Available online at www.sciencedirect.com

ScienceDirect

The Journal of Systems and Software 80 (2007) 1639–1658

The Journal of Systems and Software

www.elsevier.com/locate/jss

Active and passive techniques for group size estimation in large-scale and dynamic distributed systems ☆

Dionysios Kostoulas [a], Dimitrios Psaltoulis [a], Indranil Gupta [a,*], Kenneth P. Birman [b], Alan J. Demers [b]

[a] *Department of Computer Science, University of Illinois, 201 N. Goodwin Avenue, Urbana-Champaign, IL 61801, USA*

[b] *Department of Computer Science, Cornell University, Upson Hall, Ithaca, NY, 14853, USA*

Received 3 March 2006; received in revised form 11 January 2007; accepted 12 January 2007
Available online 27 January 2007

Abstract

This paper presents two solutions to a distributed statistic collection problem, called Group Size Estimation. These algorithms are intended for large-scale and dynamic distributed systems such as Grids, peer-to-peer overlays, etc. Each algorithm estimates (both in a one-shot and continuous manner) the number of non-faulty processes present in the global group. The first active scheme samples receipt times of gossip messages, while the second passive scheme calculates the density of process identifiers when hashed to a real interval. Our analysis, trace-driven simulation and deployment on a 33-node Linux cluster study and compare the latencies, scalability, and accuracy of these schemes.
© 2007 Elsevier Inc. All rights reserved.

Keywords: Large-scale distributed systems; Group size estimation; Probabilistic protocols; Algorithm design and analysis; Epidemics

1. Introduction

Distributed systems such as peer-to-peer overlays, sensor networks, Grid application overlays, etc., tend to be large-scale since they contain several thousands of processes. More importantly, however, they are also *dynamic*. Dynamism means that there is continuous arrival and departure activity through processes joining, crashing and voluntarily departing. At the same time, distributed applications often require an estimate of the number of non-faulty processes currently present in the group. We call this as the problem of *Group Size Estimation*. The problem is challenging not only due to the dynamism and scale involved, but also because the above mentioned peer-to-peer overlays typically require each process to maintain only *partial* information concerning the group membership; thus, no process ever has the entire group membership list with it and cannot use this to estimate the group size.

Accurate protocols for group size estimation are absolutely essential for some peer to peer systems. In other situations, these protocols can be used for purposes such as optimization and monitoring of large-scale distributed systems. Below, we give examples of systems in both these categories – this list is by no means comprehensive, especially since the peer to peer community continues to innovate more new systems. In peer-to-peer systems, several distributed hash tables (DHTs) such as Symphony (Manku et al., 2003), Y_0 (Godfrey and Stoica, 2005), Viceroy (Malkhi et al., 2002), and Kelips (Gupta et al., 2003), all critically depend on an estimation of group size. Schemes for node ID management such as by Manku (2004) also require an estimate of group size. Knowledge of group size can be

☆ This work was supported in part by DARPA AFOSR and in part by NSF (and by NSF CAREER Grant CNS-0448246).

* Corresponding author. Tel.: +1 217 265 5517; fax: +1 217 265 6494.
E-mail addresses: kostoula@cs.uiuc.edu (D. Kostoulas), psaltoul@cs.uiuc.edu (D. Psaltoulis), indy@cs.uiuc.edu (I. Gupta), ken@cs.cornell.edu (K.P. Birman), ademers@cs.cornell.edu (A.J. Demers).

used to better estimate the expected hop counts and latencies for queries in peer to peer DHTs where a routing message typically takes a virtual hop-count that is some function of the group size N – this includes classical $\log(N)$-hop routing DHTs such as Pastry (Rowstron and Druschel, 2001) and Chord (Stoica et al., 2001). Knowledge of group size can also be used to monitor and audit the performance of distributed applications based, e.g., applications running in "slices" on the PlanetLab testbed (PlanetLab, 2006). Finally, in Grid computing applications, continuous estimates of the group size can be used for dynamic load balancing and dynamic partitioning of distributed computations among participating clients.

The Group Size Estimation problem is representative of a large class of problems for collecting *statistics* about a large-scale distributed system, in a decentralized manner. Other problems in this class include aggregation, e.g., Bawa et al. (2003). However, the specificity of the Group Size Estimation problem lends itself to solutions with the potential for much greater accuracy and scalability than that obtained by the straightforward application of aggregation algorithms.

The Group Size Estimation problem has two flavors – *one-shot* and *continuous*. We formally define the one-shot problem next – the continuous version is similar.

Group Size Estimation problem (one-shot version): Give a protocol that when initiated by one process (used interchangeably with "node"), estimates N, the number of non-faulty processes (nodes) in the overlay graph component containing the initiating process.

Impossibility of group size estimation in a dynamic group: This problem is impossible to solve exactly (i.e., accurately) in a dynamic group. Notice that a group size estimation protocol will take non-zero time to run, and for any process p that is not the initiator, there will be a non-zero delay between p's last message in the estimation protocol, and the initiator finalizing the estimate. In a run where process p fails during this interval, the estimate will be incorrect.

This motivates the design of algorithms for *approximate* estimation. One simple algorithm could be the following. The initiator process sends a multicast to the overlay through a dynamic spanning tree, and waits to receive back replies. The disadvantage of this scheme is that it is non-fault-tolerant – if a process fails after forwarding the multicast and before replying to its parent in the spanning tree, the estimate will exclude all descendants of that process in the tree. There is thus a need for more robust estimation schemes.

Key contributions of the paper: This paper proposes practical, decentralized, efficient, and fault-tolerant estimation algorithms with a probabilistic output. Specifically, we study two algorithms, both based on variants of sampling:

I. *Active approach*: This scheme is called the *Hops Sampling Algorithm*. It initiates a gossip into the group, and samples, based on hop distance, the times at which the gossip is received by different processes.

II. *Passive approach*: This scheme is called the *Interval Density Approach*. It samples the density of processes that lie in a given real interval, when the process identifiers are hashed into this interval.

Each of these algorithms can be run either in a one-shot manner or on a continuous basis (the latter variant is preferable for long-running applications such as peer-to-peer overlays). When used in a one-shot manner, the algorithms have running times that grow logarithmically with group size, and impose a small sublinear overhead on each process in order to achieve an estimate that is accurate with high probability (w.h.p.). In the continuous version, the time taken for a given process arrival/failure/departure to affect the group size estimate, grows logarithmically with the group size.

We present experimental evaluation of the two algorithms. The first set of experimental results is from a simulation, and includes micro-benchmarks and trace-based experiments. The second set of experiments uses *PeerCounter*, our open-source implementation of the size estimation algorithms. Using PeerCounter allows us to run deployment experiments on a cluster of up to 33 Linux servers, with each machine running multiple processes. These experiments indicate how our estimation algorithms will perform in a single Grid-style cluster. We assume that each Grid cluster runs its own version of the estimation algorithm, and we study this experimentally. For computations that span multiple clusters, the results will then need to be aggregated across clusters via a hierarchy such as Astrolabe (van Renesse et al., 2003) – a study of the interaction between PeerCounter and Astrolabe is beyond our focus here. PeerCounter is open-source and can be easily incorporated into a variety of peer-to-peer substrates, e.g., Symphony, Y_0, Viceroy, Kelips, Pastry, Chord (Manku et al., 2003; Godfrey and Stoica, 2005; Malkhi et al., 2002; Gupta et al., 2003; Rowstron and Druschel, 2001; Stoica et al., 2001), etc., thus providing these systems with the capability to estimate the number of non-faulty processes present in the group.

System model: We assume a group of processes (alternatively, "nodes") with unique identifiers, communicating through an asynchronous network, and connected in an overlay. Messages can be sent from a process u to any other process v whose identifier is known by u at its local *membership list* – these membership entries are nothing but the links in the peer-to-peer overlay. Node v is thus node u's *neighbor* in this overlay. Membership entries are assumed to be maintained by a complementary membership protocol already running within the overlay. The main requirement from the membership component is that the overlay graph formed by it be connected, i.e., have one connected component.

For the Hops Sampling scheme, in addition to connectivity, we also assume that the neighbors of any node in the membership graph are chosen uniformly at random. While this may appear to be restrictive, this assumption can be relaxed by leveraging the "Peer Sampling service" by Jelasity et al. (2004), or by using techniques such as random walks.

For the Interval Density scheme, in addition to connectivity, we also assume that the membership protocol falls in the "all-to-all heartbeating" category – this means that all membership changes (joins, leaves, failures) will eventually be propagated to all alive (non-faulty) nodes in the system, each of which may or may not choose to remember a received update. Notice that the membership protocol needs to be only weakly consistent, i.e., detect failures eventually. It is not required to provide strong guarantees such as virtual synchrony. Notice that the Interval Density scheme *does not* require the membership graph to be uniform or complete, but only that it is connected.

While the all-to-all heartbeating assumption for the Interval Density scheme may appear to be restrictive, in fact several heartbeat-style membership protocols such as by van Renesse et al. (1998) satisfy this, and many non-heartbeat-style membership protocols such as SCAMP, SWIM and T-Man do too (see Section 2). In addition, the membership maintenance in distributed hash tables (e.g. Pastry, Chord, etc.), unstructured overlays (e.g. Gnutella), as well as overlays in Grid applications, can all be modified to satisfy the all-to-all heartbeating requirement. We detail other requirements from the membership protocol where needed.

The number of non-faulty processes is N, and is an unknown quantity. Our protocols operate in *rounds* at each process, with the round duration fixed across the group. Processes are not required to have synchronized clocks. However, since the duration of a round is O(seconds), processes can be assumed to have negligible clock drifts – this can be supported by already deployed protocols such as NTP (Network Time Protocol). Processes can undergo crash-stop failures – crash-recovery failures can be supported by having a process rejoin the group with an identifier that is unique and unused previously.

The rest of this paper is organized as follows. Section 2 discusses related work. Sections 3 and 4 describe the Hops Sampling algorithm and the Interval Density approach, respectively. Sections 5 and 6 give experimental results from our implementations. We conclude in Section 7.

2. A short survey of related work

In this section, we first expand on the recent work in estimation and aggregation algorithms for peer to peer systems. Then we discuss comparative studies involving some of these algorithms (including ours). Finally, we discuss group membership protocols, as they are used by our estimation schemes.

Estimation algorithms: Sampling-based, size estimation algorithms have already been used for membership size estimation in multicast sessions (Alouf et al., 2002; Alouf et al., 2003; Friedman and Towsley, 1999). Friedman and Towsley (1999) examined probabilistic polling, a centralized technique in which the sender multicasts a polling request and examines the replies from a probabilistically selected subset of the receivers, in order to estimate the number of other receivers. Alouf et al. (2002) proposed a dynamic polling scheme that occasionally, i.e. every S seconds, repeats the request for acknowledgment replies in order to increase the accuracy of the estimate. Similar polling schemes, which either extend the applicability of those by Alouf et al. (2002) or increase their efficiency, were presented by the same authors (Alouf et al., 2003). Similar to our schemes, all polling-based algorithms mentioned here use variants of sampling in order to estimate the size of large and dynamically changing groups. However, probabilistic polling approaches provide centralized solutions for an Internet-related size estimation problem. Our algorithms, on the other hand, are decentralized and are intended to be applied on large-scale distributed systems that require each process to maintain only partial group membership information.

Several efforts have been made to provide decentralized solutions to the group size estimation problem. Most of these approaches either do not apply to highly dynamic groups, or have too high latencies, or are not fully decentralized, or their applicability has been proposed only for non-peer-to-peer systems. We discuss a variety of previous work on the problem below.

Bawa et al. (2003) presented several mechanisms for aggregation and group size estimation. One algorithm has an estimation message sent along a random walk within the group. When the estimation message hits a process for the second time, the protocol stops and the number of hops traversed so far can be used to estimate the group size – according to the birthday paradox, it takes $\Theta(\sqrt{N})$ hops for a random walk message to encounter a process a second time. However, the latency of this algorithm is $\Theta(\sqrt{N})$, and this is high for groups that have processes joining and leaving at high rates. Our schemes have latencies that increase only logarithmically with N.

Massoulié et al. (2006) proposed the "Sample and Collide" scheme based on the inverted birthday paradox. The aggregation schemes by Jelasity and Montresor (2004), and by Jelasity and Preuss (2002), and by Kempe et al. (2003) are all based on gossip-based aggregation. Nevertheless, the accuracy of these above methods critically requires a uniform scheme for sampling a random node in the group. Our Hops Sampling approach relies on such a random sampling assumption, but our Interval Density approach does not. Furthermore, all the algorithms just mentioned have been evaluated only in systems that are either static or where N varies after long time intervals. In comparison, we show that our algorithms provide very accurate estimation even under high rates of continuous churn. Some of these algorithms were compared against the Hops Sampling approach – see our discussion on Le Merrer et al. (2006) later in this section.

