Epidemic-Style Proactive Aggregation in Large Overlay Networks*

Márk Jelasity†
University of Bologna, Italy
jelasity@cs.unibo.it

Alberto Montresor
University of Bologna, Italy
montreso@cs.unibo.it

Abstract

Aggregation—that is, the computation of global properties like average or maximal load, or the number of nodes—is an important basic functionality in fully distributed environments. In many cases—which include protocols responsible for self-organization in large-scale systems and collaborative environments—it is useful if all *nodes know the value of some aggregates* continuously. *In this paper we present and analyze novel protocols capable of providing this service. The proposed* anti-entropy aggregation *protocols compute different aggregates of component properties like extremal values, average and counting. Our protocols are inspired by the anti-entropy epidemic protocol where random pairs of databases periodically resolve their differences. In the case of aggregation, resolving difference is generalized to an arbitrary (numeric) computation based on the states of the two communicating peers. The advantage of this approach is that it is proactive and "democratic", which means it has no performance bottlenecks, and the approximation of the aggregates is present continuously at all nodes. These properties make our protocol suitable for implementing e.g. collective decision making or automatic system maintenance based on global information in a fully distributed fashion. As our main contribution we provide fundamental theoretical results on the proposed averaging protocol.*

1. Introduction

The latest generation of peer-to-peer (P2P) networks are typically self-organizing and fully distributed systems [10, 12]. Unlike many traditional distributed systems, no central authority controls the various components. Instead, a dynamically changing overlay network is maintained, where nodes appear and disappear continuously, and dynamic "cooperation" links among nodes might be created and removed based on the requirements of the particular application. Such systems are attractive for several reasons, such as the lack of single points of failure, the potential to scale to millions of nodes and the inexpensiveness of the resulting distributed computing platforms.

However, such fully distributed platforms have certain drawbacks as well. Control and monitoring is difficult and performing computations poses the challenge of orchestrating the potentially huge number of participating nodes. One useful functional building block that has emerged recently is *aggregation* [13]. Aggregation is the collective name of many functions that provide global information about a system, like extremal values of some property, average, counts, etc. Aggregation can provide users or participants of a P2P network with important information like the number of nodes connected to the network, the identity of the most powerful peer in a grid or the total amount of free space in a distributed storage.

In many cases it would be very useful if *all* nodes knew the value of some aggregate *continuously*, in an *adaptive* fashion. Adaptivity means that if the aggregate changes due to network dynamism or variations in the values to be aggregated, the output of the aggregation protocol should follow this change reasonably quickly. Potential beneficiaries of an efficient implementation of this functionality include protocols responsible for self-organization in large-scale systems and collaborative environments. Our motivation is to offer an efficient and robust solution to this problem.

1.1. Anti-Entropy Aggregation

Our approach is based on a modified and generalized push-pull anti-entropy protocol. We assume that each node n_i in the network has a non-empty set of neighbors and has a numeric attribute, or value, a_i. Aggregation is performed over the set of these values. Node n_i also stores an approximation x_i of the aggregate.

The core algorithm run by each node is shown in Figure 1. This algorithm will be the subject of our theoretical analysis. The two functions GETWAITINGTIME and AGGREGATE determine the dynamics of the system, together with the neighbor set, which defines the overlay topology.

If not otherwise stated, GETWAITINGTIME returns the constant Δt which is the parameter of the protocol and which we shall call the *cycle length* (even though there are no "real" cycles in the system since each node is au-

*Proc. ICDCS 2004 pp. 102–109.
†also with RGAI, MTA SZTE, Szeged, Hungary

```
// the active process of the protocol on node n_i
do forever
  wait(getWaitingTime())
  n_j = selectRandomNeighbor()
  // perform elementary aggregation step
  send x_i to n_j
  receive x_j from n_j
  x_i = aggregate(x_i, x_j)

// reply on node n_j
receiveApproximation(x_i, n_i)
  send x_j to n_i
  x_j = aggregate(x_j, x_i)
```

Figure 1. The skeleton of the anti-entropy aggregation protocol run by all nodes. The active process is shown on node n_i and the passive reply on node n_j activated by the active process by sending the approximation x_i.

tonomous). In Section 3 we will introduce randomized versions of GETWAITINGTIME which are interesting from a theoretical point of view.

Function AGGREGATE is used to implement the desired aggregation function. For example, finding the maximum can be implemented using AGGREGATE_MAX that returns the maximum of its parameters. We will not discuss this algorithm any further because the behavior of this protocol from the point of view of the spreading of the true maximum is identical to that of the push-pull epidemic broadcast, which is well studied [4]. Nevertheless it is interesting to point out this application.

