantees provided are weaker than ours. The computability results discussed in Section IV will hold in such networks.

The goal of our work is similar to a part of the Astrolabe [17] project. Astrolabe is designed to enable a DNS-like distributed management service. As part of this service, Astrolabe allows nodes to aggregate information by dividing the network into (non-overlapping) zones arranged in a hierarchy. A leader elected in each zone is responsible for collecting information directly from all

the members in its zone. This information is then spread across other zonal agents using a gossip protocol. Such hierarchical aggregation is akin to SINGLETREE and suffers from drawbacks discussed in Section V-A.

The PROPAGATE2ALL and MULTIPLETREES algorithms presented in Section V use the well-known schemes of Flajolet and Martin [5]. Similar techniques were used by Palmer et al [14] for approximating the neighbor function of a disk-based graph. The idea of sampling by random walks, used by our BDAYPARADOX algorithm, has also been employed for the purposes of sampling uniformly at random web pages [4], [7]. In our case, however, random sampling is an easier task since we are dealing with undirected graphs. The RANDOMINCWALK algorithm builds on the notion of Mulmuley games [13]. Several variants of Mulmuley games exist; our analysis mirrors the Type-A game.

VIII. CONCLUSIONS

We motivated and presented the problem of computing aggregate queries on a P2P network. The problem is complicated by the high degrees of distribution and dynamicity in the network. The dynamicity leads to possible changes in the aggregated values during query time. We defined natural notions of validity for aggregate queries and investigated the computability of these notions. We presented generic mechanisms that allow us to trade accuracy of the results with time and space efficiency. Experiments demonstrate the validity and accuracy of our schemes.

REFERENCES

- [1] The dss clip2 company.
- [2] R. Albert, H. Jeong, and A. Barabasi. Diameter of the World Wide Web. volume 401, pages 130–131, 1999.
- [3] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. In *Proc. 28th Annual Symposium on Theory of Computing (STOC'96)*. ACM, 1996.
- [4] Z. Bar-Yossef, A. Berg, S. Chien, J. Fakcharoenphol, and D. Weitz. Approximating aggregate queries about web pages via random walks. In *Proc. 26th Intl. Conf. on Very Large Data Bases (VLDB)*, 2000.
- [5] P. Flajolet and G. N. Martin. Probabilistic counting. In *Proc. 24th Annual Symposium on Foundations of Computer Science (FOCS'83)*. IEEE, 1983.
- [6] I. Gupta, R. van Renesse, and K. Birman. Scalable fault-tolerant aggregation in large process groups. In *Proc. Conf. on Dependable Systems and Networks*, 2001.
- [7] M. R. Henzinger, A. Heydon, M. Mitzenmacher, and M. Najork. On near-uniform url sampling. In *Proc. 9th Intl. Conf. on World Wide Web (WWW9)*, 2000.
- [8] P. Keyani, B. Larson, and M. Senthil. Peer pressure: Distributed recovery from attacks in peer-to-peer systems. In *Intl. Workshop on Peer-to-peer computing (Networking2002)*, 2002.
- [9] J. Kleinberg. The Small-World Phenomenon: An Algorithmic Perspective. In *Proc. 32nd ACM Symposium on Theory of Computing (STOC'00)*, 2000.
- [10] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. TAG: a Tiny AGgregation Service for Ad-Hoc Sensor Networks. In *Proc. 5th Symposium on Operating Systems Design and Implementation (OSDI)*, 2002.

- [11] G. S. Manku, M. Bawa, and P. Raghavan. Symphony: Distributed hashing in a small world. In *Proc. of the Usenix Symposium on Internet Technologies and Systems (USITS)*, 2003.
- [12] S. Milgram. The small world problem. volume 1, 1967.
- [13] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, June, 1995.
- [14] C. R. Palmer, P. B. Gibbons, and C. Faloutsos. ANF: A Fast and Scalable Tool for Data Mining in Massive Graphs. In *Proc. 8th Conf. on Knowledge Discovery and Data Mining (SIGKDD)*, 2002.
- [15] G. Pandurangan, P. Raghavan, and E. Upfal. Building Low-Diameter P2P Networks. In *Proc. 42nd Symposium on Foundations of Computer Science (FOCS)*, 2001.
- [16] M. Ripeanu, I. Foster, and A. Iamnitchi. Mapping the gnutella network: Properties of large-scale peer-to-peer systems and implications for system design. volume 6(1). IEEE, 2002.
- [17] R. van Renesse. Scalable and secure resource location. In *HICSS*, 2000.
- [18] J. Zhao, R. Govindan, and D. Estrin. Computing aggregates for monitoring wireless sensor networks. In *Proc. 1st IEEE Intl. Workshop on Sensor Network Protocols and Applications (SNPA)*, 2003.