The approach proposed by Awerbuch and Scheideler (2004) bears some resemblance to our Interval Density algorithm, but it is difficult to do a quantitative comparison among these two approaches. This is because of two reasons: (1) the approach by Awerbuch and Scheideler (2004) uses a hash function to *assign* special IDs to processes in the real interval [0, 1], while our approach does not assume any such ID assignment scheme; (2) the approach by Awerbuch and Scheideler (2004) works by using a single initiator process (and may cause implosion at this initiator), while our scheme is a passive approach. This scheme decomposes the interval [0, 1] into a hierarchy of subintervals, each holding identifiers from processes that lie in the same region. The estimation then occurs in a bottom-up fashion, starting from each peer and working upwards by aggregating across larger and larger intervals. In comparison, our algorithm is much simpler since it relies only on sampling the density across the interval – this will be explained later in the paper.

Stutzbach and Rejaie (2005) used a topology crawler for estimating the size of peer-to-peer systems. The topology crawler aggregates the partial group size estimates at different super-peers and thus avoids communication overhead at each of the peers in the system. While this approach would work well in systems that use supernodes, e.g., Fasttrack (on top of which Kazaa is built), or by adding algorithms to identify supernodes in a system (for estimation), this would still distribute the estimation load unevenly over the nodes in a system. In comparison, either of our algorithms can be used generically in any distributed system, and is completely decentralized.

A ring-based algorithm for group size estimation was discussed by Malkhi and Horowitz (2003); however, the estimated group size may have a large error – this could be between $N/2$ and N^2, where N is the actual group size. The latency is also high, and increases linearly with the system size N. The required logical ring may not be present in all overlays, e.g., non-ring based overlays, random overlays, etc. Our first scheme in this paper (Hops Sampling) assumes only random sampling capabilities at each node and no invariant overlay rules (such as a ring); our second scheme in the paper (Interval Density) makes no assumptions on the overlay structure.

In sensor networks, the problems of aggregation and counting have been targeted very well by the innovative and promising Synopsis Diffusion techniques developed by Nath et al. (2004), which use algorithms by Flajolet and Martin for aggregation without doubling-counting. However, so far, Synopsis Diffusion has been realized *only* for sensor networks, and not yet for p2p settings. Since the sensor network environment is remarkably different from the peer to peer environment, the Synopsis Diffusion-based aggregation approaches have different goals from ours. For instance, the effect of churn is not as important in sensor networks as in p2p systems, hence Nath et al. did not analyze this at all – in fact, most of the experiments in their paper deal with static groups (while we are targeting dynamic groups too in this paper). However, until Synopsis Diffusion-like techniques are adapted to Internet-based peer-to-peer systems, the question of whether or not that approach will outperform our presented algorithms, remains open.

Epidemic algorithms were discussed by Demers et al. (1987); Eugster et al. (2003); Karp et al. (2000). The idea of epidemic algorithms originates from the mathematical study of epidemics as studied by Bailey (1975). Birman et al. (1999) used gossip to design a probabilistically reliable multicast protocol.

Our schemes were originally published as (Kostoulas et al., 2005). This journal paper contains additional experimental data, and additional analysis of this data.

Comparisons of estimation algorithms: Although we compare our proposed active and passive schemes against each other in this paper, we do not explicitly compare these to the other schemes mentioned above. There are two reasons for this choice: (1) Our goal is to *comprehensively* study the behavior of each of our proposed schemes rather than devote space to comparison with other approaches; (2) Parallel work by Le Merrer et al. (2006) has recently compared three active estimation schemes, including a variant of the Hops Sampling approach and two sampling-based approaches (Massoulié et al. (2006) and Jelasity and Montresor (2004)) to each other. No passive sampling approaches were considered in the study by Le Merrer et al. We do not wish to replicate that work here, but instead wish to evaluate the benefits of active versus passive estimation. Le Merrer et al. (2006) did not evaluate the effect of continual churn (as we do in this paper in Section 5), but only studied churn models where system sizes change at large discrete intervals. As a result, they focus on only one notion of accuracy (i.e., error from the actual system size), while we both: (a) quantify this better as the RMSE metric (Section 5), and (b) introduce a second notion of accuracy (StdDevErr, Section 5), which bears significance for continual churn models.

For the specific discrete churn models studied by Le Merrer et al., they found that compared to the other schemes, Hops Sampling used an overhead that was neither too high nor too low (Table 1 in their paper), had an accuracy that was slightly under the actual system size, and was likely to have the lowest latency among all approaches. This would make Hops Sampling suitable for settings where bandwidth is not too much a concern, but estimation latency is. However, Le Merrer et al. did not: (1) study the bandwidth-accuracy tradeoff created by varying the number of Hops Sampling runs considered in calculating the estimate, or (2) vary the parameter values in order to optimize the behavior of Hops Sampling. We believe these might improve the performance of Hops Sampling. However, these rebuttal issues are *not* the focus of our paper here – our main goal in the experimental part of the current paper is to propose active and passive estimation schemes, and compare them against each other.

Group membership algorithms: From the System Model in Section 1, we need a group membership protocol that maintains connectivity, and for the Hops Sampling approach uniform neighbor choice, while for the Interval

Table 1
PeerCounter's API

alg = new HopCountingAlgorithm(members) or **alg = new IntervalCountingAlgorithm(members)**	Instantiates one of the estimation algorithms *members* holds the membership list of the calling process
alg.startCounting()	Causes the gossip stream to start spreading over the group
alg.getSize()	Causes a size estimate to be returned

Density scheme, satisfies an all-to-all heartbeating requirement. For this purpose we choose the heartbeat-style gossip-style membership protocol by van Renesse et al. (1998). This membership protocol was chosen because it is simple, fault-tolerant, scalable, and satisfies all the above conditions. In addition, it was the first gossip-style membership protocol ever proposed, and has been used in several systems including Astrolabe by van Renesse et al. (2003) and in Kelips by Gupta et al. (2003). Notice that our estimation protocols are not restricted to using only the van Renesse et al. protocol – in its place, other membership protocols such as SCAMP by Ganesh et al. (2001), SWIM by Das et al. (2002), and T-Man by Jelasity and Babaoglu (2005) can be used – for concreteness, Section 5 analytically calculates bandwidth if we were using SCAMP instead underneath our Hops Sampling approach.

3. Active approach – the Hops Sampling algorithm

In this section, we present the Hops Sampling algorithm for size estimation, and analyze its performance for peer-to-peer systems.

The intuition behind the Hops Sampling algorithm is to first spread a *gossip* or an *epidemic* message through the group, and then to measure the average (or distribution of) receipt times of this gossip message at different participants, comparing it with the theoretically expected receipt time in order to obtain a group size estimate. In order to relate these empirical measurements with the theoretical value, the Hops Sampling Algorithm relies on two assumptions:

1. Each process operates in *protocol periods* (also called *rounds*). The duration of a *round* is fixed and a round starts at the same time at all processes, i.e., it is synchronized. We assume that each gossip round is several seconds in length. Thus, our assumption of synchronous rounds can be supported by using any time synchronization protocol such as NTP (Network Time Protocol) in order to guarantee a sufficiently high level of accuracy.
2. The second assumption relates to the *neighbors* that processes have in the distributed group – recall that this list of neighbors is called the *membership list*. We assume that there is an underlying membership protocol already running in the distributed system that enables each process to maintain a neighbor list selected *uniformly at random* from across the group. The neighbor list (also sometimes called a *view*) is only *partial*, i.e., each process knows of only *some* other participants in the group. In general, this number is O(log(N)), where N is the current size of the group. Once again, this assumption can also be supported by using a membership protocol such as SCAMP, SWIM, T-Man, or gossip-style heartbeating, etc., that maintain partial views that are uniform at random, and autonomically updates this in spite of continuous node join, leave, and failure (although with such churn, the membership protocol may be weakly consistent).

Fig. 1 shows pseudocode for the canonical Hops Sampling algorithm, and we describe the details in words below. The estimation is started by one process called the *initiator*, but the rest of the protocol operates in a decentralized manner throughout the group. The initiator sends an *initiating message* at the start of the protocol.

At each process, the protocol uses several parameters – (1) the number of gossip targets `gossipTo`, (2) the number of rounds `gossipFor` devoted to each given gossip message, and (3) `gossipUntil`, the maximum number of received copies of a gossip beyond which outgoing gossips are stopped at the process.

In detail – once a process has received the initiating message, at the beginning of each subsequent round, it selects `gossipTo` other processes randomly from its membership list, and marks them as targets for that round. The targets are sent *gossip* messages containing the initiating message. Gossip messages need to be acknowledged, and a gossiping node retries each gossip message, until `gossipTo` targets have acknowledged for that round. A process stops gossiping once either `gossipFor` rounds have expired since receipt of the initiating message, or `gossipUntil` gossip messages have been received by the process. All `gossip`* parameters mentioned above are a priori fixed integer constants. Finally, in order to reduce the message overhead, besides the local membership list, each process p also maintains a list, called `fromList`, of some other processes that *it* knows to have already been infected – these are simply the processes that have sent gossip messages to process p. The selection of gossip targets by a process is changed so that the random choice from the membership list excludes the elements that already appear in `fromList`.

In order to enable estimation, each of the above gossip messages also carries an integer field `hopNumber`, which indicates the number of nodes the gossip message has traversed since the initiator. Each process maintains a local variable called `myHopCount`, which is initialized by the `hopNumber` of the very *first* gossip message received at this process. Thereafter, any outgoing gossip message from this process will have its `hopNumber` set to

```
Hops Sampling Protocol (gossipTo, gossipFor, gossipUntil, gossipResult, gossipSample):

/* Single initiator, Single instance. Multiple runs will require extra space and processing for a
unique run number. */
Gossip message format: (hopNumber, initiator, from);
Group Membership List: G;
fromList = NULL; /* list of processes from which I have received gossip */

  At the Initiator process p:
  Initially:
      gossip_on = TRUE;     // am I actively gossiping?
      past_gossip_rnds = 0; // number of past local gossip rounds
      myHopCount = 0;
      num_recvdgossips = 0; // number of gossips received so far
      initiator = p;

  Once every protocol period:
      if (gossip_on == TRUE)
        send message (1, p, p) to gossipTo processes selected uniformly at random from set G -
  fromList;
        past_gossip_rnds++;
        if (past_gossip_rnds > gossipFor)
           gossip_on = FALSE;

  On receipt of a gossip message M: /* received messages processed serially */
      fromList = fromList + {M.from};
      num_recvdgossips++;
      if (num_recvdgossips > gossipUntil)
           gossip_on = FALSE;

  gossipResult protocol periods after starting:
      sample a random subset of gossipSample processes for their myHopCount values;
      for each non-zero value received do: HopReportArray[myHopCount]++;
      output the mean of received non-zero values as: (weighted sum of different values) /
gossipSample;

  At a non-initiator process p:
  Initially:
      gossip_on = FALSE; // am I actively gossiping?
      past_gossip_rnds = 0; // number of past local gossip rounds
      myHopCount = 0; // at what hop did I receive the gossip?
      num_recvdgossips = 0; // number of gossips received so far
      initiator = NULL;

  Once every protocol period:
      if (gossip_on == TRUE){
        send message (myHopCount+1, initiator, p) to gossipTo processes selected uniformly at
  random from set G - fromList;
        past_gossip_rnds++;
        if (past_gossip_rnds > gossipFor) gossip_on = FALSE;
      }

  On receipt of a gossip message M: /* received messages processed serially */
      fromList = fromList + {M.from};
      if (num_recvdgossips == 0){
        myHopCount = M.hopNumber;
        initiator = M.initiator;
        gossip_on = TRUE;
      }
      num_recvdgossips++;
      if (num_recvdgossips > gossipUntil) gossip_on = FALSE;
```

Fig. 1. The Hops Sampling algorithm for estimation. The algorithm is for a one-shot estimation by a single initiator. Continuous estimation requires each gossip message to carry an extra unique identifier for the run number.

(myHopCount + 1). Note that after its initialization, myHopCount is not updated further.

