Gossip-based Aggregation in Large Dynamic Networks*

Márk Jelasity, Alberto Montresor and Ozalp Babaoglu
Università di Bologna

Abstract

As computer networks increase in size, become more heterogeneous and span greater geographic distances, applications must be designed to cope with the very large scale, poor reliability, and often, with the extreme dynamism of the underlying network. *Aggregation* is a key functional building block for such applications: it refers to a set of functions that provide components of a distributed system access to global information including network size, average load, average uptime, location and description of hotspots, etc. Local access to global information is often very useful, if not indispensable for building applications that are robust and adaptive. For example, in an industrial control application, some aggregate value reaching a threshold may trigger the execution of certain actions; a distributed storage system will want to know the total available free space; load balancing protocols may benefit from knowing the target average load so as to minimize the load they transfer. We propose a gossip-based protocol for computing aggregate values over network components in a fully decentralized fashion. The class of aggregate functions we can compute is very broad and includes many useful special cases such as counting, averages, sums, products and extremal values. The protocol is suitable for extremely large and highly dynamic systems due to its proactive structure—all nodes receive the aggregate value continuously, thus being able to track any changes in the system. The protocol is also extremely lightweight making it suitable for many distributed applications including peer-to-peer and grid computing systems. We demonstrate the efficiency and robustness of our gossip-based protocol both theoretically and experimentally under a variety of scenarios including node and communication failures.

1 Introduction

Computer networks in general, and the Internet in particular, are experiencing explosive growth in many dimensions, including size, performance, user base and geographical span. The potential for communication and access to computational resources have improved dramatically both quantitatively and qualitatively in a relatively short time. New design paradigms such as peer-to-peer (P2P) [18] and grid computing [14] have emerged in response to these trends. The Internet, and all similar networks, pose special challenges for large-scale, reliable, distributed application builders. The "best-effort" design philosophy that characterizes such networks renders the communication channels inherently unreliable and the continuous flux of nodes joining and leaving the network make them highly dynamic. Control and monitoring in such systems are particularly challenging: performing global computations requires orchestrating a huge number of nodes.

In this paper, we focus on *aggregation* which is a useful building block in large, unreliable and dynamic systems [25]. Aggregation is a common name for a set of functions that provide a

summary of some global system property. In other words, they allow local access to global information in order to simplify the task of controlling, monitoring and optimization in distributed applications. Examples of aggregation functions include network size, total free storage, maximum load, average uptime, location and intensity of hotspots, etc. Furthermore, simple aggregation functions can be used as building blocks to support more complex protocols. For example, the knowledge of average load in a system can be exploited to implement near-optimal load-balancing schemes [12].

We distinguish *reactive* and *proactive* protocols for computing aggregation functions. Reactive protocols respond to specific queries issued by nodes in the network. The answers are returned directly to the issuer of the query while the rest of the nodes may or may not learn about the answer. Proactive protocols, on the other hand, continuously provide the value of some aggregate function to *all* nodes in the system in an *adaptive* fashion. By adaptive we mean that if the aggregate changes due to network dynamism or because of variations in the input values, the output of the aggregation protocol should track these changes reasonably quickly. Proactive protocols are often useful when aggregation is used as a building block for completely decentralized solutions to complex tasks. For example, in the load-balancing scheme cited above, the knowledge of the global average load is used by each node to decide if and when it should transfer load [12].

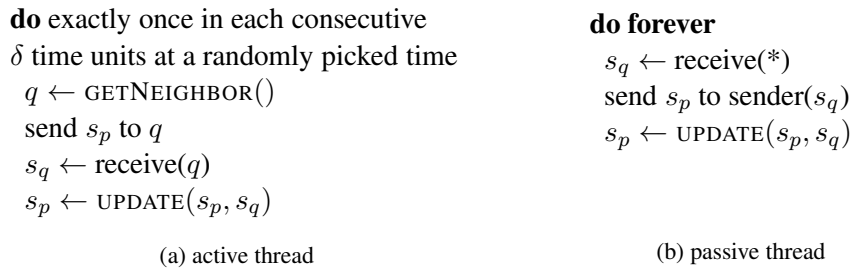
Contribution In this paper we introduce a robust and adaptive protocol for calculating aggregates in a proactive manner. We assume that each node maintains a local approximate of the aggregate value. The core of the protocol is a simple gossip-based communication scheme in which each node periodically selects some other random node to communicate with. During this communication the nodes update their local approximate values by performing some aggregation-specific and strictly local computation based on their previous approximate values. This local pairwise interaction is designed in such a way that all approximate values in the system will quickly converge to the desired aggregate value.