From now on we will focus on AGGREGATE_AVG which returns the average of its parameters. Being able to calculate the average already makes it possible to calculate any moments (using averages of different powers of the value set), the size of the system, the sum of the value set, etc.

In this simplistic presentation of the protocol we assume that there is a synchronized starting time (time 0) when at all nodes $x_i = a_i$. One *cycle* of the protocol lasts from time point $k\Delta t$ to $(k+1)\Delta t$ where k is an integer and Δt is the expected return value of GETWAITINGTIME. In Section 3 we analyze this version of the protocol.

1.2. Related Work

Epidemic protocols Epidemic protocols are becoming more and more popular since the publication of the seminal paper by Demers et al. [4]. A recently completed survey by Eugster et al. provides an excellent introduction to the field [6]. Epidemic algorithms have been applied to solving several practical problems like database replication [4], failure detection [15] and resource monitoring [14]. A large body of theoretical work is also available due to the general importance of understanding epidemics [1] and its close relation to random graph theory [3].

Our present work does not introduce new epidemic protocols. Instead, we show how to apply anti-entropy [4], a well-known epidemic protocol, to the aggregation problem. The resulting solution is efficient and enables a network of nodes to collectively find aggregates in such a way that all nodes know the value of the aggregate continously and in an adaptive fashion.

Aggregation in distributed environments The field of distributed computation of aggregates is less established than epidemic protocols. An overview of the problem can be found in [13].

A prominent approach is Astrolabe [14] which is a hierarchical architecture for aggregation in large distributed systems. Anti-entropy aggregation is substantially different in that it is extremely simple, lightweight, and targeted to unstructured, highly dynamic environments. In the case of our protocol the overhead of implementation and maintenance is diminishing. A related approach is [8] which is also based on a hierarchical approach. While building hierarchies indeed reduces the cost of finding the aggregates but introduces additional overhead having to maintain this hierarchical topology in a dynamic distributed environment. Moreover, due to being hierarchical, it also needs extra effort and protocols to broadcast the result continuously over the network if all nodes need to know the result continuously.

Another recent work [2] discusses many approaches, based on spanning tree induction and using other, more redundant topologies. The main difference from our approach is that the protocols described there are *reactive*: aggregation is initialized from a certain point and the result is known by only that node. This makes it hard to adopt for solving our present research problem, similarly to the other approaches mentioned above.

Membership management This area is also relevant to help put our results in the right context. Anti-entropy aggregation is a protocol which assumes that each node has a neighbor set, a set of other nodes in the network to potentially communicate with. However, the protocol does not address the issue of the maintenance of these sets. Instead, it can be used along with any membership management protocol. Since in this paper we focus on random networks, it is important to mention that there are membership protocols that can maintain an approximately random topology [5, 7, 9].

1.3. Contribution

In the light of related work, our contribution can be summarized as follows: (i) proposing anti-entropy aggregation,

an efficient, simple and lightweight protocol for solving the proactive aggregation problem we sketched above and (ii) theoretical analysis of anti-entropy averaging which includes giving exponential convergence rates for several versions of the protocol.

1.4. Outline of the paper

In Section 2 we summarize the assumptions that we adopt in our discussion. In Section 3 we focus on the speed of convergence of the approximations to the true average. We give several explicit convergence rates which hold under slightly different assumptions. In particular, we show that convergence is *exponentially fast*. We also discuss the effects of both node failures and message loss. Finally, experimental validation is given. Section 4 contains an illustrative example of the application of the protocol for network size estimation.

2. System Model

We consider a network of a large collection of *nodes* that communicate through the exchange of messages and are assigned a unique identifier.

We assume that nodes are connected by an existing *physical network*. Even though the protocols we suggest can be deployed on arbitrary physical networks, including sensor and ad-hoc networks, in the present work we consider only fully connected networks, like the Internet, where each node can (potentially) communicate with each other node. The physical network provides only the possibility of communication. To actually communicate, each node has to know the identifiers of a set of other nodes (its *neighbors*). This neighborhood relation over the nodes defines the topology of the *overlay network*.

Our theoretical analysis assumes that each node has access to a hardware clock without drift and a common point of reference is known in time. We further assume that communication takes zero time. While these assumptions are rather strong, in [11] we offer solutions to practical aspects of the protocol, like relaxing the synchronization assumption and introducing mechanisms for adaptivity and fault tolerance. Section 4 contains an illustrative example that points to this direction.

3. Theoretical Analysis

We begin by introducing the conceptual framework and notations to be used for the purpose of the mathematical analysis. We proceed by proving convergence rates for various algorithms. Our results are validated and illustrated by numerical simulation when necessary.