The initiating process uses another parameter to decide how long to wait before declaring the result of the estimation protocol. The initiator waits for gossipResult rounds to elapse before sampling gossipSample other processes selected uniformly at random from its membership list. Each sampled process replies with its myHopCount value. The average of these values is returned by the algorithm as an estimate of $\log(N)$, where the logarithm's base depends on the parameter settings.

On setting the values of gossip* *parameters*: As mentioned before, all gossip* parameters mentioned above are a priori fixed integer constants. We wish to point out that these integer values are *not* required to be magic numbers. Instead, in our implementation, we set them to small values and we then empirically derive the logarithm base (see Section 5.1).

Optimization: Instead of the initiator sampling the group size, an alternative sampling technique we use in our implementation has the gossip recipients themselves send their hop count values back to the initiator. However, in order to reduce message implosion, the initiating message contains a fixed value `minHopsReporting` specified by the initiator. When a process stops gossiping (if it ever gossips), it sends its `myHopCount` automatically to the initiator (i) with probability 1 if `myHopCount` < `minHopsReporting`, and (ii) otherwise with probability (1/`gossipTo`$^{(\text{myHopCount-minHopsReporting})}$). Thus, only a small fraction of all hop count values are reported back.

Relaxing the synchronous assumption: Although our protocol description and analysis assume clocks are synchronized to protocol periods, our implementation (Section 6) eliminates this requirement by setting at each process the number of gossip rounds `gossipFor` to the value 1. This means each process gossips only once (to `gossipTo` targets), and this happens as soon as it receives the initiating message. A second modification in the implementation is that the initiator waits for a large, fixed time interval (usually several minutes) before sampling the group for hop counts – this eliminates the need to set `gossipResult`.

Relaxing the uniform target choice assumption: In the rest of the paper, for simplicity, we will continue to assume that each node is able to pick gossip targets uniformly at random (assumption 2 above). However, this assumption can indeed be relaxed by using the "Peer Sampling service" by Jelasity et al. (2004), or by using techniques such as random walks or by using partial random membership protocols such as SCAMP (Ganesh et al., 2001). The use of peer sampling these orthogonal services will retain the analysis in this paper, and only add as much additional overhead as is required by these services. A study of the interaction of these services with Hops Sampling is beyond the scope of this paper.

3.1. Analysis of the hops sampling algorithm

For tractability of analysis, we assume that processes in the group have $O(\log(N))$ membership list sizes and these are constructed by uniform sampling.

Theorem 1. *Consider a group where the local membership list maintained at each process is* $O(\log(N))$, *with each entry selected uniformly at random from across the group. Then, the expected time* (*after gossip initiation*) *at which a process receives the gossip varies as* $\Theta(\log(N))$.

Proof. At the core of the Hops Sampling algorithm is a push gossip protocol, the properties of which are well studied. Bailey (1975) and Eugster et al. (2003) showed the following results for push gossips in a group where processes have partial membership lists chosen uniformly at random from across the group: (i) If the quantity (`gossipTo` × `gossipFor`) is $\Theta(\log(N))$, the gossip spreads to all processes with high probability; (ii) the expected number of rounds for the completion of the dissemination of the gossip is $\Theta(\log(N))$; (iii) gossip spread in a group with complete membership lists is the same as in a group with partial membership lists of size $O(\log(N))$.

For the purpose of our protocol, we are interested in the *average* time that it takes for the gossip to reach a given process in the group. Clearly (ii) above says it can be no larger than $\Theta(\log(N))$. For the lower bound, notice that, during each round, the total number of gossip recipients cannot grow by a factor larger than `gossipTo`. This is a branching process with degree `gossipTo` and the height of the generated tree is $O(\log(N))$. This is the lower bound. Thus, the average measured hop count in our algorithm is $\Theta(\log(N))$. □

As pointed out before, Section 5 will empirically detail how to determine the value of the logarithm base in the average gossip latency – for now, it suffices to say that with the values of `gossipTo` and `gossipFor` that we choose, the logarithm base turns out to be close to 2.0.

Foreground message overhead of the Hops Sampling approach: In the worst case, with the above setting of parameters, each process in an N-process system gossips a Hops Sampling gossip message for approximately $\log_2 N$ rounds. This is a very low overhead – for instance in a 1 million-process system, at each process, $\log_2 N = 20$ rounds are devoted to a single gossip message. For each round, a constant number of gossip messages are sent out. This is a low bandwidth per node, and arguably this overhead is independent of group size for all practical purposes.

Continuous version of the Hops Sampling algorithm: A long-running peer-to-peer application may need to continuously monitor the variation of group size. The continuous variant of the Hops Sampling algorithm can provide this service. It works as follows. The initiator periodically initiates a new one-shot protocol run, each with a unique run identifier. A parameter `gossipsAccounted` gives the number of recent one-shot estimates to be considered for the continuous estimate. Of these estimates, `gossipsDropped` of the highest and `gossipsDropped` of the lowest estimates are dropped before taking the mean of the remaining estimates. The network traffic in this continuous version is bounded since each run terminates w.h.p. in a logarithmic number of rounds.

Convergence-foreground overhead tradeoff: For the Hops Sampling approach, the convergence time depends on the foreground bandwidth, i.e., bandwidth spent per-estimation. The foreground message overhead at a given node arises from the gossip messages that need to be propagated by the node. The smaller the protocol period at a node, the higher is the overhead, and vice-versa. Thus, the message overhead is inversely related to the estimation completion time for the one-shot version. In other words, increasing (respectively decreasing) the protocol period is akin to "stretching" (respectively "compressing") the time-line.

4. Passive approach – the interval density algorithm

In this section, we present our second algorithm – the Interval Density scheme – that does not require as many assumptions as the Hops Sampling algorithm. It neither requires processes to run synchronized clocks, nor does it require a uniform membership graph to be maintained. In fact, as long as the membership graph keeps all processes connected (i.e., reachable from each other), and all memory updates are eventually spread to all alive nodes, then the algorithm will continue to function and our analysis applies.

While the Hops Sampling approach was an *active* probing approach to group size estimation, the Interval Density approach is *passive.* This approach works by measuring the density of the process identifier space, i.e., the number of processes that have (unique) identifiers lying within an interval of this space. As a first cut, directly sampling the space of process identifiers (e.g., IP addresses + port number) may prove to be inaccurate. For example, if the group were located on a small collection of subnets, the process identifiers would be correlated, resulting in a great likelihood of error in the group size estimate.

A second approach is to randomize the process identifiers by using a good hash function to map each process identifier to a point in the real interval $[0,1]$. Cryptographic hash functions such as SHA-1 (FIPS, 1995) or MD-5 can be used: the input is arbitrary length binary strings, and the output is a hash of length 160 bits for SHA-1 (or 128 bits respectively for MD-5). The hashes can be normalized by dividing with 2^{160}–1 for (or 2^{128}–1 respectively). This is convenient to do in P2P *structured* routing substrates such as Pastry (Rowstron and Druschel, 2001) and Chord (Stoica et al., 2001), where virtual "nodeIDs" are already assigned to processes by hashing their identifiers using SHA-1 or MD-5. Even in unstructured P2P routing substrates such as Gnutella and Fasttrack, it is easy for each node to locally apply the SHA-1 or MD-5 function to its IP address and port number to derive its nodeID (even though this value is not used in the overlay itself, it will be used by our estimator).

The Interval Density approach requires the "initiator" process to passively collect information about the process identifiers that lie in an interval I that is a subset of the interval $[0,1]$. We denote by I itself the size of the interval I. Suppose X is the actual number of processes that the initiator finds falling in the interval. Then the estimate for the group size is simply returned as X/I. The passive collection of such process identifier information can be achieved by snooping on the complementary membership protocol running in the overlay – details will be provided soon in Section 4.4.

Notice that this lends itself easily to both a one-shot run and a continuous run – the latter can be provided by simply maintaining information about the processes (that lie in the interval I) over a long period of time. Multiple initiators can be supported by selecting the interval specified by the initiator with the lowest identifier, and participating in only that initiator's protocol run.

Next, we analyze the accuracy of the estimate. Then, we will describe adaptive variants of the protocol, and finally we will discuss the integration with the membership protocol.

4.1. Analysis of the interval density approach

Theorem 2. *In order to obtain a group size estimate that is accurate within a factor of $\delta < 2e - 1$ of the actual estimate with a very high probability (i.e., that tends to 1 as N tends to infinity), it suffices to choose an interval size I that varies as $\Theta(\log(N)/(\delta^2 \times N))$. Here, N is the actual number of non-faulty processes in the group.*

Proof. The proof uses Chernoff bounds. First, the expected number of processes that fall in the interval I would be $I \times N$, where the size of the interval is notated simply as I. The likelihood that the estimate for the group size is off by a factor of at most 2δ from the mean is calculated as:

$$\Pr[|X/I - N| < \delta N] = 1 - \Pr[X/I < (1-\delta)N] - \Pr[X/I > (1+\delta)N]$$

If $\delta < 2e - 1$, Chernoff bounds[1] can be used on the latter terms, giving:

$$1 - \Pr[X < I(1-\delta)N] - \Pr[X > I(1+\delta)N] \geqslant 1 - e^{-IN\delta^2/2} - e^{-IN\delta^2/4}$$

If $I > c\log(N)/N$, then the probability that the estimate is accurate would be bounded from below by:

$$(1 - e^{-c\log(N)\delta^2/2} - e^{c\log(N)\delta^2/4}) \cong 1 - \frac{1}{N^{c\delta^2/2}} - \frac{1}{N^{c\delta^2/4}}$$

Choosing c larger than $1/\delta^2$, suffices to reduce this error asymptotically to zero, as N is increased. Thus, to obtain an estimate that is accurate within a constant factor (i.e.,δ is a constant) w.h.p., it suffices if I is of length $O(\log(N)/(\delta^2 \times N))$. □

Of course, choosing larger interval sizes leads to a better accuracy in the estimate. For instance, if I were of length $O(\sqrt{N}/N)$, the above analysis gives us that the estimate is accurate with probability $(1 - e^{-c\sqrt{N}\delta^2/2} - e^{c\sqrt{N}\delta^2/4})$. This value asymptotically approaches 1 if $\delta = \sqrt{\log(N)}/N^{1/4}$, which gives a better accuracy than by using a logarithmic interval size.

Practically, of course, it is difficult to set the size of the interval as $O(\sqrt{N}/N)$, without a prior estimate of N. One alternative could be the following. Since the value of $(\sqrt{N}/N)$ decreases as N is increased, if a lower bound for

[1] Other results such as the de Moivre-Laplace theorem can be used to give tighter bounds – thus, our bounds calculated here are worst-case bounds.

N is known, the interval can be chosen to be large enough to guarantee a required level of reliability. However, for larger N, a very large number of processes may fall inside this interval, thus requiring memory usage at the initiator to grow linearly with group size. Alternatively, this drawback can be addressed by using strategies that adaptively set the interval size. These adaptive strategies also adjust to a bad initial choice for the location of the interval. They are described in the next subsection.

4.2. Adapting the size of the interval

In the continuous version of the Interval Density scheme, the size of the interval needs to be adapted on the fly in order to obtain more and more accurate estimates. An interval I is defined by its *center point* and its *size*. The initiator needs to select a center point for the interval, but the size of this interval itself can be *implicit*, rather than explicit. More specifically, the interval size is determined by remembering a *number* of process identifiers that hash close to the center point. This number is initially set to a small value, but is increased over successive runs until (a) either a predefined threshold of MAXMEMORY processes is reached, or (b) the estimates of group size obtained for successive values of interval length are within a small fraction (e.g., 5%) of each other. Notice that the passive nature of the protocol means that the initiator only selects the interval, but does not communicate it to any other process.

4.3. Adapting interval location, and using multiple estimation runs for better accuracy

In the continuous version of the Interval Density scheme, choosing a "good" center point for the interval affects (either positively or negatively) the accuracy of the estimate, since process identifiers inevitably hash in a rather non-uniform manner into the interval $[0,1]$. Notice that this non-uniform mapping of the identifiers may occur even if the group is static (and in spite of the use of cryptographic functions), thus it is important to converge to a center-point that yields a good and stable estimate that is not affected by the above types of variability. In dynamic groups, this behavior may also manifest over a long term since process arrivals and departures cause the hash map to change constantly. We describe three approaches for choosing a good interval center:

1. *Random selection.* This is the simplest approach where the initiator randomly selects the center point from the interval $[0,1]$, and then uses this value for all subsequent estimation runs.
2. *Periodically changing selection.* The initiator periodically (e.g., after a few runs) changes the center point of the interval. This could be done either independent of, or dependent on, previously chosen center points. We call the latter method *self-adjusting interval selection*.

