Cover Page

Title: Practical algorithms for Size estimation in Large and Dynamic groups.

Authors: *Dimitrios Psaltoulis, Dionysios Kostoulas, Indranil Gupta,*

(Dept. of Computer Science, University of Illinois, Urbana-Champaign)

Ken Birman, Al Demers

(Dept. of Computer Science, Cornell University)

Contact Person: Indranil Gupta

Phone: (217) 265-5517

Fax: (217) 265-6494

Email: indy@cs.uiuc.edu

Mailing Address: Siebel Center for Computer Science,

201 N. Goodwin Avenue, University of Illinois, Urbana, IL 61801, USA

Abstract:

Large-scale distributed systems may be required to estimate the number of non-faulty processes present in the group at a given point of time. The problem is related to aggregation, and several solutions have been proposed. In this paper, we present two new sampling-based algorithms for estimation in dynamic groups (i.e., where processes are constantly joining and crashing), and thoroughly evaluate them using real-life traces. One scheme spreads a gossip into the overlay first, and then samples the receipt times of this gossip at different processes. The second scheme measures the density of processes when their identifiers are hashed into a real interval. The schemes have low latency, per-process overheads, while providing high levels of probabilistic accuracy. We present simulation studies that measure and compare the performance of these approaches in static groups, and groups with significant turnaround (arrival and departure) of processes. The latter are done by using traces from deployed peer to peer overlays. The schemes are generic enough to be used by any distributed application.

Student Paper : Yes (Dimitrios Psaltoulis and Dionysios Kostoulas are students)

Keywords: Distributed systems, Scalability, Group size, Probabilistic estimation, Gossip, Sampling.

Number of pages: 10

Practical Algorithms for Size Estimation in Large and Dynamic Groups

Dimitrios Psaltoulis, Dionysios Kostoulas, Indranil Gupta,

(Dept. of Computer Science, University of Illinois, Urbana-Champaign)

Ken Birman, Al Demers

(Dept. of Computer Science, Cornell University)

{kostoula,psaltoul,indy}@cs.uiuc.edu; {ken,ademers}@cs.cornell.edu

Abstract

Large-scale distributed systems may be required to estimate the number of non-faulty processes present in the group at a given point of time. The problem is related to aggregation, and several solutions have been proposed. In this paper, we present two new sampling-based algorithms for estimation in dynamic groups (i.e., where processes are constantly joining and crashing), and thoroughly evaluate them using real-life traces. One scheme spreads a gossip into the overlay first, and then samples the receipt times of this gossip at different processes. The second scheme measures the density of processes when their identifiers are hashed into a real interval. The schemes have low latency, per-process overheads, while providing high levels of probabilistic accuracy. We present simulation studies that measure and compare the performance of these approaches in static groups, and groups with significant turnaround (arrival and departure) of processes. The latter are done by using traces from deployed peer to peer overlays. The schemes are generic enough to be used by any distributed application.

1. Introduction

Distributed systems such as peer to peer overlays, sensor networks, the Grid, etc., tend to be large-scale (several thousands of processes), but more importantly they are also *dynamic*. This means that new processes are joining, processes are dropping out of the group either through a crash-stop failure, or voluntarily, and these changes are occurring all the time. 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 the problem of *Group Size Estimation*. The value so calculated can be used to set the routing table sizes in overlays such as Pastry [11], to decide subgroup sizes in the Kelips overlay [7], to set timeouts for query replies, to collect statistics, etc.

The problem has two flavors - one-shot and continuous. We formally define the one-shot problem as follows; the continuous version is similar.

Group Size Estimation Problem: Give a protocol that when initiated by one process, estimates the number of non-faulty processes in the overlay graph component containing the initiating process.

Clearly this problem is impossible to solve accurately in a dynamic group. Notice that a group size estimation protocol will take non-zero time to run, and for any other process p than the initiator, there will be a time interval after the last message for the estimation protocol seen by p, and before the initiator finalizes the estimate. In a run where process p fails during this interval, the estimate will be incorrect.

Nevertheless, it is easy to design algorithms for approximate estimation. A simple algorithm is as follows. The initiator process can send a multicast to the overlay through a dynamic spanning tree, and wait to receive back replies. Clearly this scheme creates a large number of messages at the initiator, and can be non-fault-tolerant when a process fails after receiving the multicast, but before replying to its parent in the spanning tree.