// vector a is the input
do N times
  $(i, j) =$ getPair()
  // perform elementary variance reduction step
  $a_i = a_j = (a_i + a_j)/2$
return a

Figure 2. The skeleton of the algorithm AVG**.**

3.1. Basic Concepts and Notations

For the purpose of mathematical analysis we translate the networking terminology into mathematical structures and concepts. We will treat anti-entropy averaging as an iterative variance reduction algorithm over a vector of numbers. In this framework, we can formulate our approach as follows. We are given an initial vector of numbers $\mathbf{a}_0 = (a_{0,1} \dots a_{0,N})$. We shall model this vector by assuming that $a_{0,1}, \dots, a_{0,N}$ are independent random variables with identical expected values and a finite variance.

The assumption of identical expected values is not as strong as it seems. The protocol is not sensitive to the ordering of values, so after any permutation of the initial values the statistical behavior remains the same. Starting with random variables $a_{0,1}, \dots, a_{0,N}$ with arbitrary expected values, after a random permutation the new value at index i, b_i will have the distribution

$$P(b_i < x) = \frac{1}{N} \sum_{j=1}^{N} P(a_j < x). \qquad (1)$$

That is, we obtain an equivalent probability model where the distribution of random variables $b_0, \dots, b_N$ is identical. Note that in this case the assumption of independence is violated, but—in the case of large networks—only to an insignificant extent.

When considering the network as a whole, one cycle of anti-entropy averaging can be looked at as an algorithm (let us call it AVG) which takes a vector as a parameter and produces a new vector of the same length (N). Furthermore, the consecutive cycles of the protocol result in a series of vectors $\mathbf{a}_1, \mathbf{a}_2, \dots$ where $\mathbf{a}_{i+1} = \text{Avg}(\mathbf{a}_i)$. The elements of vector $\mathbf{a}_i$ will be denoted by $\mathbf{a}_i = (a_{i,1} \dots a_{i,N})$

Figure 2 shows algorithm AVG, which takes $\mathbf{a}$ as a parameter and modifies it in place producing a new vector. Note that all the practical aspects of the overlay topology, synchronization (or the lack of it), and eventual node failures can be modeled by properties and constraints of the function GETPAIR. For example, if node j is not in the neighborhood of node i then the pair (i, j) will never be returned. If node i is not reachable, no pairs containing it will be returned. Also, the distributed and local nature of the epidemic protocol underlying this model can be expressed by the constraint that the returned pair cannot be determined

(or affected) by some global property of the value vector, like the maximum of the values for instance.

3.2. Convergence

We introduce the following notations for empirical statistics:

$$\overline{\mathbf{a}_i} = \frac{1}{N}\sum_{k=1}^{N} a_{i,k} \tag{2}$$

$$\sigma_i^2 = \sigma_{\mathbf{a}_i}^2 = \frac{1}{N-1}\sum_{k=1}^{N}(a_{i,k} - \overline{\mathbf{a}_i})^2 \tag{3}$$

Only linear operations are performed on the vector elements so without loss of generality we will assume that the common expected value of the elements of $\mathbf{a}_0$ is zero. The purpose of this choice is merely to simplify our expressions. In particular, for any vector $\mathbf{a}$, if the elements of $\mathbf{a}$ are independent random variables with zero expected value then

$$E(\sigma_{\mathbf{a}}^2) = \frac{1}{N}\sum_{k=1}^{N} E(a_k^2). \tag{4}$$

Furthermore, the elementary variance reduction step in which both selected elements are replaced by their average does not change the sum of the elements in the vector so $\overline{\mathbf{a}_i} \equiv \overline{\mathbf{a}_0}$ for all cycles $i = 1, 2, \ldots$. This property is very important because it guarantees that the algorithm does not introduce any errors into the approximation. This means that from now on we can focus on variance. Clearly, if the expected value of σ_i^2 tends to zero with i tending to infinity then the variance of all vector elements will tend to zero as well so the correct average μ_0 will be approximated locally with arbitrary accuracy by each node.

Let us begin our analysis of the convergence of variance with some fundamental observations.

Lemma 1. *Let $\mathbf{a}'$ be the vector that we get by replacing both a_i and a_j with $(a_i + a_j)/2$ in vector $\mathbf{a}$. If $\mathbf{a}$ contains uncorrelated random variables with expected value 0 then the expected value of the resulting variance reduction is given by*

$$E(\sigma_{\mathbf{a}}^2 - \sigma_{\mathbf{a}'}^2) = \frac{1}{2(N-1)}E(a_i^2) + \frac{1}{2(N-1)}E(a_j^2). \tag{5}$$

Proof. Simple calculation using the fact that if a_i and a_j are uncorrelated then $E(a_i a_j) = E(a_i)E(a_j) = 0$. □

Considering also (4), an intuitive interpretation of this lemma is that after an elementary variance reduction step both participating nodes will contribute only approximately the half of their original contribution to the overall expected variance, provided they are uncorrelated. In the extreme case of maximal correlation ($a_i \equiv a_j$) the variance reduction is zero. From this it can be seen that the assumption of uncorrelatedness is crucial.