   A good self-adjusting interval selection approach is the following. At every stage, history data for the results of a few recent estimation runs is maintained (say the recent 8 runs), along with the intervals that were used for them. The average estimate returned by these recent runs is calculated, and the center point (say C) of the run that returned an estimate closest to this average, is used in the next run, *except that it is changed by a small perturbation* δ (*i.e., the next value of* C *is either increased or decreased by the perturbation value, with equal probability*). The value of δ *is* chosen based on the consistent hash function used – for SHA-1, this is $\delta = \frac{1}{2^{160}}$ and for MD-5 this is $\delta = \frac{1}{2^{128}}$. Notice that this perturbation avoids stagnation of center-points and enables them to change all the time, yet the small perturbations ensure that when the system comes closer to convergence, the estimate will remain stable.Thus, the center of the interval would continuously move right and left in the $[0,1]$ interval after each round depending on the results of the estimations made on the last rounds. This algorithm will work well if the group size does not change dramatically between two successive estimation runs.
3. *Multiple selections and estimation by mean value.* Instead of changing the center of the interval over time, each estimation run includes multiple runs, each with the interval centered at a different point in $[0,1]$. The mean value of the multiple results is then returned as the estimate.

Our experiments in Section 5 compare the relative performance of the above three approaches.

4.4. Snooping on a membership maintenance protocol

We will describe now how the initiator passively collects information about process identifiers that lie inside the selected interval I by snooping on the membership protocol. Below, we explain how this snooping operates on a well-known class of membership protocols called all-to-all heartbeat-style membership. In a nutshell, these protocols lazily spread information about each node (joining, leaving, failing) to each other alive node (which in turn may or may not choose to remember it). A large class of membership protocols satisfy this requirement. For others that do not (e.g., Chord membership), it is easy to think of simple modifications to these protocols that will spread all membership information to all other nodes.

We describe in detail how to incorporate the Interval Density approach into the gossip-style membership protocol detailed by van Renesse et al. (1998). The basic gossip-style membership protocol has each process p periodically (a) increment its own heartbeat counter; (b) select some of its neighbors ("fanout" neighbors, selected from the membership list maintained at p), and send to each of these a membership gossip message containing its entire membership list, along with heartbeats. Each process receiving the message merges heartbeat values in the received message with its own membership list. If gossip messages are allowed to carry N heartbeats and identifiers, the time for a membership updated to spread to everyone is *O(log(N))*. These results were shown by van Renesse et al. (1998). Notice that this comes at the cost of only *constant message overhead* at each node, i.e., the overhead in messages per protocol period at each node is independent of group size.

Please also note that gossip does *not require a single message of N identifiers to be sent out at once to* "fanout" *random nodes.* Alternative schemes where c messages of N/c size each are sent out to $(c \times \text{fanout})$ nodes, and these transmissions are spread out over the course of a protocol period, have the same bandwidth, latency, and reliability as the monolithic message scheme.

In the modified version of this protocol, used by the Interval Density approach for snooping, a process receiving such a gossip message additionally hashes each previously unknown process identifier appearing in the message, and remembers it only if it lies in the interval I. If such a remembered entry is not refreshed (i.e., a newer heartbeat is not received via gossip) within a timeout T_{down}, then the entry is deleted (T_{down} values are specified during our experiments in Section 5). Hence, this keeps a running estimate of the number of process identifiers lying within I, and estimates the group size. Notice that the above incorporation is non-intrusive, i.e., it does not affect the normal working of the heartbeat-style membership protocol itself. The time required for the algorithm to complete is $O(\log(N))$, if gossip messages are allowed to carry up to N heartbeats and identifiers. If the gossip message length is restricted to carry $O(I)$ randomly chosen identifiers and heartbeats, the time complexity is $O((N/I) \cdot \log N)$.

Convergence-background overhead tradeoff: The convergence of the Interval Density scheme depends on the background bandwidth due to the underlying membership protocol. Suppose the given membership protocol imposes a background overhead of O messages per round per node (where round is a time unit specific to the membership protocol). Given this, the time of convergence of the Interval density scheme is, say, C seconds. For instance, from the above discussion, C may be $O(\log(N))$. The relationship between the two depends on the underlying membership protocol used. For the gossip-style heartbeating protocol of van Renesse et al. (1998), the relationship is an inverse one, i.e., $O \cdot C = \text{constant}$. The intuition behind this is straightforward – if nodes gossip and spread heartbeats twice as fast, the overhead is doubled and the convergence time is halved. Similarly, if nodes gossip half as fast, the overhead is halved and the convergence time doubled.

A final note that applies to both algorithms discussed – a membership protocol that keeps track of the entire list of peers in the system could itself be used to estimate group size. However, most membership protocols for large peer-to-peer overlays do not maintain an entire list due to memory usage and bandwidth constraints. Both presented algorithms require only partial membership lists; the Interval Density algorithm imposes an additional (small) memory requirement to remember some process identifiers.

5. Experimental results

We first evaluate the performance of the Interval Density and the Hops Sampling approaches under two simulation scenarios: (i) Micro-benchmarks, with a static group of a fixed size, and (ii) Trace-based experiments, using trace-log data for node availability, originally collected from the Overnet file-sharing network (Bhagwan, 2003).

In all these experiments, the main metric measured is the "Network Size", i.e., the number of nodes in the system, either estimated or actual. The reason for this is that other metrics are largely inapplicable for our study, e.g., a comparison of message overheads does not make sense since the Interval Density scheme is a clearly a winner as it is a passive approach, while Hops Sampling is an active approach.

The traces from the Overnet network (Bhagwan, 2003) measure the availability of nodes in a 3000-sized subset of hosts in the deployed Overnet peer-to-peer system. In these traces, the number of hosts that are present in the system changes by as much as 10–30% per hour (counting node joins and leaves separately). We first present individual studies for each of the Hops Sampling algorithm and Interval Density approach, and then compare them against each other.

Our experiments in this section and the next are motivated by two goals: (1) we wish to compare the accuracy of the active and passive approaches, *all other things being equal;* (2) we are not interested in the convergence times of these algorithms (since these can be speeded up by increased bandwidth), but rather the *accuracy* of these algorithms after convergence. Below, we first discuss (ξ.1) the metrics we use for measuring the accuracy of the estimation protocols, and then (ξ.2) the foreground and background bandwidth consumption due to our protocols.

ξ.1 Metrics for measuring accuracy of the estimation protocols

In order to better quantify the accuracy of estimators, we define two accuracy metrics. For a given experiment with estimator algorithm A (where A could be either the Hops Sampling approach, or the Interval Density with one of the interval selection schemes of Sections 4.2, 4.1–4.3), we wish to quantitatively compare the accuracy of the estimator algorithm A against the actual value of the group size. We would also like the metrics to extend to both static and dynamic systems.

Towards this, we define the following two metrics: RMSE and StdDevErr. Before we elaborate below on each metric, we need a couple of definitions. Let $(x_1^A, x_2^A, \ldots, x_m^A)$ be the estimates returned by m consecutive runs of algorithm A, where x_i^A is the estimate returned by A at (global) time instant i. Similarly, let $(N_1, N_2, \ldots, N_m)$ be the respective values of actual group size at the respective time instants i. Notice that in a static system, all N_i's will have the same value (=N).

I. *Root mean square error* (RMSE): This metric is used in order to measure how *far* a given algorithm A's returned estimate is from the actual value of group size. We define the root mean square error of algorithm A as:

$$\mathrm{RMSE}(A) = \sqrt{\frac{\sum_{i=1}^{m}(x_i^A - N_i)^2}{m}} \tag{1}$$

In addition, for static groups (i.e., all $N_i = N$) we also introduce a *normalized* version of the above two metrics:

$$\mathrm{RMSE} - \mathrm{Norm}(A) = \mathrm{RMSE}(A)/N \tag{2}$$

Note that these *-Norm metrics (Eq. (2)) do not apply to dynamic groups.

II. *Standard deviation of error* (StdDevErr): This metric is useful to measure how *consistent* a given algorithm A's estimate is from the actual value of group size. We define this first, then explain its difference from the RMSE metric. The standard deviation of algorithm A is defined simply as the standard deviation of the set $(x_1^A - N_1, x_2^A - N_2, \ldots, x_m^A - N_m)$.

RMSE versus StdDevErr: RMSE and StdDevErr are orthogonal metrics, and both are measures of accuracy of an estimator algorithm. Intuitively, RMSE denotes how far algorithm A's estimate is from the real value, while StdDevErr shows how consistent A's estimate is. A good estimator algorithm should have low values of both RMSE(A) and StdDevErr(A). However, one property does not imply the other: for example, an algorithm that always returns an estimate that is lower than N by a fixed constant value is consistent but may be far: thus it will have a zero StdDevErr value but may have a high RMSE. On the other hand, an estimator algorithm may return estimates with a distribution centered around N but with a high standard deviation – this estimator is not far but may not be consistent – thus it will have a low value of RMSE, but a high value for StdDevErr.

Both RMSE and StdDevErr are important metrics – choosing between them depends on the application using the estimator. An application that would like the returned estimate to be always "close" to the actual group size would desire a low RMSE, while a low StdDevErr would be preferred by an application that would like the returned estimate to "shadow" the actual group size, i.e., show similar variation trends as the group size. StdDevErr is especially useful in a dynamic group, thus we consider it to be more important than RMSE.

ξ.2 Bandwidth consumption– minimal, implementation, and low achievable numbers

We now calculate the bandwidth usage for the Hops Sampling and Interval Density schemes. The per-node bandwidth for any estimation scheme has two components – a *background* component arising out of the underlying membership service, and a *foreground* component for the estimations themselves (we assume one-shot estimation only). The foreground component is thus expressed as messages per estimation, and does not depend on the underlying membership protocol used. The background component is in Bps (Bytes per second), and depends on the actual underlying membership mechanism.

(a) *Foreground bandwidth*: Since Interval Density is a passive scheme, the foreground component for it is 0 messages per estimation. The foreground component for Hops Sampling is = (`gossipTo` × `gossipFor`) messages per estimation; for our parameter settings, this is 2 messages per estimation.

It is important to observe that if the Hops Sampling approach or Interval Density scheme is used in a system that *already has an appropriate underlying membership mechanism,* i.e., one that helps to select random targets (if Hops Sampling is the scheme), or one that spreads membership information to everyone (if Interval Density is the scheme), then the foreground bandwidth consumed above is the *sole bandwidth due to the estimation algorithms.*

(b) *Background Bandwidth*: We analyze the background bandwidth below for each algorithm. Here, we will discuss three types of bandwidth numbers (for each estimation scheme) – (1) the minimal possible, (2) for our chosen experimental implementation, and (3) low achievable bandwidth using alternative underlying membership approaches (different from our implementation). During this, we will also mention the exact underlying membership mechanism used in our experiments.

(b.1) *Hops Sampling Background bandwidth – minimal, implementation, and low achievable*: In order to support Hops Sampling, the underlying membership mechanism needs to provide, *at the minimum*, a capability for each node to sample (`gossipTo` × `gossipFor`) random nodes locally, i.e., without sending any extra messages for the random selection. This means the node needs to maintain a local list that is a random subset of other nodes in the system. In our experiments, our Hops Sampling approach uses a scheme where each joining node's membership update is *directly* sent to a random subset of $\log_2(N)$ other nodes in the system, and each entry of the random subset is deleted the first time it is picked as target but does not respond back. Thus, for this scheme, our implementation bandwidth is in fact the same as the minimal bandwidth. However, this best-case choice is realistic for our goals since it does not affect the accuracy of the algorithms (only their convergence time, which is not of primary interest).

For the *minimal* bandwidth, in a system of $N = 1$ million nodes, with an average 26.7% churn rate per hour (counting node joins and leaves separately, from (Bhagwan, 2003)), each node will then receive, at minimum, an average notifications per hour of $(0.267 \times 20 \times 0.5)$, counting only node joins. With 8 B node identifiers, this is 0.0059 Bps. Although this minimal bandwidth is also the *implementation* background bandwidth in our case, alternative membership protocols can be used to achieve the same, but with only slightly higher bandwidths. For instance, using a system like SCAMP (Ganesh et al., 2001) would cause a log(N) blowup in these numbers. This is because, in SCAMP, since each node maintains a log(N) – sized membership list of random nodes, and each new notification (about a new node only; failed node entries time out) goes via an expected (log(N) × log(N)) (random) nodes – for details, please see work by Ganesh et al. (2001). Assuming a $\log_2(N)$-fold (20-fold) blowup in bandwidth, this implies a *low achievable* background bandwidth (counting only node joins again) of only (0.0118 × 0.5 × 20) Bps = 0.118 Bps per node.