Contributions of the Paper: This paper proposes practical, efficient, and fault-tolerant estimation algorithms with a probabilistic output. Specifically, we study two algorithms, both based on variants of sampling. The first algorithm, called Hops Sampling, initiates a gossip into the group, and samples the times at which it is received by different processes in order to estimate the value of the logarithm of group size. The second approach, called

Interval Density, samples the density of processes that lie in a given real interval when their identifiers are hashed.

Both our algorithms can be run either in a one-shot manner, or on a continuous basis (the latter is preferable for long-running applications such as peer to peer overlays). When used in a one-shot manner, they 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 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 group size.

We evaluate the two algorithms using simulations of peer to peer overlays. Experiments include simple micro-benchmarks as well as those based on availability traces from a deployed peer to peer system.

Related Work: A ring-based algorithm for group size estimation is given in [10], which requires a logical ring to be maintained among existing processes in the network; however, the estimated group size is between $N/2$ and N^2, where N is the actual group size Ref. [2] presents several mechanisms for aggregation an group size estimation. The technique that bears similarity to ours is the random walk approach, where an estimation message is sent on a random walk, and the hop count is returned when a process is encountered a second time. According to the birthday paradox, it takes $\Theta(\sqrt{N})$ hops for the message to encounter a process a second time. However, the accuracy of the algorithm requires each process to maintain a membership list that is uniformly sampled from across the group. This is known to be difficult to achieve. Ref. [2] implements each "step" using a random walk, however, this does not completely eliminate dependence of accuracy on the distribution of membership selection. Our Hops Sampling algorithm also suffers from the same disadvantage, but the proposed Interval Density approach does not. Kermarrec et al discuss practical estimation schemes in [9], but at the time of writing this paper, they have not been evaluated. Due to space limitations, we defer experimental comparison of our approaches with those in [4,10].

Epidemic algorithms were discussed by Demers et al in [5]. Most of the gossip algorithms in our paper are push-based. In [8], a push-pull hybrid gossip algorithm is presented, that takes *O(N log log N)* transmissions and *O(log N)* time. Birman et al [4] use gossip to design a probabilistically reliable multicast protocol.

System Model: We assume a group of processes with unique identifiers, communicating over an asynchronous network. The number of non-faulty processes is N, and is an unknown quantity. Since the duration of a round is *O(seconds)*, processes can be assumed to have negligible clock drifts. Processes can undergo crash-stop failures; crash-recovery failures can be supported by having a process rejoin the group with an identifier that is different and unique from other processes.

The rest of this paper is organized as follows. Section 2 presents and analyzes the Hops Sampling Algorithm. Section 3 presents the Interval Density Approach. Section 4 demonstrates experimental results. Finally, Section 5 summarizes our contributions and presents items for future work.

2. The Hops Sampling Algorithm

Figure 1 shows pseudocode for a canonical Hops Sampling Algorithm. The main idea is as follows. A one-shot estimation run is initiated by a single process. Once a process receives a message for the protocol run (the initiator sends a message to itself), it actively gossips to gossipTo processes during each *protocol period*, also called a *round* (durations fixed and rounds start at same time at all processes). Gossip targets at a process p are chosen from only among those that are not known to have previously sent a gossip message to process p during this run (see variable called fromlist). Each process gossips either for gossipFor periods, or until gossipUntil messages have been received. Gossip messages carry a hop count, which specifies the number of processes the message has passed through, after the initiator (maintained using hopnumber in the message, and myhopcount at each process).

The initiator waits for gossipResult rounds to elapse before collecting a sample of hop counts from at most gossipSample other processes, and outputting the average hop count as an estimate of *log(N)*, where the logarithm's base depends on the parameter settings.

Each process is required to maintain a membership list, a partial list of currently non-faulty processes in the group. Gossip-based protocols require partial lists that are only logarithmic in the total group size [6]. The

fraction of list entries that are failed processes affects the base of the estimation algorithm as well. It is difficult to analyze failures effect mathematically, hence our analysis assumes that gossip targets that are unresponsive will be retried, and thus all gossip attempts succeed.

```
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, fromlist);
      Group Membership List: G;

      fromlist=NULL;                    /* list of processes from which I have received gossip */
      At the Initiator 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, fromlist) 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};
                  numrecvdgossips++;
                  if (numrecvdgossips > gossipUntil)
                        gossip_on=FALSE;
            gossipResult protocol periods after starting::
                  sample a random subset of gossipSample processes for their myhopvalues;
                  output the mean of received non-zero values

      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, fromlist) 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;
                  numrecvdgossips++;
                  if (numrecvdgossips > gossipUntil)
                        gossip_on=FALSE;
```

Figure 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.*

Analysis: Although some explicit equations can be derived for the number of gossip recipients at time *t* [1], it is difficult to calculate the exact time distribution of average hop count. Hence, we present a qualitative argument that the average hop count is indeed *O(*log(*N*)*),* and defer details (e.g., the base of the logarithm) to the experiments in Section 4.

