Decentralized Schemes for Size Estimation in Large and Dynamic Groups*

Dionysios Kostoulas, Dimitrios Psaltoulis, Indranil Gupta
Dept. of Computer Science
University of Illinois, Urbana-Champaign
{kostoula, psaltoul, indy}@cs.uiuc.edu

Ken Birman, Al Demers
Dept. of Computer Science
Cornell University
{ken, ademers}@cs.cornell.edu

Abstract

Large-scale and dynamically changing distributed systems such as the Grid, peer-to-peer overlays, etc., need to collect several kinds of global statistics in a decentralized manner. In this paper, we tackle a specific statistic collection problem called Group Size Estimation, for estimating the number of non-faulty processes present in the global group at any given point of time. We present two new decentralized algorithms for estimation in dynamic groups, analyze the algorithms, and experimentally evaluate them using real-life traces. One scheme is active: it spreads a gossip into the overlay first, and then samples the receipt times of this gossip at different processes. The second scheme is passive: it measures the density of processes when their identifiers are hashed into a real interval. Both schemes have low latency, scalable per-process overheads, and provide high levels of probabilistic accuracy for the estimate. They are implemented as part of a size estimation utility called PeerCounter that can be incorporated modularly into standard peer-to-peer overlays. We present experimental results from both the simulations and PeerCounter, running on a cluster of 33 Linux servers.

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.

Accurate estimates of the number of processes currently present in the overlay, i.e., the group size, are absolutely essential in order to enable several notions of run-time adaptivity in peer-to-peer applications. For example, in a Grid application, continuous estimates of the group size can be used to dynamically partition a distributed problem among participating clients. In peer-to-peer distributed hash tables such as Pastry [16] and Chord [17], the estimated value is required to set the routing table size and to set timeouts for queries – the number of virtual hops for routing a query depends on the group size. Other overlays, e.g., Kelips [9], use the notion of process subgroups whose size depends on the number of processes present in the system.

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., [3]. 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: *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.*

Impossibility of Group Size Estimation in a Dynamic Group: 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 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 work was supported in part by NSF CAREER grant CNS-0448246.

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 receiving the multicast and before replying to its parent in the spanning tree, the estimate will exclude all descendants of that process in the tree.

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. The first algorithm is active: 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. The second approach is passive: called the *Interval Density Approach*, it samples the density of processes that lie in a given real interval, when the process identifiers are hashed.

Both 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, 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 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 our simulations to be run on a cluster of up to 33 Linux servers, each with multiple processes. PeerCounter is open-source and can be easily incorporated into a variety of peer-to-peer substrates, e.g., Pastry, Chord, Kelips [9, 16, 17], etc., thus providing them with the capability to estimate the number of non-faulty processes present in the group. More details and evaluation results for our algorithms may be found in an extended technical report version of the paper [12].

System Model: We assume a group of processes 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. Membership entries are assumed to be maintained by a complementary membership protocol already running within the overlay. The only requirement from the membership component is that the overlay graph formed by it be connected. This admits overlays constructed by distributed hash tables (e.g. Pastry, Chord, etc.), unstructured overlays (e.g. Gnutella), as well as overlays in Grid applications. Moreover, 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. 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. 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 summarize in Section 7.

2. Related Work

Several efforts have been made to provide decentralized solutions to the group size estimation problem. Unfortunately, most of these approaches either do not apply to dynamic groups, or make unreasonable assumptions. Ref. [3] presents 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 impractical in groups that have processes joining and leaving at high rates. Our schemes have a latency that increases only logarithmically with N.

Our Interval Density algorithm bears resemblance to the approach in [1], but there are significant differences. Ref. [1] uses a hash function to assign process IDs in the real interval [0,1] and decomposes the [0,1] interval into a hierarchy of subintervals, each holding identifiers from processes that run on the same region. A recursive algorithm is used, starting from the peer's home region

and continuing upwards until an estimate for the [0,1] interval is achieved. In comparison, our Interval Density approach does not depend on the way process identifiers are assigned in the group. Further, the algorithm of [1] could suffer from message implosion at the initiator process, while the Interval Density algorithm imposes no extra message complexity since it is a passive approach.

Kermarrec et al also discuss practical estimation schemes in [13], but at the time of writing this paper, they have not been evaluated. Ref. [10] presents a sampling-based size estimation scheme. Nevertheless, the accuracy of the method depends on assumptions on the independence of samples from different processes. The algorithms proposed in this paper do not assume independence of membership views at each process. In addition, a ring-based algorithm for group size estimation is discussed in [14]; however, the estimated group size can have a large error – it can be between $N/2$ and N^2, where N is the actual group size. The required logical ring may not be present in all overlays, e.g., Grid overlays. Our schemes in this paper do not require such a fixed minimal overlay, and can be run on a large range of overlays.