Having this observation and (4) in mind let us consider instead of $E(\sigma_i^2)$ the average of a vector of values $\mathbf{s}_i = (s_{0,1} \ldots s_{0,N})$ that are defined as follows. The initial vector $\mathbf{s}_0 \equiv (a_{0,1}^2 \ldots a_{0,N}^2)$ and $\mathbf{s}_i$ is produced in parallel with $\mathbf{a}_i$; using the same pair (i, j) returned by GETPAIR and used by performing the elementary averaging step (see Figure 2), we perform the step $s_i = s_j = (s_i + s_j)/4$ as well. This way, according to Lemma 1, $E(\overline{\mathbf{s}_i})$ will emulate the evolution of $E(\sigma_i)$ with a high accuracy provided that each pair of values a_i and a_j selected by each call to GETPAIR are practically uncorrelated. Intuitively, this assumption can be expected to hold if the original values in $\mathbf{a}_0$ are uncorrelated and GETPAIR is "random enough" not to introduce significant correlations.

Working with $E(\overline{\mathbf{s}_i})$ instead of $E(\sigma_i^2)$ is easier in a mathematical sense but at the same time we capture the dynamics of the system with a high accuracy as will be confirmed by empirical simulations.

Using this simplified model, now we turn to the following theorem which will be the basis of our results on specific implementations of GETPAIR. First let us define random variable ϕ_k to be the number of times index k was selected as a member of the pair returned by GETPAIR in algorithm AVG during the calculation of $\mathbf{a}_{i+1}$ from the input $\mathbf{a}_i$. In networking terms, ϕ_k is the number of peer communications node k was involved in during cycle i.

Theorem 1. *Let us assume that* GETPAIR *has the following properties.*

1. *The random variables $\phi_1, \ldots, \phi_N$ are identically distributed. Let ϕ denote a random variable with this common distribution.*
2. *After (i, j) is returned by* GETPAIR *the number of times i and j will be selected by the remaining calls to* GETPAIR *has identical distribution.*

Then we have

$$E(\overline{\mathbf{s}_{i+1}}) = E(2^{-\phi})E(\overline{\mathbf{s}_i}) \tag{6}$$

Proof. We only sketch the proof here. The basic idea is thinking of the value $s_{i,k}$ as a quantity of some material. According to the definition of $s_{i,k}$, each time k is selected by GETPAIR we loose half of the material and the remaining material will be divided among the locations. Using assumption 2 we can observe that it does not matter where a given piece of the original material ends up, it will have the same chance of loosing its half then the proportion that stays at the original location. That means that the original material will lose its half as many times on average as the expected number of selection of k by GETPAIR, hence

the term $\frac{1}{N}E(2^{-\phi_k})E(s_{i,k}) = \frac{1}{N}E(2^{-\phi})E(s_{i,k})$. Applying this for all k and summing up the terms we have the proof. □

This theorem will allow us to concentrate on $E(2^{-\phi})$. According to the arguments on the role of $\mathbf{s}_i$ we expect that

$$E(\sigma_{i+1}^2) \approx E(2^{-\phi})E(\sigma_{i+1}^2) \tag{7}$$

will be true.

3.3. Case Studies

In this section we give explicit convergence rates for specific implementations of GETPAIR. In all cases it will be assumed that the overlay topology is the complete graph, that is, whenever a random neighbor has to be selected, it can be considered as sampling the whole set of nodes. Note that since each node uses only a limited number of random neighbors only, these results hold also in random graphs which are connected and where a sufficient number (normally a small constant number) of random edges is present from each node. This latter type of graph is much less unrealistic then the fully connected one, as scalable and robust protocols are available which approximate such a random structure [5, 7, 9].

3.3.1. Perfect matching: the optimal strategy

Let us begin with an artificial example which represents the optimal implementation of GETPAIR. We will call this implementation GETPAIR_PM where PM stands for perfect matching. Unfortunately this implementation cannot be mapped to an efficient distributed P2P protocol because it requires global knowledge of the system. What makes it interesting is the fact that it is optimal under the assumptions of Theorem 1 so it can serve as a reference for comparison with more practical approaches.