Consider a group of non-faulty processes, and a reliable communication medium. It is shown in [1] that the number of rounds for a gossip to reach 50% of the processes grows as $\Theta(\log(N))$. From this, it is clear that the average hop count has to grow at least as fast as $\Theta(\log(N))$.

Suppose each process that receives a gossip message sends gossips to a *total* of *b* other processes (so that if a process gossips for gossipFor rounds, *b* = gossipTo X gossipFor), each selected at random from the group. If *r* is the fraction of processes that do not receive a gossip, we can write $r = (1 - b/N)^{-N.(1-r)}$. For large *N*, this becomes $r = e^{-b(1-r)}$. Now, if the value of b is high enough, we can assume r to be small, and it can be ignored in the exponent, giving us $r = e^{-b}$. This is *o(1)* when gossipFor equals log(*N*), justifying our assumption. Thus, most of the final samples by the initiator will return useful hop counts. Further, [1] also shows the time required to reach all processes w.h.p. varies logarithmically with group size.

The average hop count is sandwiched between the time to reach 50% of the group, and to reach all but *o(1)* of the processes. Hence from the above discussion, we conclude that the average hop count varies as $\Theta(\log(N))$.

Continuous Version of the Protocol: The initiator periodically initiates a new one-shot protocol run, each with a unique run identifier. The network traffic is bounded since each run terminates w.h.p. in a logarithmic number of rounds. 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.

Drawbacks: In practice, an accurate group size estimation by the Hops Sampling algorithm requires a membership list that is chosen uniformly at random from among the non-faulty processes in the system. Since such a membership protocol is difficult to design (we believe there is none yet), an estimation algorithm whose correctness does not depend on the uniformity of membership lists is required. The next described approach, called the Interval Density approach, only requires the membership graph to be connected in order for a correct run of the protocol.

3. The Interval Density Approach

The Interval density approach attempts to measure 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 may not provide an accurate estimate. For example, if process identifiers consist of a concatenation of the IP address at the computer host the process is running, and the port number at which the process is communicating, sampling an interval of the 40 bit process identifier space (IPaddress+port number) might give a biased estimate if the process group were mostly running on a few subnets.

The Basic Algorithm: The second approach is to randomize the process identifiers by using a uniform hash function to map each process' identifier to a point in the real interval [0,1]. Cryptographic hash functions such as SHA-1 [14] (or MD-5) can be used: these take as input arbitrary length binary strings, and output a hash of length 160 bits (or 128 bits respectively for MD-5). The hashes can be normalized by dividing with $2^{160} - 1$ (or $2^{128} - 1$ respectively). This is convenient to do in peer to peer routing substrates such as Pastry [11] and Chord [12], where virtual "nodeID"s are assigned to processes by hashing their identifiers using SHA-1 or MD-5 anyway.

Consider that the initiator is able to calculate the number of processes in an interval of size $I < 1$ that is a subset of the interval [0,1]. Suppose *X* is the actual number of processes that the initiator finds to fall in the interval. Then the estimate for group size would be simply *X/I*.

Analysis: Let *N* be the actual number of non-faulty processes in the group, and assume a uniform hash function. Then the expected number of processes that fall in the interval would be *I.N*. Call δ as the *accuracy* of the protocol, defined as an interval around the mean where an estimate is likely to be, with high probability. The likelihood that the estimate for group size is off by a factor of at most 2δ from the mean is:

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

If $\delta < 2e-1$, the latter terms can be calculated using Chernoff bounds as:

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

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 at least *O*(log(*N*)/*N*). If *I*>*c*.log(*N*)/*N,* then the probability that the estimate is accurate would be

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

This is very close to 1.0. Better accuracy can be obtained by using larger interval sizes. 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 is close to 1 if $\delta = \sqrt{\log(N)} / N^{1/4}$, which gives a better accuracy than 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 *N* is known, the interval can be chosen to be large enough to guarantee a required level of reliability. However, since a very large number of processes fall inside this interval, this would result in high overhead, i.e., either the memory usage at the initiator, or the message overhead, will increase linearly with the system size. Strategies for adaptively setting the interval size can be used, and these are described in subsequent sections. The adaptive strategies also adjust to a bad choices for location of the interval.