Epidemic algorithms were discussed by Demers et al in [6], Eugster et al in [7], Karp et al in [11], and originate from the Mathematical study of epidemics [2]. Birman et al [5] use gossip to design a probabilistically reliable multicast protocol.

3. Active Approach - Hops Sampling Algorithm

In this section, we present the Hops Sampling algorithm. Pseudocode of the algorithm can be found in [12]. To present the algorithm, we make the assumption that it 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. 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. Once a process has received the initiating message, at the beginning of each subsequent round, it selects `gossipTo` other processes as targets and sends them *gossip* messages containing the initiating message. 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. Besides the local membership list, each process p also maintains a list, `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 done *uniformly at random* from the membership list, but by excluding the elements that appear in `fromList`.

Since the membership protocol is weakly consistent, the membership list may be out of date. In order to maintain the `gossipTo` parameter, gossip messages need to be acknowledged, and a gossiping node retries each message with different target until acknowledged.

In order to enable estimation, each of the above gossip messages also carries an integer field `hopNumber`, which indicates the number of nodes the message has traversed since the initiator. Before forwarding a gossip, the process stores this hop count in a local variable called `myHopCount`. For multiple received gossip messages, the lowest received `hopNumber` value is remembered. All outgoing gossip messages have their `hopNumber` set to (`myHopCount+1`).

The initiating process 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 parameter settings.

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) with probability $(1/\texttt{gossipTo}^{(\texttt{myHopCount}-\texttt{minHopsReporting})})$ otherwise. Thus, only a small fraction of all hop count values will be received.

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 `gossipFor` to 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`.

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 sampled uniformly at random.

Property: 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 $\Theta(\log(N))$.

Explanation: At the core of the Hops Sampling algorithm is a push gossip protocol, the properties of which are well studied. Bailey [2] and Eugster et al [7] 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 O(log(*N*)), the gossip spreads to all processes with high probability; (ii) the expected number of rounds for the gossip spread to complete 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 $\Theta(\log(N))$.

We detail how to determine the value of the logarithm base in the average gossip latency when discussing experiments in Sections 5 and 6.

3.2. 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.

4. Passive Approach - Interval Density Approach

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 [8] (or MD-5) can be used: the input is arbitrary length binary strings, and the output is 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 P2P routing substrates such as Pastry [16] and Chord [17], where virtual "nodeIDs" are assigned to processes by hashing their identifiers using SHA-1 or MD-5.

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]. 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.

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.

Below, we first analyze the accuracy of the estimate, then describe adaptive variants of the protocol, and finally discuss the snooping on the membership protocol.

4.1. Analysis of the Interval Density Approach

The assumption on uniform sampling of membership lists used in analysis of the Hops Sampling scheme is not required in the case of the analysis of the Interval Density scheme.

Property: With a good (i.e., uniform) hash function, an interval size I that varies as $O(\log(N)/(\delta^2 \times N))$ suffices to obtain an estimate that is accurate within a factor of δ with high probability, where N be the actual number of non-faulty processes in the group.

Explanation: The expected number of processes that fall in the interval I would be $I * N$, where the size of the interval is notated simply as I. Call δ the *accuracy* of the protocol, defined as the multiplicative error in the group size. The likelihood that the estimate for the group size is off by a factor of at most 2δ from the mean is:

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

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

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

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 $O(\log(N)/(\delta^2 \times N))$.

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 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 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 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 choice for the location of the interval. They are described next.

The discussion in the remainder of this section applies to only the continuous flavor of the Interval Density approach.

4.2. Adapting the Size of the Interval

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

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]. Although process identifiers are initially randomized using a cryptographic hash function, this behavior may manifest over a long term due to process arrivals and departures. 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 of the run that returned an estimate closest to the average is used in next run.

   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.

In Section 5, we refer to results for the self–adjusting approach. More results, showing the relative performance of all approaches, can be found in [12].

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 heartbeat-style membership.

In a heartbeat-style membership protocol, e.g., [15], each process *p* periodically multicasts a heartbeat message (incremented sequence numbers) to *all* other processes in the group. These heartbeats are used to proactively learn about new prospective neighbors in the overlay, as well as for failure detection. The latter is achieved by timing out on the time since the last heartbeat was received by process *p* from a neighbor process *q* – this results in *p* deleting *q* from its membership list.

At the initiator, the received heartbeat messages can be used to continuously learn about previously unknown processes whose hashed identifiers lie within the interval *I* being used for estimation, as well as to drop known processes (lying within interval *I*) that have not responded with an updated heartbeat for a while. Hence, this keeps a running estimate of the number of process identifiers lying within *I*, and estimates the group size.