(b.2) *Interval Density Background bandwidth – minimal, implementation, and low achievable*: For supporting Interval Density, the underlying membership mechanism needs to provide *at the minimum*, a scheme that spreads information about all membership updates (both node joins and leaves) to all other nodes in the system (of course, as mentioned earlier, only a fraction I of such received entries will actually be remembered, where I is the size of the interval). However, our implementation uses the gossip-style membership protocol of Section 4.4 underneath. At the same time, there are other, more efficient approaches that can be used underneath. Below, we analyze the background bandwidth consumed by each of these three individually.

With a 1 million system with 26.7% churn per hour (from (Bhagwan, 2003)) and 8 B node identifier entries, the *minimal* background bandwidth per node is (0.267 × 1 M × 8 B) per hour, which is 593.33 Bps. In our implementation, the Interval Density scheme uses the gossip-style membership protocol described in Section 4.4. This gives an implementation background bandwidth per node of N node identifier entries, spread out over a protocol period length. Assuming 8 B per entry and N = 10,000 (the maximum value in our experiments), and a protocol period of time length, say, 1 min, we get an *implementation* background bandwidth consumption of 1.33 KBps. Notice that all the "time" units on our plots in Sections 5 and 6, are in terms of protocol periods, and a minute-long protocol period implies that the time to have a new membership entry spread to all nodes w.h.p is O(log(N)) minutes. Thus, the latency is low, but the resulting bandwidth may be acceptable only for medium group sizes (the ones considered in our simulations).

However, alternative schemes can be used to achieve much lower bandwidth, although the scalability is still limited compared to the Hops Sampling approach. For instance, one could spread use an underlying protocol that disseminates *updated (both joins and leaves) membership entries* only (not *all* membership entries, as in the protocol of Section 4.4) using gossip, e.g., akin to the rations of the Kelips system (Gupta et al., 2003). Since each updated entry is gossiped log(N) times by each node, the *low achievable* bandwidth is thus a factor of log(N) above that of the minimal bandwidth calculated above. Assuming a $\log_2(N)$-fold (20-fold) blowup in bandwidth then implies an average bandwidth, for a very large 1 million node system, of 11.866 KBps per node. Firstly, notice that if the underlying distributed system is already running the membership protocol (as would be the case with the Kelips system, for instance), then this membership information is already being used for other purposes (e.g., peer-to-peer discovery in Kelips), and thus the added bandwidth cost of running the passive Interval Density algorithm is zero. On the other hand, if there is no appropriate membership protocol running underneath, then Interval Density's scalability (bandwidth-wise) depends on the efficiency of the additional membership protocol. For instance, by adding the Kelips-like ration mechanism, one could easily scale to a 100,000 node system, with an added per-node bandwidth that is = (0.267 × 100 K × 8 B × $\log_2$(100 K)) B per hour = 985.5 Bps, which is so low as to be smaller than most modem capacities.

Yet, Interval Density is in fact *fundamentally* scalable, since its minimal bandwidth per node is 593.33 Bps even for a 1 million node system. As more and more efficient membership algorithms are discovered in the community – this is an active area of research – Interval Density can be extended to more systems that are not already running the appropriate membership protocol.

5.1. Hops Sampling approach

Micro-benchmarks: As pointed out earlier, the values assigned to the `gossip*` parameters are not magic values. However, in order to empirically derive the logarithm base for one setting of parameters, we choose the parameter settings of `gossipTo` = 2 and `gossipFor` = 1. Fig. 2 plots the average number of hops measured in a static network with N nodes, versus $\log_2 N$, for different group sizes. The

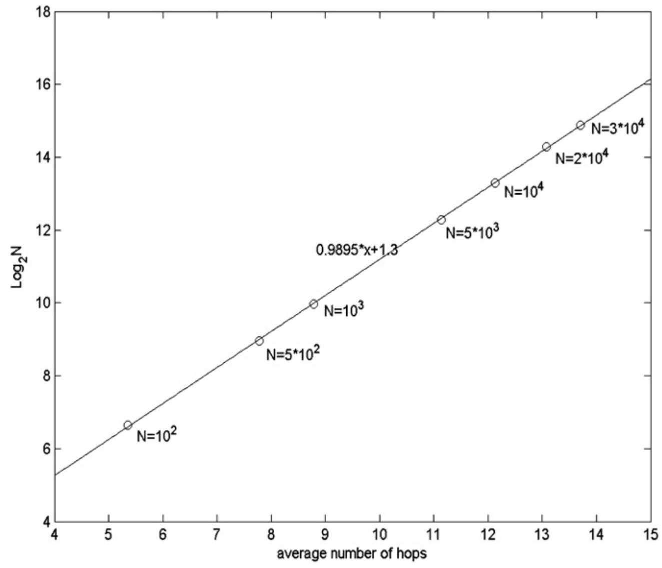
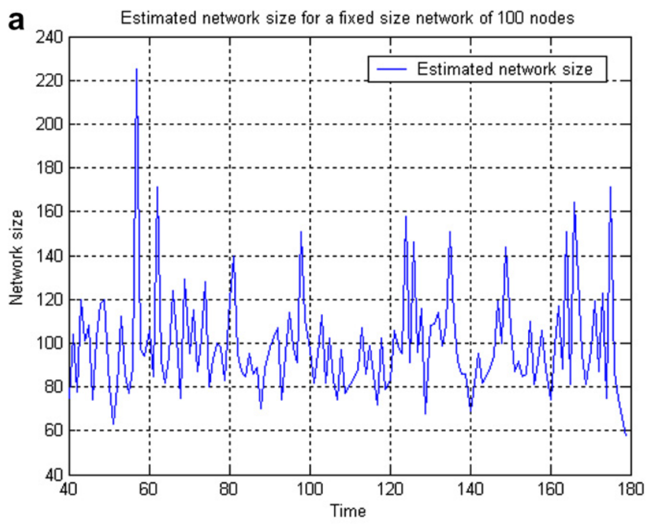


Fig. 2. Average number of hops versus $\log_2 N$ for groups with sizes of 100–30,000 nodes.

straight line shown in the figure is given by $\log_2 N = 0.9895 * \texttt{averageNumberOfHops} + 1.3$, and appears to hold for group sizes up to 30,000. The estimated group size is thus calculated as $SizeEst = 2^{(0.9895*avHops+1.3)}$.

This experiment also brings out a disadvantage of the Hops Sampling approach (the Interval Density scheme does not suffer from this disadvantage). The base of the logarithm (hence the constants in the equation) are related to gossip* parameter values, as well as the topology of the overlay itself. The constant 1.3 is thus calculated only for this experimental setting – our deployment experiments required us to recalibrate the constants (see Section 6). However, once Hops Sampling is calibrated for a particular setting, the estimator's accuracy stays stable for that scenario. Our deployment experiments with the PeerCounter deployment in Section 6 confirm this.

Fig. 3a shows the variation, over time, of the estimated group size, for a static group of size 100. Fig. 3b shows the effect of averaging over several estimation runs; the continuous protocol version parameters used are gossipsAccounted = 10 and gossipsDropped = 2. The benefit of the latter is clear from the figures – both its RMSE-Norm and StdDev-Err numbers are smaller. Furthermore, the estimate in Fig. 3b stays within 10% of the actual group size most of the time.



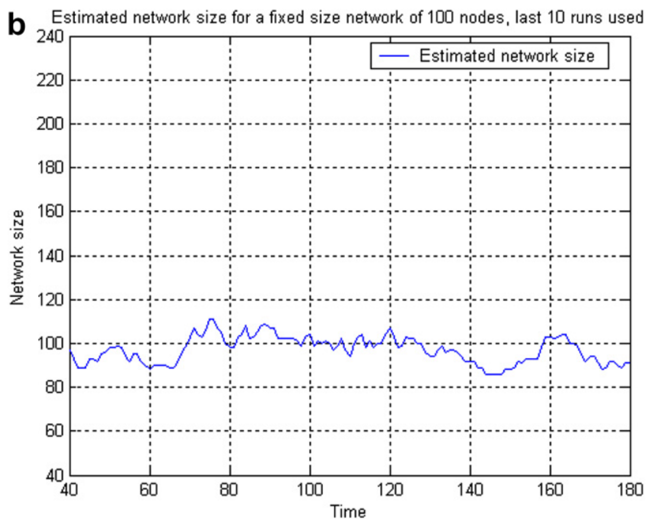


Fig. 3. Hops Sampling size estimation for a static sized network with 100 nodes: (a) One estimate; (b) Average over last 10 estimates. For (a), RMSE-Norm = 0.2397 and StdDev-Norm = 0.2404. For (b), RMSE-Norm = 0.0673 and StdDevErr-Norm = 0.0607.

Trace-based simulations: The hourly Overnet traces are injected into the simulator at time intervals of 40 timeslots. By "injection", we mean that the status of the system nodes is updated (as "up" or "down") at the timeslot we use an Overnet trace. Each time unit of the above plot corresponds to 40 timeslots. The continuous protocol is used, with parameters gossipTo = 2, gossipFor = 1, gossipsAccounted = 10 and gossipsDropped = 2. Fig. 4 shows the variation, over time, of the estimated group size. The RMSE and StdDevErr numbers are both low but not small – dividing each by 500 (the "approximate" group size) gives "normalized" values of 0.1883 and 0.1706, respectively. As expected, this is higher than in the case of static groups (Figs. 2 and 3).

5.2. Interval Density approach

As described in Section 4.4, Interval Density messages are piggybacked on a gossip-style membership protocol such as by van Renesse et al. (1998). The membership protocol by van Renesse et al. (1998) uses a timeout value for deleting entries relating to processes that have not sent a heartbeat message for a while. In our implementation of that membership protocol, we use such a parameter and call it T_{down} time units. In brief, membership entries for processes lying within the interval time out and are marked for deletion if updated heartbeats have not been received for T_{down} time units. Marked entries are deleted after another T_{down} time units; $T_{\text{down}} = 10$ rounds and MAXMEMORY = 60 for simulations presented in this section.

Micro-benchmark: static sized groups: Fig. 5 shows the variation, over time, of the estimate in a static sized group with 10,000 nodes. The random selection approach is applied to decide on the interval used for estimation. The plot shows that the estimate's RMSE and StdDevErr values are smaller than the Hops Sampling approach's. These numbers can be further reduced by increasing the T_{down} value. A smaller T_{down} value causes early expiry of some process entries in the modified gossip-based heartbeat protocol. Fig. 6 shows the corresponding timeline when T_{down} = infinity, so that membership entries never expire. The convergence is within a factor of 5% of the actual group size. As expected, the RMSE-Norm and StdDev-Err-Norm numbers are thus much lower in Fig. 6 compared to Fig. 5.

Figs. 5 and 6 also reveal an interesting property of the Interval Density approach. When membership timeouts (T_{down}) are finite, the estimate is likely to be below the real group size, since some membership entries that lie in the interval are not considered (as they expire). However, at

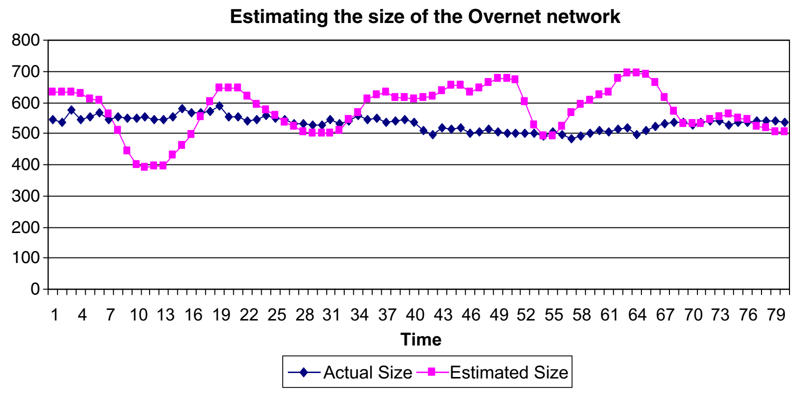


Fig. 4. Hops Sampling size estimation for Overnet network. RMSE = 94.1327 and StdDevErr = 85.2875.