The leftover issues from the above discussion are (a) adaptive approaches to decide the interval size and the interval center; and (b) protocols for collecting the samples. (a) and (b) are orthogonal and are hence discussed separately next ((a) in Sections 3.1, 3.2, and (b) in Section 3.3). The remaining discussion in this section is for the continuous flavor of the Interval Density approach.

3.1. Adapting the Size of the Interval

At the initiator, after a center point is chosen, processes hashing into an interval around the center point are remembered. In practice, the size of this interval is not fixed *a priori*. It is determined by remembering a *number* of processes around 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.

3.2. Adapting Interval Location, and Using Multiple Estimation Runs for better Accuracy

Choosing a good center point for the interval can affect the accuracy of the estimate, especially if process identifiers hash in a rather non-uniform manner into the interval [0,1]. We describe three approaches for choosing a good interval center, and using multiple estimations to obtain a better estimate.

1. *Random selection:* This is the simple approach where the initiator randomly selects the center point interval from [0,1], and uses it 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 independent of, or dependent on, the previous intervals. An example of the former approach would be the following: each change involves moving the center point up by a constant pre-fixed amount.

A better approach is the following. At every stage, histories of the results of a few recent estimation runs are maintained (say the recent 8 runs), along with the intervals that were used for them. The center point is then selected as follows. The average estimate returned by these recent runs is calculated, and the center point of the run that returned an estimate closest to this average, is used.

Thus, the center of the interval would continuously move right and left in the nodeId space after each round depending on the results of estimations made on the last rounds. This algorithm will work well if the group size does not change dramatically between two successive estimation runs. We call the latter method as *self-adjusting interval selection*.

3. *Multiple selections and estimation by mean value.* Instead of changing the center of the interval over time, the initiator each estimation run includes multiple runs, each with the interval centered at a different point within the nodeId space. The mean value of the multiple results is then returned as the estimate.

Our experiments in Section 4 compare the relative performance of the three approaches.

3.3. Piggybacking on a Membership Maintenance Protocol

We are yet to describe how to collect information about processes lying inside the selected interval. One possibility is the following: a multicast by the initiator is followed by responses sent back by all processes that hash to the specified interval. Clearly, this is inefficient since it can result in message implosion at the initiator.

Alternatively, we observe that all distributed systems use membership maintenance protocol, wherein processes periodically multicast their heartbeats (incremented sequence numbers) to all other processes. When an updated heartbeat counter has not been received for a remote process q that is in the membership list at p, q is deleted from p's membership list. One of the many ways in which such heartbeats are multicasted is through gossiping. Gossiping is advantageous since gossip messages can carry multiple heartbeats, for several other processes [13].

Each process p periodically (a) increments its own heartbeat counter, and (b) selects some of its neighbors (defined by the membership list at p), and sends them a list of heartbeats and identifiers for all processes that p knows lie in the interval. The interval is also included in the message in case the target process has not yet received an estimation message. Upon receipt of such a gossip message, a member merges the list in the message with its own list, and adopts the maximum heartbeat counter for each member. Each previously unknown process identifier is hashed, and if it lies in the interval, is included in future gossip messages.

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, the time complexity is $O(I.N.logN)$.

Similar to [13], membership entries time out if updated heartbeats have not been received for T_{down} time units are marked for deletion, and are deleted after another T_{down} time units (if they are not remembered in such a manner, the failed member may be re-included in the membership list).

Clearly, if processes never stop gossiping, the above protocol can be used for a continuously running estimation.

Multiple Initiators: Multiple initiators can be supported by selecting the interval specified by the initiator with the lowest identifier. When a process p running an estimation for one initiator hears about another initiator with a lower identifier, process p instead starts participating in the latter initiator's run.

4. Experimental Results

We evaluate the performance of the Interval Density and the Hops Sampling approaches under two scenarios: (i) Microbenchmarks, with a static group with a fixed size, and (ii) Trace-based experiments, using trace-log data from Overnet file-sharing network [3] to simulate a dynamic and open group. Our discussion replaces the previous notation "process" with "node", since the algorithms are also applicable to groups of hosts (not just processes).