APPENDIX I: PROOFS OF THEOREMS

Theorem V.1: The SINGLETREE algorithm cannot guarantee practical validity even with 1 node failure.

Proof: Consider a node u that has participated in the broadcast phase and is now an interior node in the resulting spanning tree. Let one of its descendants in the tree be d that is also connected to q by a path P such that $u \notin P$. Note that the path P will not exist in the tree. If the node u unsubscribes from the system, then d no longer has a path to the root q that lies along the tree. Since information is propagated to the root along the tree in SINGLETREE, the information about d cannot reach q . However, there exist some path p in $G(V, E)$ that still connects d to q . Since d was not taken into account in the final answer at q , the algorithm does not ensure practical validity.

Theorem V.2: Assume that PROPAGATE2ALL with value \hat{D} is used by the querying node q for computing MIN or MAX aggregates. Then the answer of the query would be correct over all nodes u that are constantly connected with q over a path of length at most \hat{D} .

Proof: Consider a node u connected with q through a path P of length at most \hat{D} . The broadcast message takes time at most $\hat{D}\Delta$ to reach u , and the value of u takes time at most $\hat{D}\Delta$ to reach q . Every node $v \in P$ will transmit the broadcast message since we assume no failures on P . Similarly, every node on path P will transmit the value from u back to q since it gets activated after q does, and hence is active for a longer time period.

Theorem VI.1: The algorithm RANDOMIZEDREPORT guarantees probabilistic approximate practical validity, provided $p \geq \frac{4}{\epsilon^2 n} \ln \frac{2}{\delta}$. The variance of the estimate is $n(1-p)/p$.

Proof: Consider the random variables Y_i that take value 1 if node i replies to querying node, and 0 otherwise. The estimated size of the network computed by the querying node would be $Z = \sum_{i=1}^n Y_i/p$. Define

$Y = \sum_{i=1}^n Y_i$ so that $Z = Y/p$. It is easy to verify that $E[Z] = n$. Since the random variables Y_i are independent Poisson trials with $Pr[Y_i = 1] = p$ we can apply the Chernoff bound in order to bound the probability $Pr[(1-\epsilon)n \leq Z \leq (1+\epsilon)n]$. Following the details of Theorem 11.1 of [13], we can show that for $p \geq \frac{4}{\epsilon^2 n} \ln \frac{2}{\delta}$, the latter probability is at least $1 - \delta$.

Theorem VI.2: Let ϵ be a small positive numbers. Let n be the number of nodes in the network and X be an estimate produced by repeating the BDAYPARADOX algorithm $O(\frac{1}{\epsilon^2})$ times. Then, the relative error of the estimate, defined as $\frac{|X_n - n|}{n}$, is less than ϵ with high probability.

Proof: Assume that $\epsilon < 1$. We will show how we can approximate $\sqrt{2n}$ with relative error $\epsilon/3$, which yields an ϵ approximation for n . To see that assume that the estimate X approximates $\sqrt{2n}$ with relative error $\frac{\epsilon}{3}$, i.e. $|X - \sqrt{2n}| \leq \frac{\epsilon}{3}\sqrt{2n}$. Set $X_n = \frac{X^2}{2}$. The error of the approximation of X_n to n is

$$\begin{aligned} |X_n - n| &= \frac{1}{2}|X - \sqrt{2n}||X + \sqrt{2n}| \\ &\leq \frac{1}{2}\left(\frac{\epsilon}{3}\sqrt{2n}\right)\left((2 + \frac{\epsilon}{3})\sqrt{2n}\right) < \epsilon n \end{aligned}$$

Now let X_i be the estimate produced by the i th run of the algorithm. We have $E[X_i] = \sqrt{2n}$ and $Var[X_i] \leq c_1\sqrt{n}$, for some constant $c_1 > 0$. Let X be the average of $O(\frac{1}{\epsilon^2})$ independent X_i variables (produced by independent runs of the algorithm). The expectation of X is the same with the expectation of X_i , while $Var[X] \leq \frac{c_2\sqrt{n}}{\epsilon^2}$. By Chebyshev's inequality [13], we get

$$Pr[|X - \sqrt{2n}| < \frac{\epsilon}{3}\sqrt{2n}] \leq \frac{9Var[X]}{2\epsilon^2 n} = \frac{c_3}{\sqrt{n}},$$

which provides the high probability bound.