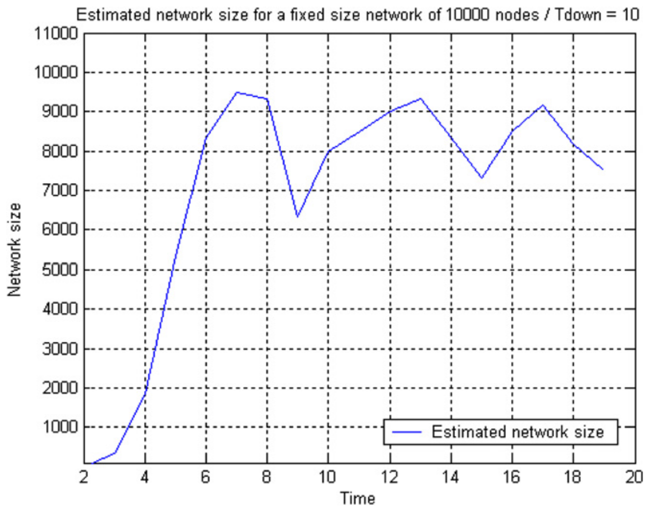


Fig. 5. Interval Density size estimation (random interval selection) for a static sized network of 10,000 nodes, $T_{\text{down}} = 10$. Ignoring estimates before convergence (time $t < 6$), RMSE-Norm = 0.1846 and StdDevErr-Norm = 0.0929.

large or infinite values of T_{down}, the estimate can be offset by the local density of hashed node identifiers in the chosen interval, and thus may also be higher than the value of N.

Finally, comparing Fig. 6 with Fig. 3, we observe that *for static groups, Interval Density gives far better RMSE and StdDevErr than Hops Sampling*.

Trace-based simulations – effect of adaptive mechanisms: Figs. 7–9 show the behavior, over Overnet traces, of the three adaptive approaches for the selection of interval, presented in Section 4.3. All estimates by all approaches appear to lie within 20% of the actual group size. Among these three approaches, the "self-adjusting interval selection" (Fig. 8) produces the lowest value of RMSE, while the lowest StdDevErr results from the "multiple selections estimation by mean value" (Fig. 9). These two are thus the best approaches – we consider StdDevErr (consistency) to be more important than RMSE (distance). Notice how the lower StdDevErr numbers in Fig. 9 correspond to a closer resemblance between the "shapes" of the actual and estimated group sizes. Based on these observations, we will use the self-adjusting interval selection for only the next

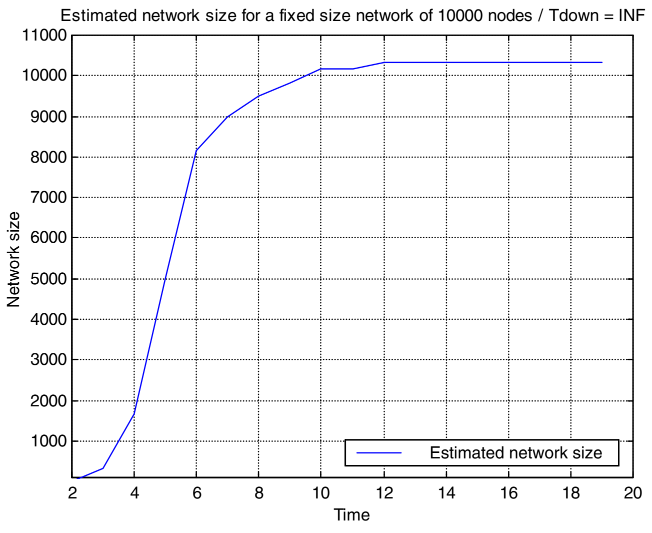


Fig. 6. Interval Density size estimation (random interval selection) for a static sized network of 10,000 nodes, $T_{\text{down}} = \text{infinity}$. Ignoring estimates before convergence (time $t < 10$), RMSE-Norm = 0.0269 and StdDevErr-Norm = 0.0070.

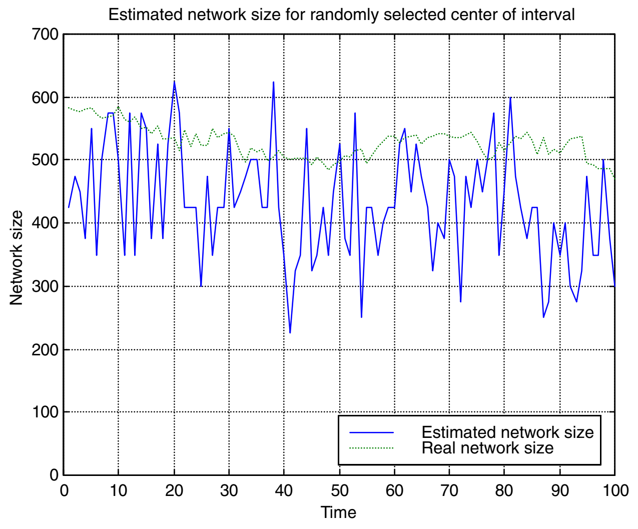


Fig. 7. Interval Density estimate calculated by the random interval selection approach. RMSE = 131.1893 and StdDevErr = 89.1347.

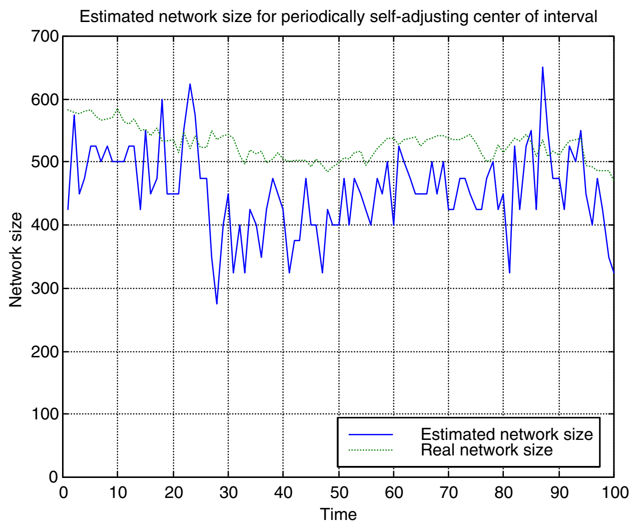


Fig. 8. Interval Density estimate calculated by the self-adjusting interval selection approach. RMSE = 94.9638 and StdDevErr = 61.4406.

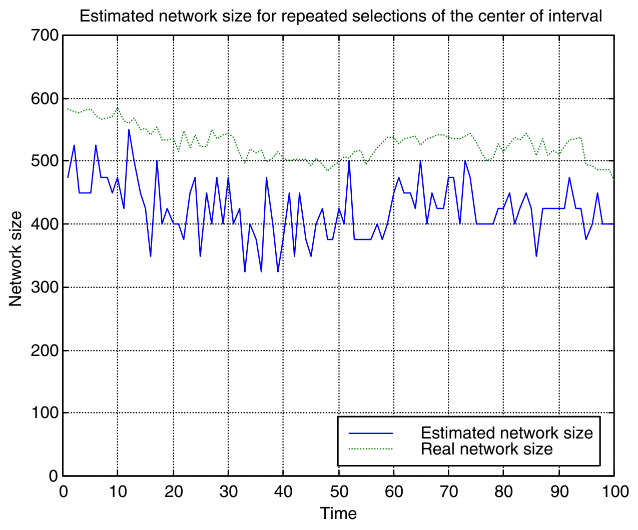


Fig. 9. Interval Density estimate calculated by the estimation by mean value approach. RMSE = 110.8349 and StdDevErr = 38.7300.

experiment, but thereafter we shall consider only the multiple selections estimation by mean value approach.

Highly dynamic groups: Although the fraction of nodes joining and leaving the system is as high as around 25% in the Overnet traces, the reader would have observed from the previous experiments that the actual group size itself does not vary much. To model a more dynamic group, we modify the Overnet traces so that an additional set of arrivals and departures is added to the original ones, causing the number of nodes in the network to vary very highly between time intervals. The self-adjusting method is used for providing estimations. The results are presented in Fig. 10, and show good behavior (RMSE and StdDevErr) in spite of highly dynamic groups. A look at Fig. 10 also shows that the estimate closely follows the variation of the actual group size over time – although it does not make

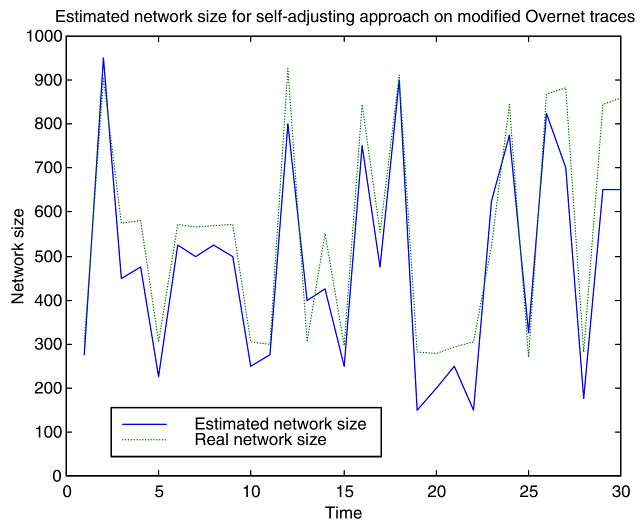


Fig. 10. Interval Density – behavior of self-adjusting interval selection in a highly dynamic group. RMSE = 101.6384 and StdDevErr = 82.2710.

sense to calculate normalized values of RMSE and StdDevErr for a dynamic group (both around 100), it is easy to see the absolute values are small compared to the peaks in actual group size (around 900).

5.3. Hops Sampling algorithm versus Interval Density algorithm

Although it is evident from Sections 5.1 and 5.2 that the Interval Density scheme is superior, the Hops Sampling scheme can be caused to perform comparable to the Interval Density scheme by choosing a good setting of parameters. We experimented with a wide variety of parameter settings for Hops Sampling and settled on the “best” choice. The comparisons of this “best” Hops Sampling with Interval Density are below.

Figs. 11 and 12 compare the proposed approaches for static (fixed size) and dynamic (Overnet-trace based) groups. For the Interval Density scheme, we use the “Multiple Selections by Mean Value” approach consistently (i.e., approach number 3 in Section 4.3). The gossip parameter settings for Hops Sampling are `gossipTo` = 2, `gossipFor` = 1, `gossipsAccounted` = 10 and `gossipsDropped` = 2. In Fig. 11, `MAXMEMORY` = 60 and T_{down} = infinity in Fig. 11. The self-adjusting method, with `MAXMEMORY` = 120 and $T_{\text{down}} = 10$, is used for Fig. 12.

For static sized groups (Fig. 11), both the Interval Density and the Hops Sampling approaches converge quickly to within 5% of the actual group size (Fig. 11). However, the Interval Density scheme performs slightly better than the Hops Sampling approach because it has a slightly lower RMSE-Norm value and a much lower StdDevErr-Norm value. In a dynamic group (Overnet churn traces injected, Fig. 12), the estimate from the Interval Density approach has a smaller RMSE than the Hops Sampling approach,

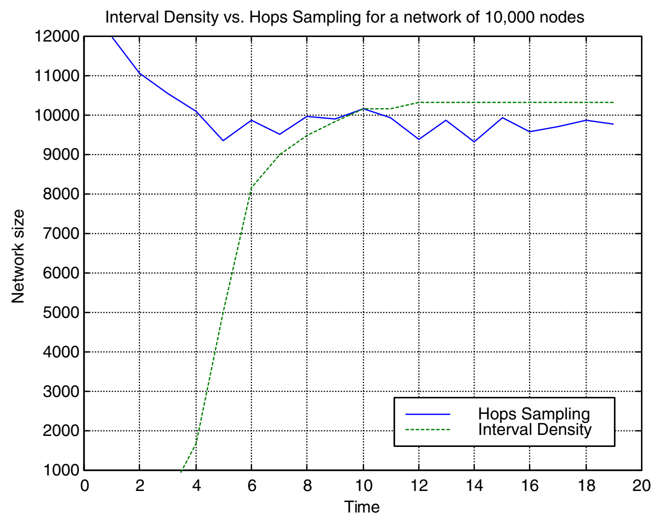


Fig. 11. Comparing accuracy of Interval Density and Hops Sampling approaches for large fixed size network. Ignoring estimates before convergence (time $t < 10$), RMSE-Norm(Hops) = 0.0347, StdDevErr-Norm(Hops)= 0.0260; and RMSE-Norm(Int.Dens.) = 0.0307, StdDev-Err-Norm(Int.Dens.) = 0.0070.