Overnet network data measures 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 present in the system changes by as much as 10%-25% every hour. We first present individual studies for each of the Hops Sampling algorithm and Interval Density approach, and then compare them against each other.

4.1. Hops Sampling Approach

We assume the membership list at each node to be sampled uniformly at random from the entire set of nodes; lists may contain faulty nodes.

Microbenchmarks: With parameter settings of gossipTo=2 and gossipFor=1, Figure 2 plots the average number of hops measured in a static network with N nodes versus log_2N, for different group sizes. The straight line shown in the figure is given by log_2N=0.9895*averageNumberOfHops+1, and appears to hold for group sizes up to 30,000. From this, the estimated group size is thus calculated as $SizeEst = 2^{(0.9895*avHops+1.3)}$.

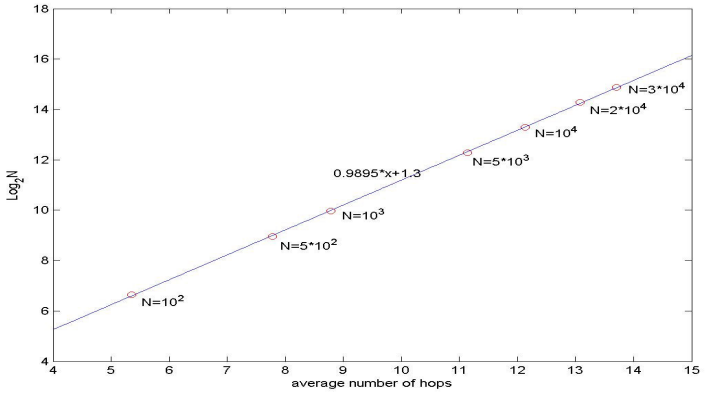
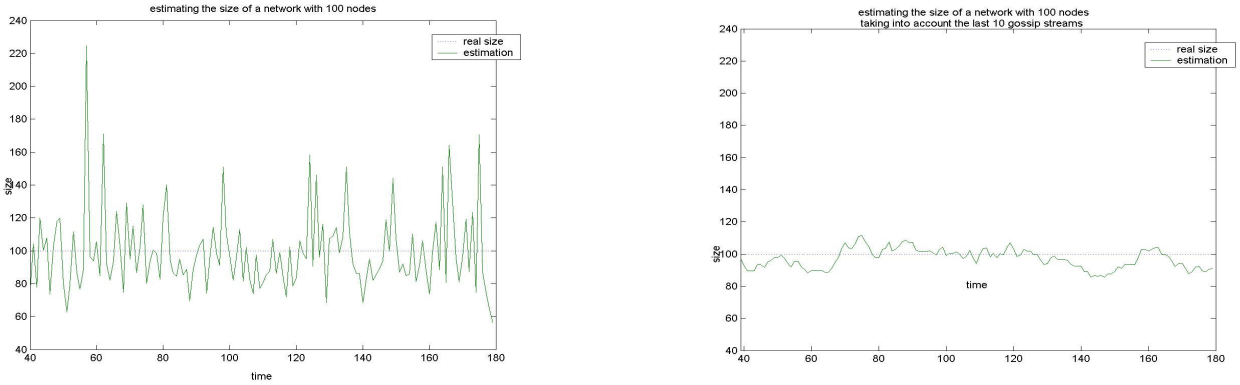


Figure 2. *Average number of hops versus Log_2N for networks with size of 100-30000 nodes.*



(a) (b)

Figure 3. *Estimation of the size of a network with 10^2 nodes.*

Figure 3(a) shows the variation, over time, of the estimated group size, for a static group of size 100. Figure 3(b) shows the effect of averaging over several estimation runs; the continuous protocol version parameters gossipsAccounted=10, gossipsDropped=2 are used. The benefit of the latter is clear from the figures; the estimate in Figure 3(b) is mostly within 10% of the actual group size.

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 nodes of the system 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. Figure 4 shows the variation, over time, of the estimated group size. A close look reveals that although the estimate is off by a constant factor (due to a smaller exponent value), the estimate appears to follow and mirrors the variation, over time, of the actual system size.