Heartbeat-style membership protocols also come in different flavors. We describe in detail how to incorporate the Interval Density approach into the popular gossip-style membership protocol detailed by van Renesse et al [15]. The basic gossip-style membership protocol has each process *p* periodically (a) increment its own heartbeat counter; (b) select some of its neighbors (defined by the membership list 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. 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*. 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, the time complexity is *O(I N logN)*.

5. Experimental Results

We 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 from Overnet file-sharing network [4] to simulate a dynamic and open group. Our discussion replaces the previous notation "process" with "node".

The traces from the Overnet network [4] measure the availability of nodes in a 3,000-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% - 25% every hour. We present individual studies for each of the Hops Sampling algorithm and Interval Density approach. A comparison of the two schemes can be found in [12].

Experiments do not use any assumptions for the membership protocol and are done under realistic conditions, as described in section 1.

5.1. Hops Sampling Approach

Micro-benchmarks. By plotting the average number of hops measured in a static network with *N* nodes versus $log_2 N$ for different group sizes [12], we find that the estimated group size is calculated as $SizeEst = 2^{(0.9895 * avHops + 1.3)}$. Here, we use as parameters `gossipTo`=2 and `gossipFor`=1, as explained in [12].

Figure 1 shows the continuous protocol version for a static group of size 100; the parameters used are `gossipsAccounted`=10, `gossipsDropped`=2. A comparison between the continuous and the one-shot version can be found in [12]. It is shown that the estimation is mostly within 10% of 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 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 same gossip parameters as earlier. Figure 2 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), it appears to follow and mirror, over time, the variation of the actual system size.

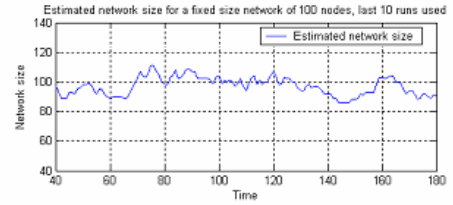

Figure 1. Size estimation for static group of 100 nodes.

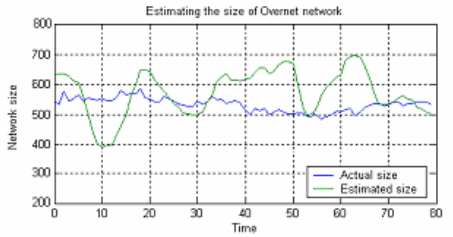

Figure 2. Size Estimation for Overnet network.

5.2. Interval Density Approach

Interval Density messages are piggybacked on a gossip-style membership protocol, as described earlier in this paper. Similar to [15], 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.

Micro-benchmark. Figure 3 shows the variation, over time, of the estimate in a static sized group with 10,000 nodes, when T_{down}=infinity and `MAXMEMORY`=60. In [12], we present 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 node identifiers in the chosen interval. The random approach has 5% accuracy; for better accuracy, the "multiple selections estimation by mean value" approach should be used.

Trace-based Simulations. Figure 4 shows the behavior, over Overnet traces, of self-adjusting method, for T_{down}=10 and `MAXMEMORY`=60. Ref. [12] demonstrates results for all adaptive methods, presented in section 4.3. Furthermore, in [12], we modify the Overnet traces so that an additional set of arrivals and departures is added, and provide results for estimations.

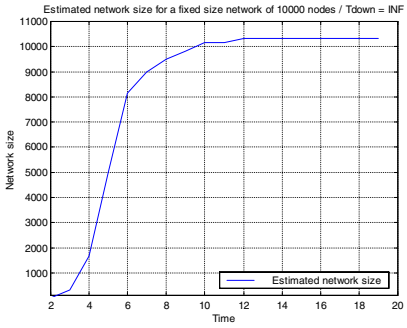

Figure 3. Size estimation for a static network of 10,000 nodes.

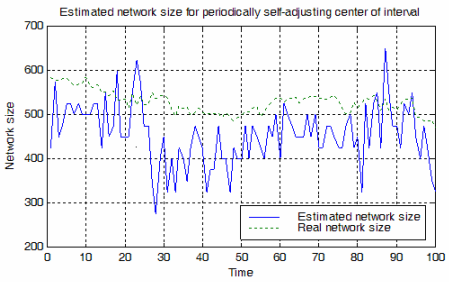

Figure 4. Estimate calculated by the self-adjusting interval selection approach for Overnet network.

6. PeerCounter: A Size Estimation Utility

In this section, we introduce PeerCounter; a generic module containing both algorithms presented in this paper which can be incorporated into an arbitrary peer-to-peer overlay. We briefly describe PeerCounter 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. Any synchronization assumptions made in our analysis or simulations are relaxed here since PeerCounter does not require any synchronization among processes.