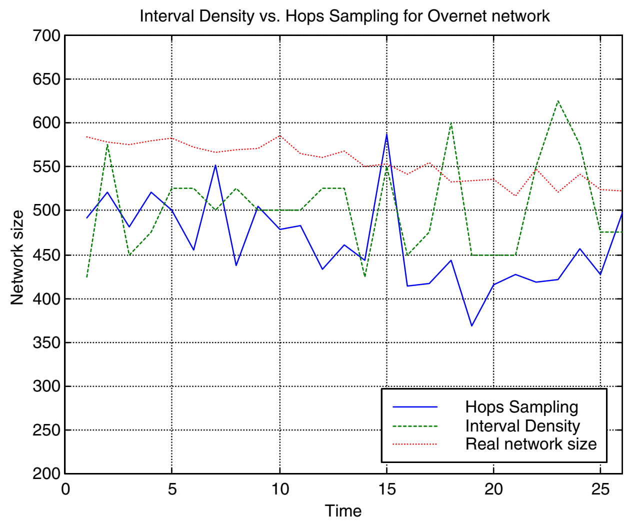


Fig. 12. Comparing accuracy of Interval Density and Hops Sampling approaches for Overnet traces. RMSE (Hops) = 100.2036, StdDevErr (Hops) = 42.3256; and RMSE (Int.Dens.) = 77.0360, StdDevErr-Norm(Int.Dens.) = 58.6015.

but the Hops Sampling approach gives a (slightly) lower StdDevErr than the Interval Density scheme. This StdDev-Err result is explained by the sensitivity of the Hops Sampling to its parameter settings – a fortunately good setting of parameters will result in a consistent estimate, the case here.

6. PeerCounter: a size estimation utility

In this section, we present PeerCounter; a generic module containing both algorithms presented in this paper which can be incorporated into an arbitrary appropriate peer-to-peer overlay. We briefly outline Peer-Counter's Application Programming Interface (API) and then demonstrate evaluation results obtained by applying this prototype implementation of our estimation schemes on a cluster of 33 Linux servers located in the University of Illinois. For our evaluation analysis we use a topology generator script in order to set up membership lists used for gossiping. Any synchronization assumptions made in our analysis or simulations are relaxed here since Peer-Counter does not require any synchronization among processes.

6.1. PeerCounter's API

PeerCounter is a command line application implemented in Java. It takes as parameters the server port to run the application, a parameter indicating which of the two algorithms, the Hops Sampling or the Interval Density, is used for estimation and the membership list of the calling process. The membership list can be generated using a topology generator utility or, in case PeerCounter is incorporated into a standard overlay, this may be provided by the underlying application. We thus use a static membership list for our experiments. PeerCounter's API can be used to let applications layered above it utilize its estimation schemes. PeerCounter exports the operations listed in Table 1.

6.2. Performance results

The Hops Sampling scheme: The parameters used here are `gossipTo` = 2, `gossipFor` = 1, `minHopsReporting` = 4, `gossipsAccounted` = 5 and `gossipsDropped` = 1. We test both the standard estimation scheme, which uses the average hop distance from processes sampled by the initiator (as presented in Fig. 1), as well as the alternative sampling scheme, mentioned in Section 3, where hop count values reported back at the initiator by processes with certain myHopCount values, are taken into account for size estimation. The equation used for the first case is $size = 2^{(averageNumberOfHops+1.75)}$. Notice that the values of the constants in this equation are different from those used in Section 5.1 (Fig. 2). This is because the Hops Sampling approach requires recalibration to the choice of parameter values and the overlay topology. It will be evident from the experiments that once a setting has been recalibrated, the estimates returned are quite stable.

Fig. 13 demonstrates the estimates provided by Peer-Counter's Hops Sampling approach for static groups. Two variants of the algorithms are compared – the darker diamonds show "Estimation by average number of hops" (annotated as "Hops"), and the lighter squares show "Estimation by sum of values from processes with same myHop-Count" (annotated as "myHopCount"). Three group sizes are considered – 80 nodes, 806 nodes, and 6430 nodes. Table 2 summarizes the RMSE-Norm and StdDevErr-

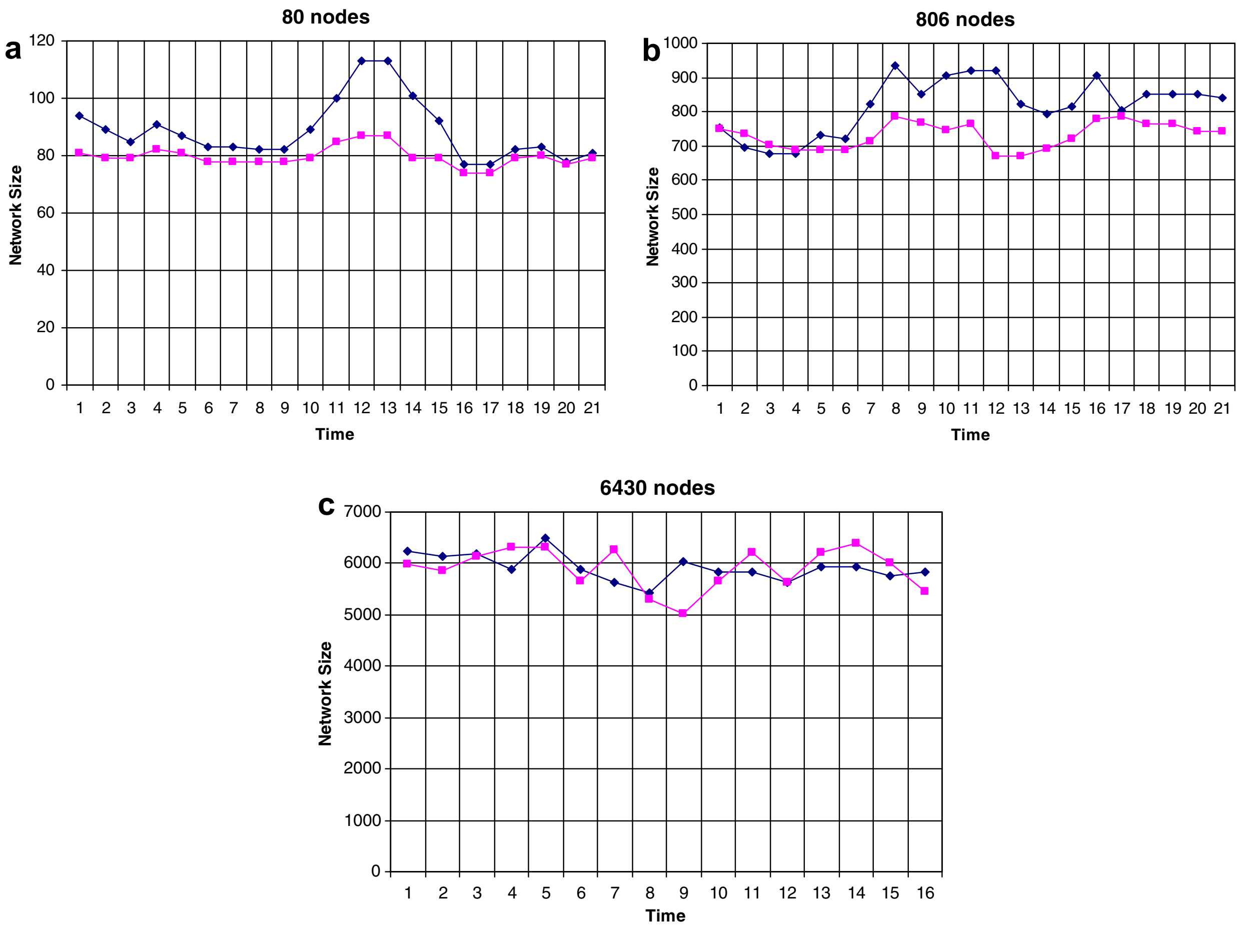


Fig. 13. PeerCounter Estimates using the Hops Sampling scheme on: (a) A static group of 80 nodes; (b) A static group of 806 nodes; (c) A static group of 6430 nodes. Two variants are compared – the darker diamonds show "Estimation by average number of hops," and the lighter squares show "Estimation by sum of values from processes with same myHopCount." The RMSE and StdDevErr numbers for these plots are shown in Table 2.

Norm from Figs. 13a–c. For large group sizes ($N > 800$), using the average number of hops metric performs slightly better than the myHopCount metric, when considering the RMSE-Norm metric. However, the performance on StdDevErr-Norm is about the same for both schemes. Finally and most importantly though, notice that for $N = 806$ and $N = 6430$, each of the two schemes (Hops and myHopCount) have stable values of the RMSE-Norm metric, and low values of the StdDevErr-Norm metric – this illustrates convergence of behavior as the group size is increased.

The Interval Density scheme: Fig. 14 shows the estimates provided by PeerCounter's for static groups of different sizes when the one-shot flavor of the Interval Density approach is used and a random selection of the interval is made. Since different group sizes are used, we do not calculate the RMSE and StdDevErr metrics for this experiment. The parameters used here are $\mathtt{MAXMEMORY} = 100$ and $T_{\text{down}} = \text{infinity}$. As shown previously in Section 5, the accuracy of estimations is expected to be better when adaptive methods or multiple selections and estimation by mean value are applied in order to choose the interval location. Moreover, a continuous running of the application should cause the estimates to converge to a value near the actual group size, when membership timeouts are ignored, as studied earlier in this paper. Fig. 15, which shows results for the continuous flavor of the Interval Density approach, indicates that this actually holds for

Table 2
Comparing the behavior of the two PeerCounter Hops Sampling metrics from Figs. 13a–c

Group size	RMSE-Norm(Hops)	StdDevErr-Norm(Hops)	RMSE-Norm(myHopCount)	StdDevErr(myHopCount)
80	0.1677	0.1311	0.0416	0.0424
806	0.0980	0.0995	0.1030	0.0473
6430	0.0891	0.0410	0.1025	0.0634

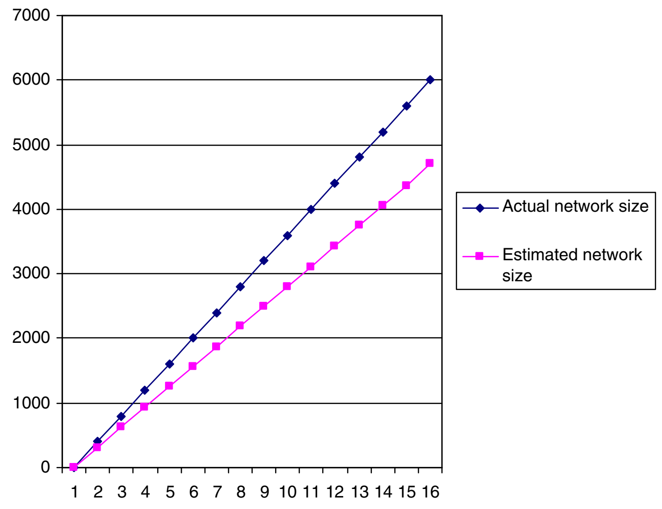


Fig. 14. PeerCounter estimates for one-shot Interval Density scheme with random interval selection.

our prototype implementation as well. For Fig. 15a, both the RMSE and StdDevErr are equal to 0, when one ignores datapoints before convergence (i.e., time $t < 5$); the corresponding numbers for Fig. 15b are small.

7. Conclusions

Estimating the size of a decentralized group of processes connected within an overlay (e.g., in a peer-to-peer system, or in the Grid) is a difficult problem, especially when the group is dynamic and contains thousands of processes. Several previous solutions to the problem make assumptions that may be either unscalable or limit applicability. In this paper, we have proposed and compared two new approaches for estimation – an active approach called Hops Sampling and a passive approach called Interval Density. The only requirements for these algorithms are the existence of complementary membership protocols, which are already present as a part of many overlays. Both approaches require the overlay graph among all processes in the system to be connected – in addition, Hops Sampling requires a random sampling service, while Interval Density requires an all-to-all heartbeating membership protocol underneath. Several existing systems can be leveraged to satisfy these assumptions.

Micro-benchmarks (for static groups) and trace-based simulations have shown that both above approaches are able to obtain an estimate close to the actual group size. In general, both the approaches have low Root Mean Square Errors (RMSE – thus the estimate returned is *close* to the actual group size) and also a low Standard Deviation in Error (StdDevErr – thus they shadow and are consistent with the variation of group size in dynamic groups). For static groups, passive Interval Density is better than active Hops Sampling on both RMSE and StdDevErr. For dynamic groups, passive Interval Density is comparable to active Hops Sampling on RMSE, but better than Hops Sampling on StdDevErr.