GETPAIR_PM works as follows. Before the first call $N/2$ pairs of indeces are created (let us assume that N is even) in such a way that each index is present in exactly one pair. In other words, a perfect matching over the overlay topology is created. Subsequently these pairs are returned, each exactly once. When the pairs run out (after the $N/2$-th call) another perfect matching is created which contains none of the pairs from the first perfect matching, and these pairs are returned by the second $N/2$ calls.

Using the assumption that any pair of values from $\mathbf{a_i}$ that are connected by an edge in the overlay topology are uncorrelated we can easily construct two non-overlapping perfect matchings that define uncorrelated pairs. We can verify the assumptions of Theorem 1 as well: (i) the algorithm is clearly value- and index-blind and (ii) after the first selection of any index i it is guaranteed that it will be selected exactly once more.

We can therefore apply the theorem to GETPAIR_PM. The convergence rate is given by

$$E(2^{-\phi}) = E(2^{-2}) = 1/4. \tag{8}$$

We now prove the optimality of this convergence rate under the assumptions of Theorem 1.

Lemma 2. *For any random variable X if $E(X) = 2$ then the expected value $E(2^{-X})$ is minimal if $P(X = 2) = 1$.*

Proof. The proof is straightforward but technical so we only sketch it. It can be shown that for any distribution different from $P(X = 2) = 1$ we can decrease the value $E(2^{-X})$ by transforming the distribution into a new one which still satisfies the constraint $E(X) = 2$. The basic observation is that if $P(X = 2) < 1$ then there are at least two indeces $i < 2$ and $j > 2$ for which $P(X = i) > 0$ and $P(X = j) > 0$. It can be technically verified that if we reduce both $P(X = i)$ and $P(X = j)$ while increasing $P(X = 2)$ by the same amount in such a way that $E(X) = 2$ still holds then $E(2^{-X})$ will decrease. □

3.3.2. Random selection

Moving towards more practical implementations of GETPAIR, our next example is GETPAIR_RAND which works simply choosing a random edge from the underlying overlay topology, picking each edge with the same probability.

GETPAIR_RAND can easily be implemented as a distributed P2P protocol. When iterating ALG, the waiting time between the selection of a given node can be described by the exponential distribution. In a distributed implementation a given node can approximate this behavior by waiting for a time interval randomly drawn from this distribution before initiating communication. However, as we will see, GETPAIR_RAND is not a very efficient pair selector, so it serves only as a stepping stone for the analysis of a more practical protocol.

Like for GETPAIR_PM, the assumptions of Theorem 1 hold: (i) the algorithm is clearly value- and index-blind and (ii) all indeces have exactly the same probability to be selected after each elementary variance reduction step. There is no strict guarantee that no correlated pairs will be returned. However, due to the randomness of the algorithm and the fully connected (or random, respectively) overlay topology we can expect to observe only negligible correlation.

Now, to get the convergence rate, the distribution of ϕ can be approximated by the Poisson distribution with parameter 2, that is

$$P(\phi = j) = \frac{2^j}{j!}e^{-2}. \tag{9}$$

Substituting this into the expression $E(2^{-\phi})$ we get

$$E(2^{-\phi}) = \sum_{j=0}^{\infty} 2^{-j} \frac{2^j}{j!} e^{-2} = e^{-2} \sum_{j=0}^{\infty} \frac{1}{j!} = e^{-2} e = e^{-1}. \tag{10}$$

Comparing the performance of GETPAIR_RAND and GETPAIR_PM we can see that convergence is significantly slower than in the optimal case (the rates are $1/e \approx 1/2.71$ vs. $1/4$).

We ran AVG with GETPAIR_RAND using several network sizes, on both the fully connected topology and on a random topology with a fixed view size of 20. The results are shown if Figure 3. On Figure 3(a) we can see that convergence is independent of network size and also that theory predicts the observed convergence rate with very high accuracy in the case of the fully connected topology. As could be expected, in the case of the random 20-regular graph we can observe a slightly slower convergence due to the violation of the constraints of Theorem 1, but the difference is insignificant.

Figure 3(b) reveals however that the difference becomes more significant during the iteration of AVG. A possible reason is that correlation accumulates during the cycles due to the more limited amount of neighborhood information available.

3.3.3. A practical protocol

Building on the results we have so far it is possible to analyze a practically relevant version of the protocol, still assuming fully connected (or random) overlay topology.

This implementation of pair selection will iterate over the node set in a fixed order, picking a random neighbor for each node thereby generating the pair. We call this algorithm GETPAIR_SEQ. This algorithm can be implemented as a distributed P2P protocol easily. Each node has to pick a neighbor periodically in regular intervals and perform the variance reduction step with the neighbor. As we will see, this protocol is not only implementable in a distributed way but its performance is also superior to that of GETPAIR_RAND although not reaching GETPAIR_PM.