Theorem VI.3: Assume, without loss of generality, that node ids are drawn from the set $\{1, \dots, n\}$. Let random variable L denote the number of steps in a random increasing walk starting at node 1 and reaching node n . Define $A_n = E[L]$; then, $A_n = H_{n-1}$, where $H_0 = 0$ and H_n is the n th Harmonic number for positive integer n .

Proof: Conditional upon the first hop being to node i , the expected value of the path length is $1 + A_{n-i+1}$. This is because the nodes $1, 2, \dots, i-1$ can no longer occur on the path, and the new starting point is node i . Since i is uniformly distributed over $2, 3, \dots, n$, we obtain the following recurrence.

$$A_n = \sum_{i=2}^n \frac{1 + A_{n-i+1}}{n-1} = \sum_{j=1}^{n-1} \frac{1 + A_j}{n-1} = 1 + \sum_{j=1}^{n-1} \frac{A_j}{n-1}.$$

Upon rearrangement, using the fact that $A_1 = 0$, we obtain $\sum_{j=2}^{n-1} A_j = (n-1)A_n - (n-1)$. Note that Harmonic numbers satisfy the property that $\sum_{j=1}^{n-2} H_j = (n-1)H_{n-1} - (n-1)$ and hence are the solution to the recurrence. Also note that $H_n = \ln n + O(1)$.

APPENDIX II: PSEUDO-CODE FOR
NODEAGGREGATION SCHEMES

SINGLETREE(f)

▷ **On initialization at each node i**

$received := FALSE$

$parent := \phi$

$children := \phi$

$toBeAccounted := N(i)$

▷ **To begin aggregating at q**

$received := TRUE$

$treeAggregate := f(i)$

for each $v \in N(q)$

do SEND("QUERY", f, v)

▷ **On RECV("QUERY", f, i) at j**

if $received \neq TRUE$

then $received := TRUE$

$treeAggregate := f(i)$

$parent := i$

 SEND("CHILD", "YES", i)

for each $v \in N(j) - \{i\}$

do SEND("QUERY", f, v)

else SEND("CHILD", "NO", i)

▷ **On RECV("CHILD", R, i) at j**

if $R = "YES"$

then $children := children \cup \{i\}$

$toBeAccounted := toBeAccounted - \{i\}$

ReportAggregate()

▷ **On detection of fault of node i at j**

$children := children - \{i\}$

$toBeAccounted := toBeAccounted - \{i\}$

ReportAggregate()

▷ **On ReportAggregate() at node i**

if $toBeAccounted \neq \phi$ or $children \neq \phi$

then RETURN

if $i \neq q$

then SEND("VALUE", $treeAggregate, parent$)

else PRINT($treeAggregate$)

TERMINATE

▷ **On RECV("VALUE", A, i) at j**

$treeAggregate := f(treeAggregate, A)$

$children := children - \{i\}$

PROPAGATE2ALL(f, \hat{D})

▷ **On initialization at each node i**

$active := FALSE$

$timer := \phi$

▷ **To start aggregating at node q**

$active := TRUE$

$aggregate := f(q)$

$timer := T + 2\hat{D}\Delta$

for each $v \in N(q)$

do SEND("QUERY", $f, aggregate, v$)

▷ **On RECV("QUERY", f, A) at node i**

if $active \neq TRUE$

then $active := TRUE$

$timer := T + 2\hat{D}\Delta$

$aggregate := f(i)$

for each $v \in N(i)$

do SEND("QUERY", $f, aggregate, v$)

$newAggregate := f(aggregate, A)$

if $newAggregate \neq aggregate$

then for each $v \in N(i)$

do SEND("QUERY", $f, aggregate, v$)

▷ **On expiry of timer at node i**

if $i = q$

then PRINT($aggregate$)