The Hops Sampling scheme can be caused to perform almost as well as the Interval Density scheme by trying out different parameter values and settling on a "good" choice. However, this is a challenging task. In general, in a scenario where a distributed application already has a membership protocol already running, the passive Interval Density appears to be more preferable because it adds no extra overhead. In addition, it has fewer assumptions than active Hops Sampling (i.e., it does not need a Random Sampling Service).

The PeerCounter utility, implementing these algorithms, is available at kepler.cs.uiuc.edu/~psaltoul/peerCounter as open-source software. PeerCounter can be incorporated as a part of any arbitrary peer-to-peer overlay.

PeerCounter Software for Download: kepler.cs.uiuc.edu~psaltoul/peerCounter.

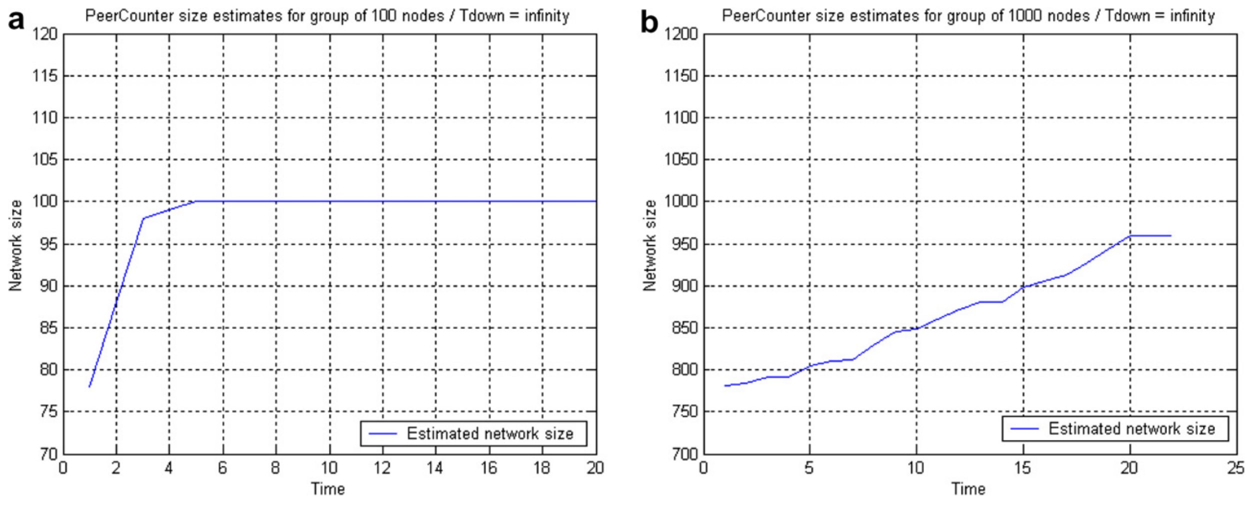


Fig. 15. PeerCounter estimates for groups of 100 and 1000 nodes for the continuous Interval Density scheme with random interval selection. (a) Static group with 100 nodes, (b) static group with 1000 nodes.

References

Alouf, S., Altman, E., Nain, P., 2002. Optimal on-line estimation of the size of a dynamic multicast group. In: Proceedings of IEEE INFOCOM '02, New York, NY, USA.

Alouf, S., Altman, E., Barakat, C., Nain, P, 2003. Estimating membership in a multicast session. In: Proceedings of ACM SIGMETRICS '03, San Diego, CA, USA.

Awerbuch, B., Scheideler, C., 2004. Robust distributed name service. In: Proceedings of the 3rd International Workshop on Peer-to-Peer Systems (IPTPS), La Jolla, CA, USA.

Bailey, N.T.J., 1975. Epidemic Theory of Infectious Diseases and its Applications, second ed. Hafner Press.

Bawa, M., Garcia-Molina, H., Gionis, A., Motwani, R., 2003. Estimating aggregates on a peer-to-peer network. Technical Report, Dept. of Computer Science, Stanford University.

Bhagwan, R., 2003. Overnet availability traces. ramp.ucsd.edu/projects/recall/download.html.

Birman, K.P., Hayden, M., Ozkasap, O., Xiao, Z., Budiu, M., Minsky, Y., 1999. Bimodal multicast. ACM Transactions on Computer Systems 17 (2), 41–88.

Das, A., Gupta, I., Motivala, A., 2002. SWIM: Scalable Weakly-consistent Infection-style process group Membership protocol. In: Proceedings of the 2002 IEEE International Conference on Dependable Systems and Networks, pp. 303–312.

Demers, A., Greene, D., Hauser, C., Irish, W., Larson, J., Shenker, S., Sturgis, H., Swinehart, D., Terry, D., 1987. Epidemic algorithms for replicated database maintenance. In: Proceedings of the 6th ACM Symposium on Principles of Distributed Computing (PODC), pp. 1–12.

Eugster, P.Th., Guerraoui, R., Handurukande, S.B., Kermarrec, A.M., Kouznetsov, P., 2003. Lightweight probabilistic broadcast. ACM Transactions on Computer Systems 21 (4), 341–374.

FIPS 180-1, 1995. Secure Hash Standard. NIST, US Department of Commerce, Washington, DC.

Friedman, T., Towsley, D., 1999. Multicast session membership size estimation. In: Proceedings of IEEE INFOCOM '99, New York, NY, USA.

Ganesh, A.J., Kermarrec, A.-M., Massoulie, L., 2001. SCAMP: Peer-to-peer lightweight membership service for large-scale group communication. In: Proceedings of the 3rd International Workshop on Networked Group Communication (NGC). LNCS, vol. 2233. Springer, pp. 44–55.

Godfrey, P.B., Stoica, I., 2005. Heterogeneity and load balance in distributed hash tables. In: Proceedings of IEEE Infocom.

Gupta, I., Birman, K.P., Linga, P., Demers, A.J., van Renesse, R., 2003. Kelips: building an efficient and stable P2P DHT through increased memory and background overhead. In: Proceedings of the 2nd International Workshop on Peer-to-peer Systems (IPTPS). LNCS, vol. 2735. Springer, pp. 160–169.

Jelasity, M., Babaoglu, O., 2005. T-Man: Gossip-based overlay topology management. In: Brueckner, S.A. (Ed.), Engineering Self-Organising Systems: Third International Workshop (ESOA 2005), Revised Selected Papers, LNCS, vol. 3910. Springer, pp. 1–15.

Jelasity, M., Montresor, A., 2004. Epidemic-style proactive aggregation in large overlay networks. In: Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS), Tokyo, Japan.

Jelasity, M., Preuss, M., 2002. On obtaining global information in a peer-to-peer fully distributed environment. In: LNCS, vol. 2400. Springer, pp. 573–577.

Jelasity, M., Guerraoui, R., Kermarrec, A.-M., van Steen, M, 2004. The Peer Sampling Service: experimental evaluation of unstructured gossip-based implementations. In: Proceedings of the ACM/IFIP/USENIX 5th International Middleware Conference (Middleware), Toronto, Canada.

Karp, R.M., Schindelhauer, C., Shenker, S., Vocking, B., 2000. Randomized rumor spreading. In: Proceedings of the IEEE Symposium on Foundations of Computer Science, 565–574.

Kempe, D., Dobra, A., Gehrke, J., 2003. Gossip-based computation of aggregate information. In: Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science (FOCS), p. 482.

Kostoulas, D., Psaltoulis, D., Gupta, I., Birman, K.P., Demers, A.J., 2005. Decentralized schemes for size estimation in large and dynamic groups. In: Proceedings of the 4th IEEE International Symposium on Network Computing and Applications (NCA). Cambridge, MA, USA, pp. 41–48.

Le Merrer, E., Kermarrec, A.-M., Massoulié , L., 2006. Peer to peer size estimation in large and dynamic networks: a comparative study. In: Proceedings of the 15th IEEE International Symposium on High Performance Distributed Computing, Paris, France.

Malkhi, D., Horowitz, K., 2003. Estimating network size from local information. ACM Information Processing Letters 88 (5), 237–243.

Malkhi, D., Naor, M., Ratajcjak, D., 2002. Viceory: a scalable and dynamic emulation of the butterfly. In: Proceedings of the 21st Annual ACM Symposium on Principles of Distributed Computing (PODC), pp. 183–192.

Manku, G., 2004. Balanced binary trees for ID management and load balance in distributed hash tables. In: Proceedings of the 23rd Annual ACM Symposium on Principles of Distributed Computing (PODC), Newfoundland, Canada, pp. 197–205.

Manku, G., Bawa, M., Raghavan, P., 2003. Symphony:distributed hashing in a small-world. In: Proceedings of the 4th USENIX Symposium on Internet Technologies and Systems (USITS), pp. 127–140.

Massoulié, L., Le Merrer, E., Kermarrec, A.-M., Ganesh, A.J., 2006. Peer counting and sampling in overlay networks: random walk methods. In: Proceedings of the 25th Annual ACM Symposium on Principles of Distributed Computing (PODC), Denver, Colorado.

Nath, S., Gibbons, P.B., Seshan, S., Anderson, Z.R., 2004. Synopsis diffusion for robust aggregation in sensor networks. In: Proceedings of ACM SenSys, pp. 250–262.

PlanetLab, 2006. An open platform for developing, deploying and accessing planetary-scale services. www.planet-lab.org.

van Renesse, R., Minsky, Y., Hayden, M., 1998. A gossip-style failure detection service. In: Proceedings of IFIP/ACM Middleware, Lancaster, England, 1998.

van Renesse, R., Birman, K.P., Vogels, W., 2003. Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining. ACM Transactions on Computer Systems 21 (2), 164–206.

Rowstron, A., Druschel, P., 2001. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer Systems. In: Proceedings of IFIP/ACM Middleware, Heidelberg, Germany, 2001, pp. 329–350.

Stoica, I., Morris, R., Karger, D., Kaashoek, F., Balakrishnan, H., 2001. Chord: A scalable peer-to-peer lookup service for Internet applications. In: Proceedings of ACM SIGCOMM, San Diego, CA, USA.

Stutzbach, D., Rejaie, R., 2005. Characterizing unstructured overlay topologies in modern P2P file-sharing systems. In: Proceedings of the Internet Measurement Conference (IMC), pp. 49–62.

Dionysios Kostoulas completed his Master of Science from the Department of Computer Science in the University of Illinois at Urbana-Champaign in 2005. He was a recipient of the Graduate Student Fellowship from Lilian Voudouris Foundation (2004–2005), a Fulbright Scholar (2003–2004), a Triantafyllou Scholar (2003–2004), and received a Graduate Student Scholarship from the Gerondelis Foundation (2003–2004).

Dimitrios Psaltoulis completed his Master of Science from the Department of Computer Science in the University of Illinois at Urbana-Champaign in 2005. He was a Fulbright Scholar (2003–2004), a Triantafyllou Scholar

(2003–2004), and received a Graduate Student Scholarship from the Gerondelis Foundation (2003–2004).

Indranil Gupta is Assistant Professor of Computer Science in the University of Illinois at Urbana-Champaign. He completed his Ph.D. from Cornell University in 2004, and received the NSF CAREER award in 2005. He heads the Distributed Protocols Research Group (DPRG) at UIUC, which studies various distributed systems such as peer-to-peer systems, the Grid, sensor networks, and design methodologies that span the breadth of these areas. He has served on the Program Committees for several ACM and IEEE conferences.

Kenneth P. Birman is Professor of Computer Science at Cornell University where he has studied distributed systems security and reliability issues since 1981, before which he completed his Ph.D. from the University of California, Berkeley. He is probably best known for having developed the Isis Toolkit, a fault-tolerance technology that became the centerpiece of the communications systems in the New York and Swiss Stock Exchanges, the French Air Traffic Control System, the Naval AEGIS warship, Florida Gas and Electric's control system, and many other mission-critical applications. Isis was also the basis of a company that he founded in 1987 and sold to Stratus Computer in 1993. He is the author of a book called "Reliable Distributed Systems" (Springer-Verlag; 2004), and has written several other books and many articles on the subject, including one that appeared in Scientific American in May, 1996. He was Editor in Chief of ACM Transactions on Computer Systems from 1993–1998 and is a Fellow of the ACM.

Alan J. Demers is Professor of Computer Science at Cornell University, where his research deals with weakly-consistent data replication in databases and distributed systems. He completed his Ph.D. at Princeton University in 1975, and was subsequently involved with the Clearinghouse and Bayou projects at Xerox PARC.