Unfortunately, GETPAIR_SEQ does not satisfy assumption 2 of Theorem 1. Therefor we have to apply a trick to make our framework applicable. First of all, note that when using GETPAIR_SEQ $\phi = 1 + \phi'$ where ϕ' has a Poisson distribution with parameter 1. We introduce another, non-practical implementation of GETPAIR_SEQ for which ϕ has the same distribution.

This new implementation, called GETPAIR_PMRAND combines GETPAIR_PM and GETPAIR_RAND within one cycle. During the first $N/2$ calls GETPAIR_PMRAND behaves like GETPAIR_PM and during the remaining calls it behaves like GETPAIR_RAND. Obviously, GETPAIR_PMRAND also has the same $\phi = 1 + \phi'$ calling frequency for each node.

Based on the argumentation presented in connection with GETPAIR_PM and GETPAIR_RAND we can apply Theorem 1 to GETPAIR_PMRAND. The distribution of ϕ can be approximated by $\phi = 1 + \phi'$ where ϕ' has the Poisson distribution with parameter 1, that is, for $j > 0$

$$P(\phi = j) = P(\phi' = j - 1) = \frac{1}{(j-1)!} e^{-1}. \tag{11}$$

Substituting this into the expression $E(2^{-\phi})$ we get

$$\begin{aligned} E(2^{-\phi}) &= \sum_{j=1}^{\infty} 2^{-j} \frac{1}{(j-1)!} e^{-1} \\ &= \frac{1}{2e} \sum_{j=1}^{\infty} \frac{2^{-(j-1)}}{(j-1)!} = \frac{1}{2e} \sqrt{e} = \frac{1}{2\sqrt{e}}. \end{aligned} \tag{12}$$

Comparing the performance to GETPAIR_RAND and GETPAIR_PM we can see that convergence is slower than in the optimal case but faster than in the random case (the rates are $1/e \approx 1/2.71$, $1/2\sqrt{e} \approx 1/3.3$ and $1/4$, respectively).

As we already saw from the experimental analysis of GETPAIR_RAND, this theorem can be considered an extremely accurate approximation (in the case of GETPAIR_PM the convergence theorem was exact). This means that the only assumption that remains to be validated experimentally is the substitution of GETPAIR_SEQ with GETPAIR_PMRAND. We ran AVG with GETPAIR_SEQ using several network sizes, on both the fully connected topology and on a random topology with a fixed view size of 20.

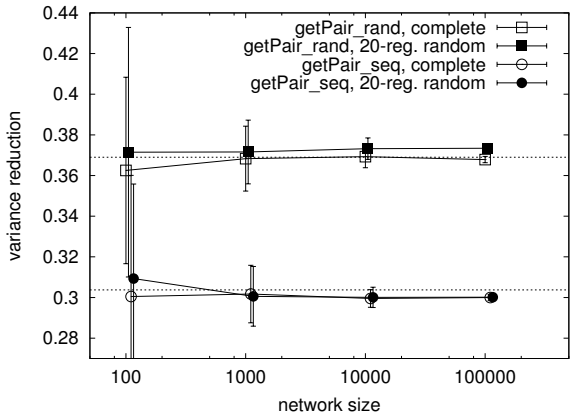
Similarly to GETPAIR_RAND, on Figure 3(a) we can see that convergence seems to be independent of network size. What is more interesting however is that both topologies result in a slightly better convergence than predicted. This effect is due to the fact that to derive the convergence rate GETPAIR_PMRAND was used instead of GETPAIR_SEQ. We can also notice that there is no observable difference between the random and fully connected topologies.

The difference becomes more significant when iterating AVG as Figure 3(b) shows, just like in the case of GETPAIR_RAND. However, probably due to its more regular nature, GETPAIR_SEQ seems to be less sensitive to the accumulation of correlation and it tolerates the random topology better.

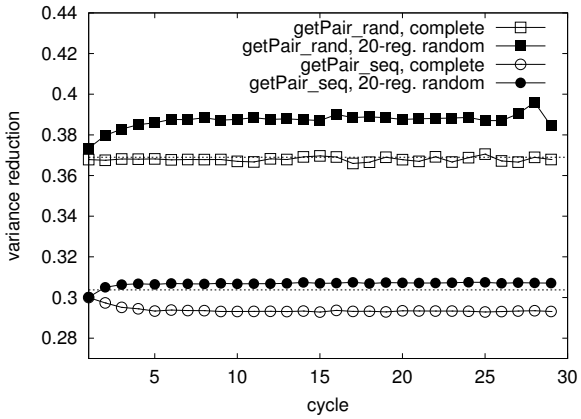
4. An Example: Network Size Estimation

In [11] we offer solutions to practical aspects of the protocol, like relaxing the synchronization assumption and introducing mechanisms for adaptivity and fault tolerance. The present section contains an illustrative example that points to this direction.