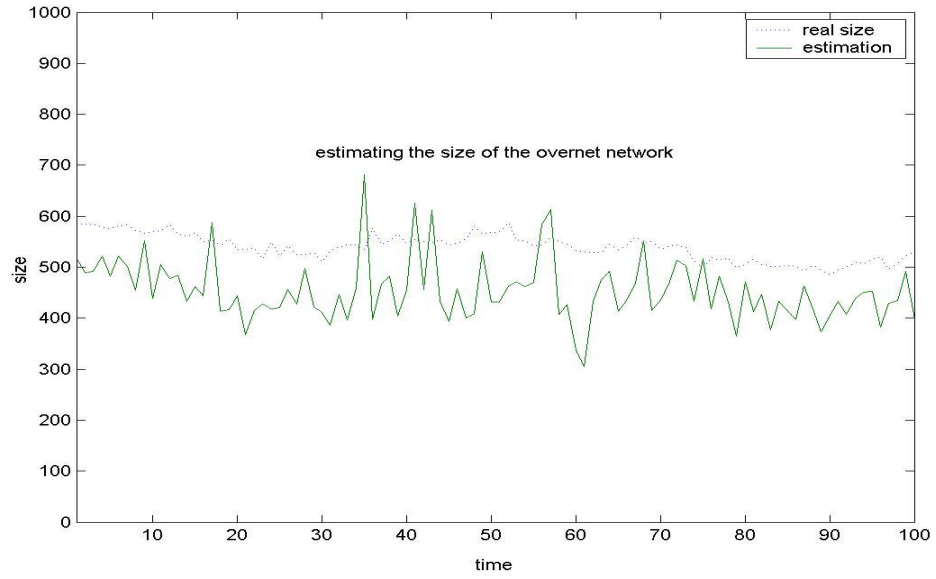


Figure 4. *Estimation of the size of the Overnet network.*

4.2. Interval Density Approach

In Section 4.2.1 we study the accuracy of the approach for fixed group sizes. In Section 4.2.2 we study the adaptive mechanisms. Interval Density messages are piggybacked on a gossip based heartbeating protocol, as described in Section 3.3, with $T_{down} = 10$ rounds. MAXMEMORY is set to 60.

4.2.1. Microbenchmark: Static Sized Groups

Figure 5 shows the variation, over time, of the estimate in a static groups with 10,000 nodes. The random selection approach is used to decide the center of the estimation. The plot shows that the estimate does not converge. This is due to the early expiry of some membership entries in the gossip-based heartbeat protocol. Figure 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. Figures 5-6 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 large or infinite values of T_{down}, the estimate can be offset by the local density of hashed process identifiers in the chosen interval. The random approach has a 5% accuracy; for better accuracy, the "multiple selection estimation by mean value" approach should be used.

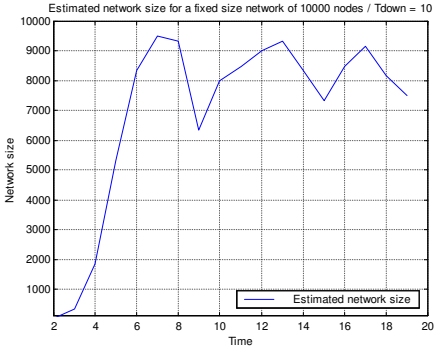


Figure 5. *Estimated network size for fixed size networks of 10,000 nodes (random selection, MAXMEMORY = 60, T_{down} = 10).*

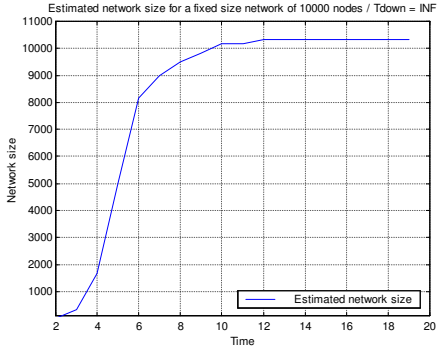


Figure 6. *Estimated network size for fixed size networks of 10,000 nodes (random selection, MAXMEMORY = 60, T_{down} = infinity).*

4.2.2. Trace-base Simulations: Effect of Adaptive Mechanisms

Varying the center point of the interval: Figures 7-10 show the behavior of four different center point selection strategies for the Interval Density approach. The "self-adjusting interval selection" (Figure 9) and the "multiple selection estimation by mean value" (Figure 10) appear to perform the best, although the other estimates all lie within 20% of the actual group size. As expected, "multiple selection estimation by mean value" has the least variance among since multiple intervals are sampled, and an average is taken over them.