TERMINATE

MULTIPLEPARENTS(f, k, \hat{D})

▷ **On initialization at each node i**

$finished := FALSE$

$toBeAccounted := N(i)$

$level := \infty$

$lMaps := \phi$

▷ **To begin aggregating at q**

$level := 0$

for each $v \in N(q)$

do SEND("QUERY", $f, k, level, v$)

▷ **On RECV("QUERY", f, k, l) from j at i**

if $level = \infty$

then $level := l + 1$

for each $v \in N(i)$

do SEND("QUERY", $f, k, level, v$)

$lMaps := lMaps \cup \{ \langle j, l \rangle \}$

$toBeAccounted := toBeAccounted - \{j\}$

formTrees(k)

▷ **On formTrees(k) at i**

if $toBeAccounted = \phi$

then $finished := TRUE$

$candidates := \{ j : \langle j, l \rangle \in lMaps, l < level \}$

$parents := \text{PICK UPTO } k \text{ RANDOM NODES}$

 FROM $candidates$

for each $v \in parents$

do SEND(“CHILD”, v)
ReportAggregate()

▷ **On RECV(“CHILD”) from j at i**
 $children := children \cup j$

▷ **On RECV(“VALUE”, A) from j at i**
 $children := children - \{j\}$
 $aggregate := f(aggregate, A)$
ReportAggregate()

▷ **On ReportAggregate() at i**
if $children = \phi$
 then for each $v \in parents$
 do SEND(“VALUE”, $aggregate, v$)
 if $i = q$
 then PRINT($aggregate$)
 TERMINATE

PRINT($numNodes$)
TERMINATE

▷ *PerformTrial*(r)
 $samples := \phi$
 $nextSample := \phi$
repeat $hops := 0$
 $v := q$
 while $hops < r$
 do $nh := \text{SEND(“RANDOMNBR”,$
 $q, v)$

$hops := hops + 1$
 $v := nh$
 $nextSample := v$
 until $nextSample \in samples$
 RETURN($|samples|^2$)

▷ **On RECV(“RANDOMNBR”, q) at i**
 $randomNbr := \text{PICK A RANDOM NODE FROM } N(i)$
SEND(“VALUE”, $randomNbr, q$)

APPENDIX III: PSEUDO-CODE FOR NODECOUNTING SCHEMES

RANDOMIZEDREPORT(p, \hat{D})

▷ **To begin counting at q at time T**

$count := 1$
 $timer := T + \hat{D} \times \Delta$
for each $v \in N(q)$
 do SEND(“COUNT”, p, q, v)

▷ **On RECV(“COUNT”, p, q) at j**
 $heads := TRUE$ WITH PROB. p and $FALSE$ O.W
if $heads = TRUE$
 then SEND(“VALUE”, $1, q$)
for each $v \in N(j) - \{i\}$
 do SEND(“COUNT”, p, q, v)
TERMINATE

▷ **On RECV(“VALUE”, 1) at q**
 $count := count + 1$

▷ **On expiry of timer at q**
 $numNodes := count/p$
PRINT($numNodes$)
TERMINATE

BDAYPARADOX(r, ϵ)

▷ **To begin counting at q**
 $trials := 0$
 $answer := 0$
repeat $trials := trials + 1$
 $answer += \text{PerformTrial}(r)$
until $trials < \frac{1}{\epsilon^2}$
 $numNodes := answer/trials$

RANDOMINCWALK()

▷ **To begin counting at q**

$l := 0$
 $nh := \text{PICK A RANDOM NODE FROM } N(q)$
SEND(“INCWALK”, q, l, nh)

▷ **On RECV(“INCWALK”, q, l) from j at i**

$l := l + 1$
 $adjustedId := i + q$
 $greaterNbrs := \phi$
for each $v \in N(i) - \{q\}$
 do $adjustedNbr := v + q$
 if $adjustedNbr > adjustedId$
 then $greaterNbrs := greaterNbrs \cup \{v\}$

if $greaterNbrs \neq \phi$
 then $nh := \text{PICK A RANDOM NODE FROM}$
 $greaterNbrs$

 SEND(“INCWALK”, q, l, nh)

else SEND(“VALUE”, l, q)

 TERMINATE

▷ **On RECV(“VALUE”, l) at q**

$numNodes := e^l$
PRINT($numNodes$)
TERMINATE