To allow the protocol to be adaptive, we need to extend anti-entropy aggregation with a restarting mechanism. First, we have to solve the problem of termination. The solution



(a) Average variance reduction after one execution of AVG on a vector of uncorrelated values (σ_1^2/σ_0^2).



(b) Average variance reduction for network size $N = 100000$ when iterating algorithm AVG ($\sigma_i^2/\sigma_{i-1}^2$).

Figure 3. Simulation results using several network sizes, on both the fully connected topology and on a random topology with a fixed view size of 20. Values are averages over 50 independent runs. Dotted lines show the two theoretically predicted reduction rates: $1/e \approx 0.368$ and $1/(2\sqrt{e}) \approx 0.303$.

that we adopt is that each node executes the protocol for a predefined number of cycles k, depending on the required accuracy of the output (see convergence rates given in Section 3).

To make the protocol truly adaptive, we divide the execution of the aggregation protocol in consecutive *epochs* of length ΔT and start a new instance of the protocol in each epoch. Depending on the ratio between ΔT and $k\Delta t$, it is possible that different epochs of the protocol are executed concurrently in the network. Thus, messages exchanged for a particular epoch have to be tagged with unique identifiers, e.g. obtained by using a monotone counter maintained at each node.

When a node joins the network, it contacts one of the nodes that are already participating in the aggregation protocol. Here, we assume the existence of an out-of-band mechanism to discover such nodes. The existing node provides the new node with the next epoch identifier and the amount of time left until the next run starts. The node will start to actively participate in the aggregation protocol after the specified units of time, measured on its local clock, tagging its messages with the suggested identifier. Additionally, to avoid drift, if a node receives a message with an identifier larger than its current one, it switches to the new epoch immediately. This solution is sufficient because this way a new epoch-start spreads like an epidemic broadcast which is exponentially fast.

As an example application, let us intruduce a mechanim for *network size estimation*. We base this protocol on the following observation: if exactly on of the values stored by nodes is equal to 1 and all the others are equal to 0, then the average is exactly $1/N$ so N can be calculated directly. We can implement this idea by enabling multiple nodes to start concurrent instances of the averaging protocol. Each concurrent instance is lead by a different node. Messages and data related to an instance are tagged with a unique identifier (e.g., the address of the leader). The leader will initialize its approximation to 1, and all the rest of the nodes that are reached by this instance start to behave as if they had 0 as initial value.

To bound the number of instances running concurrently we allow each node to become a leader at the beginning of each epoch with a sufficiently small probability that can also depend on the previous approximation of network size.

Our simulations are reported in Figure 4. A new epoch was started every 30 cycles. Two values are reported: one is the actual size of the network, and the other is the observed estimate. Converged estimates are reported at the end of each epoch. The error bars show the range of estimates obtained by all nodes that participated in the full epoch.

The behavior of the network is as follows: the size oscillates between 90.000 and 110.000. In addition to nodes added and removed because of the oscilattion, 100 nodes are removed from the network and 100 nodes are added to simulate fluctuation. Nodes that join the network are not allowed to participate in the current epoch, as described earlier. This is necessary to make sure each epoch converges to the correct average *at the start* of the epoch. Continuously adding new nodes would make it impossible to achieve any convergence.

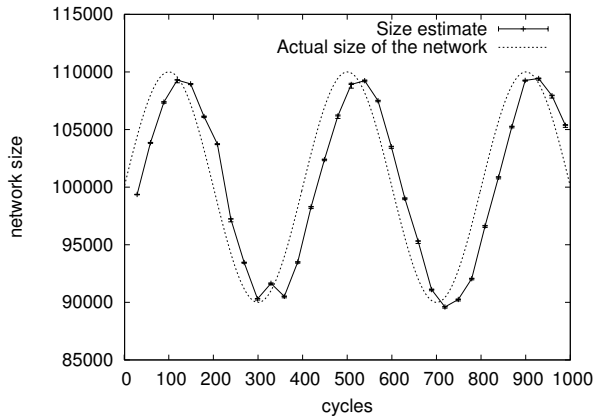


Figure 4. Network size estimation by anti-entropy counting.

This scenario is close to a realistic P2P network, where the size is not constant but oscillates between a minimum and a maximum (for example on a day/night alternation basis). We can see that the curve of estimates is similar to the actual size curve, only translated by an epoch. This is because new nodes do not participate in the aggregation so the final estimate describes the state of the network when the epoch was started.