Highly Dynamic Groups: Although the fraction of nodes joining and leaving the system is as high as 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 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 between time intervals. The self-adjusting method is used for providing estimations. The results are presented in Figure 11, and show good behavior in spite of highly dynamic groups. A look at Figure 11 also shows that the variation over time of the estimate closely follows and mirrors the variation of the actual group size.

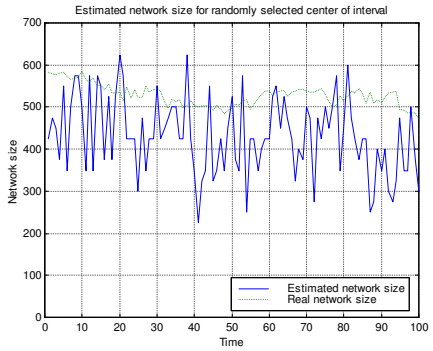


Figure 7. *Estimate calculated by the random selections of the center of interval approach.*

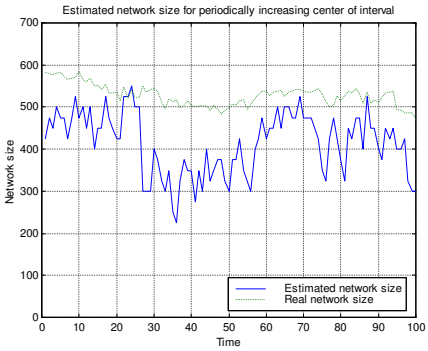


Figure 8. *Estimate calculated by the periodically increasing center of interval approach.*

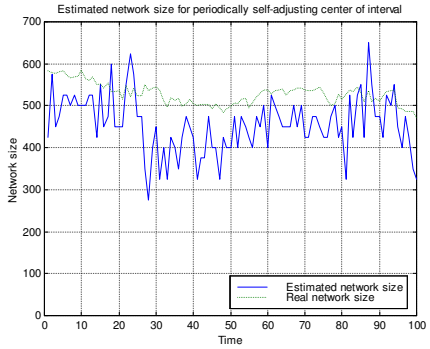


Figure 9. *Estimate calculated by the periodically self-adjusting center of interval approach.*

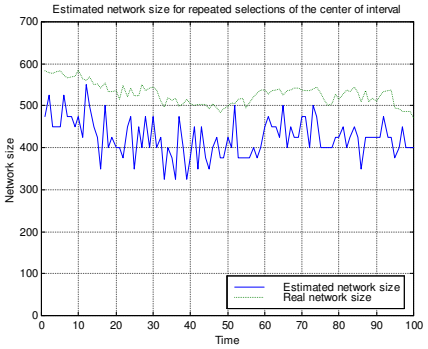


Figure 10. *Estimate calculated by the estimation by mean value approach.*

4.3. Hops Sampling Algorithm versus Interval Density Algorithm

Figures 12 and 13 compare the approaches in static (fixed size) and dynamic (Overnet-trace based). Parameter settings are gossipTo = 2, gossipFor = 1, gossipsAccounted = 10 and gossipsDropped = 2. MAXMEMORY = 60 and T_{down} = infinity, are used for Figure 12. The periodically self-adjusting interval method with MAXMEMORY = 120 and T_{down} = 10 is used for Figure 13.

For static group sizes, both the Interval Density and Hops Sampling approaches converge quickly to within 5% of the actual group size (Figure 12), with the Hops Sampling approach performing slight better. For dynamic behavior due to Overnet traces, the estimate from the Interval Density approach is slightly better than that of the Hops Sampling approach (Figure 13). However, the Interval Density approach has the advantage that it does not require the membership list to contain nodes that are selected uniformly at random.