PeerCounter is a command line application implemented in Java. It takes as parameters the server port to run, a symbol indicating which algorithm, Hops Sampling or 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, it may be provided by the underlying application. PeerCounter's API can be used to let applications layered under it utilize its estimation schemes. PeerCounter's API is described in [12].

Figure 5 demonstrates results for the continuous flavor of the Hops Sampling approach for static groups with sizes of 6430 nodes. Estimates are found to lie on

average within 10% of the actual group size. We test both the standard estimation scheme and the alternative sampling scheme. The equation used for the first case is $size = 2^{(averageNumberOfHops+1.75)}$. This is very close to the one extracted by using our simulations in Section 5, with only a small difference in the constant factor.

Figure 6 shows results for the continuous flavor of the Interval Density approach with random interval selection. As shown 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. More evaluation results can be found in [12].

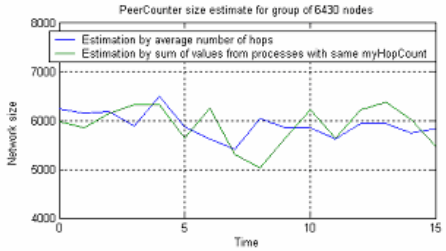

Figure 5. PeerCounter estimates for Hops Sampling scheme on groups of 6430 nodes (gossipTo=2, gossipFor=1, minHopsReporting=4, gossipsAccounted=5, gossipsDropped=1).

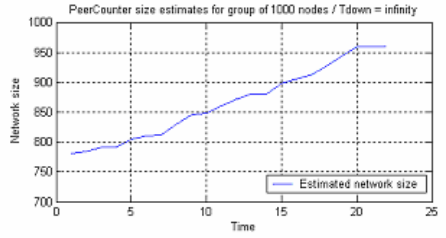

Figure 6. PeerCounter estimates for Interval Density scheme on groups of 1,000 nodes (MAXMEMORY=100, T_{down}= infinity).

7. Conclusions

Estimating the size of a decentralized group of processes connected within an overlay is a difficult problem, especially when the group is dynamic and contains thousands of processes. Previous solutions to the problem make assumptions that may be unscalable or limit applicability. In this paper, we have proposed 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 most overlays. Both approaches only require the overlay graph among all processes in the system to be connected. Micro-benchmarks and trace-based simulations have shown that both above approaches are able to obtain an estimate within a few percentage points of the actual group size. Both algorithms appear at http://kepler.cs.uiuc.edu/~psaltoul/peerCounter/ as an open-source size estimation utility called PeerCounter that can be incorporated into any peer-to-peer overlay.

8. References

[1] B. Awerbuch and C. Scheideler, "Robust Distributed Name Service", In Proc. 3rd International Workshop on Peer-to-Peer Systems (IPTPS), La Jolla, CA, USA, February 2004.

[2] N. T. J. Bailey, "Epidemic Theory of Infectious Diseases and its Applications", *Journal*, Hafner Press, 2nd Edition, 1975.

[3] 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.

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

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

[6] 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. 6th ACM Symposium on Principles of Distributed Computing (PODC), Vancouver, British Columbia, Canada, August 1987.

[7] P. Th. Eugster, R. Guerraoui, S. B. Handurukande, A.M. Kermarrec and P. Kouznetsov, "Lightweight probabilistic broadcast", *Journal*, ACM Transactions on Computer Systems, Vol. 21, No. 4, pp. 341-374, November 2003.

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

[9] 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. 2nd International Workshop on Peer-to-peer Systems, Springer LNCS, Vol. 2735, pp. 160-169, 2003.

[10] M. Jelasity and M. Preuss, "On Obtaining Global Information in a Peer-to-Peer Fully Distributed Environment", Springer LNCS, Vol. 2400, pp. 573-577, 2002.

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

[12] D. Kostoulas, D. Psaltoulis, I. Gupta, K. Birman and A. Demers, "Decentralized Schemes for Size Estimation in Large and Dynamic Groups", *Technical Report*, UIUCDCS-R-2005-2524, Dept. of Computer Science, University of Illinois, Urbana-Champaign, February 2005

[13] A.M. Kermarrec and L. Massoulie, *Private communication*, 2003.

[14] D. Malkhi and K. Horowitz, "Estimating network size from local information", *Journal*, ACM Information Processing Letters, Vol. 88, Issue 5, pp. 237-243, December 2003.

[15] R. van Renesse, Y. Minsky and M. Hayden, "A gossip-style failure detection service", Middleware '98, Lancaster, England, September 1998.

[16] A. Rowstron and P. Druschel, "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer Systems", In Proc. IFIP / ACM Middleware, pp. 329-350, November 2001.

[17] I. Stoica, R. Morris, D. Karger, F. Kaashoek and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for Internet applications", ACM SIGCOMM Conference, San Diego, CA, USA, August 2001.