5. Conclusions

Let us summarize the practical implications of our theoretical results.

The protocol is scalable, as all convergence results presented in the paper are independent of network size N. Furthermore, the distributions of the number of communications (ϕ) at a fixed node are also independent of N for both GETPAIR_RAND and GETPAIR_SEQ. In other words, increasing the system size will not slow convergence down and will not increase resource requirements on the particular nodes. Furthermore, since ϕ is independent of location, there are no performance peaks, the costs are distributed very smoothly over the network. However, the overall traffic in the *entire* network will grow linearly.

The protocol is efficient, as a good approximation of the average is obtained very quickly by *all* nodes due to the exponential convergence. Even in the worst case we examined, with GETPAIR_RAND, the variance over the network will decrease 99.9% in $\ln 1000 \approx 7$ cycles of AVG.

Recall that the main assumption was that the overlay network has either a fully connected topology or a connected unbiased random topology. This might be seen as a limiting factor to practical applicability, but as we mentioned much effort has been devoted into developing topology-management protocols recently that provide nodes with (approximately) random neighbors while also taking care of connectivity [5, 7, 9].

Our future work is targeted towards extending our analysis to more realistic topologies and combining anti-entropy aggregation with membership protocols that maintain a sufficiently random (or at least predictable) topology.

Acknowledgment

This work was partially supported by the Future & Emerging Technologies unit of the European Commission through Project BISON (IST-2001-38923). The basic idea of anti-entropy epidemic style variance reduction is one of the countless ideas that emerged during our many inspiring discussions with Maarten van Steen and Wojtek Kowalczyk.

References

[1] N. T. J. Bailey. *The mathematical theory of infectious diseases and its applications*. Griffin, London, second edition, 1975.

[2] M. Bawa, H. Garcia-Molina, A. Gionis, and R. Motwani. Estimating aggregates on a peer-to-peer network. submitted for publication.

[3] B. Bollobás. *Random graphs*. Cambridge University Press, Cambridge; New York, second edition, 2001.

[4] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry. Epidemic algorithms for replicated database management. In *Proceedings of the 6th Annual ACM Symposium on Principles of Distributed Computing (PODC'87)*, pages 1–12, Vancouver, Aug. 1987. ACM.

[5] P. T. Eugster, R. Guerraoui, S. B. Handurukande, A.-M. Kermarrec, and P. Kouznetsov. Lightweight probabilistic broadcast. *ACM Transactions on Computer Systems*, 21(4):341–374, 2003.

[6] P. T. Eugster, R. Guerraoui, A.-M. Kermarrec, and L. Massoulié. From epidemics to distributed computing. *IEEE Computer*. to appear.

[7] A. J. Ganesh, A.-M. Kermarrec, and L. Massoulié. Peer-to-peer membership management for gossip-based protocols. *IEEE Transactions on Computers*, 52(2), Feb. 2003.

[8] I. Gupta, R. van Renesse, and K. P. Birman. Scalable fault-tolerant aggregation in large process groups. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN'01)*, Göteborg, Sweden, 2001.

[9] M. Jelasity and M. van Steen. Large-scale newscast computing on the Internet. Technical Report IR-503, Vrije Universiteit Amsterdam, Department of Computer Science, Amsterdam, The Netherlands, Oct. 2002.

[10] D. S. Milojicic, V. Kalogeraki, R. Lukose, K. Nagaraja, J. Pruyne, B. Richard, S. Rollins, and Z. Xu. Peer-to-peer computing. Technical Report HPL-2002-57, HP Labs, Palo Alto, 2002.

[11] A. Montresor, M. Jelasity, and O. Babaoglu. Robust aggregation protocols for large-scale overlay networks. Technical Report UBLCS-2003-16, University of Bologna, Department of Computer Science, Bologna, Italy, Dec. 2003.

[12] A. Oram, editor. *Peer-to-Peer: Harnessing the Benefits of a Disruptive Technology*. O'Reilly, Mar. 2001.

[13] R. van Renesse. The importance of aggregation. In A. Schiper, A. A. Shvartsman, H. Weatherspoon, and B. Y. Zhao, editors, *Future Directions in Distributed Computing*, number 2584 in Lecture Notes in Computer Science, pages 87–92. Springer, 2003.

[14] R. Van Renesse, K. P. Birman, and W. Vogels. Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining. *ACM Transactions on Computer Systems*, 21(2), May 2003.

[15] R. van Renesse, Y. Minsky, and M. Hayden. A gossip-style failure detection service. In N. Davies, K. Raymond, and J. Seitz, editors, *Middleware '98*, pages 55–70. Springer, 1998.