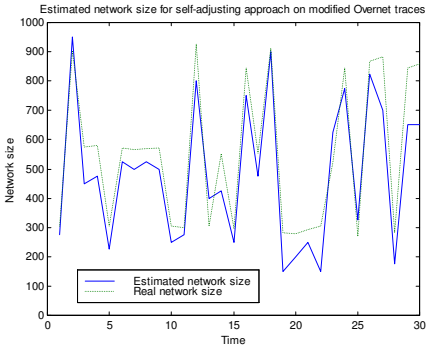


Figure 11. *Behavior of self-adjusting center selection in a dynamic group (Overnet traces, with additional random joins and failures).*

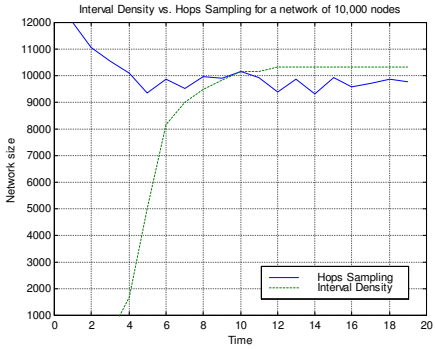


Figure 12. *Comparing the accuracy of the Interval Density and the Hops Sampling approaches for a large fixed size network of 10,000 nodes.*

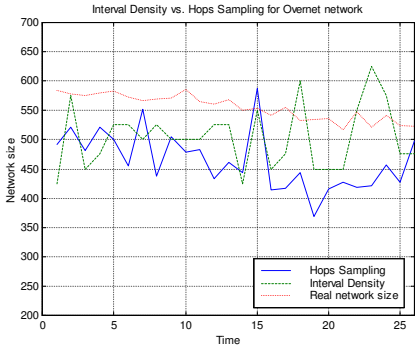


Figure 13. *Comparing the accuracy of the Interval Density and the Hops Sampling approaches for Overnet traces.*

5. 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. Previous solutions to the problem make assumptions that may be impractical. In this paper, we have proposed two new approaches, called Hops Sampling and Interval Density, for group size estimation problem in a large-scale and dynamic group. Microbenchmark (for static groups) and trace-based simulations show that both approaches are able to obtain estimate within a few percentage points of the actual group size. The Interval Density approach is highly practical, since its accuracy only requires the overlay graph among all processes in the system to be connected (when membership timeouts are very large). Both approaches are generic enough to be used within any distributed system.

References

[1] N. T. J. Bailey, *"Epidemic Theory of Infectious Diseases and its Applications"*, Hafner Press, Second Edition, 1975.

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

[3] R. Bhagwan, Overnet availability traces: http://ramp.ucsd.edu/projects/recall/download.html

[4] K.P. Birman, M. Hayden, O. Ozkasap, Z. Xiao, M. Budiu and Y. Minsky, "*Bimodal Multicast*", ACM Transactions on Computer Systems, vol. 17, no. 2, pp. 41-88, May 1999.

[5] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart and D. Terry, "*Epidemic algorithms for replicated database maintenance*", In Proc. Symposium on Principles of Distributed Computing, pages 1-12, Aug. 1987.

[6] P. Th. Eugster, R. Guerraoui, S. B. Handurukande, A.M. Kermarrec, and P. Kouznetsov, *"Lightweight probabilistic broadcast"*, ACM Transactions on Computer Systems, vol. 21, no. 4, pp. 341-374, Nov. 2003.

[7] I. Gupta, K. Birman, P. Linga, A. Demers, and R. van Renesse, "*Kelips: building an efficient and stable P2P DHT through increased memory and background overhead",* In Proc. Second International Workshop on Peer-to-peer Systems, Springer LNCS 2735, pp. 160-169, 2003.

[8] R. M. Karp, C. Schindelhauer, S. Shenker and B. Vocking, "*Randomized rumor spreading",* IEEE Symposium on Foundations of Computer Science, pp. 565-574, 2000.

[9] A.-M. Kermarrec, L. Massoulie, Private communication, 2003.

[10] D. Malkhi, K. Horowitz, "*Estimating network size from local information"*, ACM Information Processing Letters, vol. 88, issue 5, pp. 237-243, Dec. 2003.

[11] A. Rowstron and P. Druschel, "*Pastry: scalable, distributed object location and routing for large-scale peer-to-peer Systems*", Proc. of IFIP /ACM Middleware, pp. 329-350, Nov. 2001.

[12] I. Stoica, R. Morris, D. Karger, F. Kaashoek and H. Balakrishnan, "*Chord: A scalable peer-to-peer lookup service for Internet applications"*, In Proc. ACM SIGCOMM Conference, Aug. 2001.

[13] R. van Renesse, Y. Minsky, and M. Hayden, "*A gossip-style failure detection service*", in Proc. Middleware '98, Lancaster, England, Sep. 1998.

[14] FIPS 180-1, "*Secure Hash Standard*", NIST, US Department of Commerce, Washington D.C., Apr. 1995.