In addition to introducing our gossip-based protocol, the contributions of this paper are threefold. First, we present a full-fledged practical solution for proactive aggregation in dynamic environments, complete with mechanisms for *adaptivity, robustness* and *topology management*. Second, we show how our approach can be extended to compute complex aggregates such as variances and different means. Third, we present theoretical and experimental evidence supporting the efficiency of the protocol and illustrating its robustness with respect to node and link failures and message loss.

Outline In Section 2 we define the system model. Section 3 describes the core idea of the protocol and presents theoretical and simulation results of its performance. In Section 4 we discuss the extensions necessary for practical applications. Section 5 introduces novel algorithms for computing statistical functions including several means, network size and variance. Sections 6 and 7 present analytical and experimental evidence on the high robustness of our protocol. Section 8 describes the prototype implementation of our protocol on PlanetLab and gives experimental results of its performance. Section 9 discusses related work. Finally, conclusions are drawn in Section 10.

2 System Model

We consider a network consisting of a large collection of *nodes* that are assigned unique identifiers and that communicate through message exchanges. The network is highly dynamic; new



(a) active thread

(b) passive thread

Figure 1: Push-pull gossip protocol executed by node p. The local state of p is denoted as s_p.

nodes may join at any time, and existing nodes may leave, either voluntarily or by *crashing*. Our approach does not require any mechanism specific to leaves: spontaneous crashes and voluntary leaves are treated uniformly. Thus, in the following, we limit our discussion to node crashes. Byzantine failures, with nodes behaving arbitrarily, are excluded from the present discussion (but see [11]).

We assume that nodes are connected through an existing routed network, such as the Internet, where every node can potentially communicate with every other node. To actually communicate, a node has to know the identifiers of a set of other nodes, called its *neighbors*. This neighborhood relation over the nodes defines the topology of an *overlay network*. Given the large scale and the dynamicity of our envisioned system, neighborhoods are typically limited to small subsets of the entire network. The set of neighbors of a node (thus the overlay network topology) can change dynamically. Communication incurs unpredictable delays and is subject to failures. Single messages may be lost, links between pairs of nodes may break. Occasional performance failures (e.g., delay in receiving or sending a message in time) can be seen as general communication failures, and are treated as such. Nodes have access to local clocks that can measure the passage of real time with reasonable accuracy, that is, with small short-term drift.

In this paper we focus on node and communication failures. Some other aspects of the model that are outside of the scope of the present analysis (such as clock drift and message delays) are discussed only informally in Section 4.

3 Gossip-based Aggregation

We assume that each node in the network holds a numeric value. In a practical setting, this value can characterize any (possibly dynamic) aspect of the node or its environment (e.g., the load at the node, available storage space, temperature measured by a sensor network, etc.). The task of a proactive protocol is to continously provide all nodes with an up-to-date estimate of an aggregate function, computed over the values held by the current set of nodes.

3.1 The Basic Aggregation Protocol

Our basic aggregation protocol is based on the "push-pull gossiping" scheme illustrated in Figure 1. Each node p executes two different threads. The *active* thread periodically initiates an *information exchange* with a random neighbor q by sending it a message containing the local state s_p and waiting for a response with the remote state s_q. The *passive* thread waits for messages sent by an initiator and replies with the local state. The term push-pull refers to the fact that each information exchange is performed in a symmetric manner: both participants send and receive their states.

Even though the system is not synchronous, we find it convenient to describe the protocol execution in terms of consecutive real time intervals of length δ called *cycles* that are enumerated starting from some convenient point.

Method GETNEIGHBOR can be thought of as an underlying *service* to the aggregation protocol, which is normally (but not necessarily) implemented by sampling a locally available set of neighbors. In other words, an overlay network is applied to find communication partners. In Section 3.2 we will assume that GETNEIGHBOR returns a uniform random sample over the entire set of nodes. In Section 4.4 we revisit this service from a practical point of view, by looking at realistic implementations based on non-uniform or dynamically changing overlay topologies.

Method UPDATE computes a new local state based on the current local state and the remote state received during the information exchange. The output of UPDATE and the semantics of the node state depend on the specific aggregation function being implemented by the protocol. In this section, we limit the discussion to computing the average over the set of numbers distributed among the nodes. Additional functions (most of them derived from the averaging protocol) are described in Section 5.

In the case of computing the average, each node stores a single numeric value representing the current estimate of the final aggregation output which is the global average. Each node initializes the estimate with the local value it holds. Method UPDATE(s_p, s_q), where s_p and s_q are the estimates exchanged by p and q, returns $(s_p + s_q)/2$. After one exchange, the sum of the two local estimates remains unchanged since method UPDATE simply redistributes the initial sum equally among the two nodes. So, the operation does not change the global average but it decreases the variance over the set of all estimates in the system.

It is easy to see that the variance tends to zero, that is, the value at each node will converge to the true global average, as long as the network of nodes is not partitioned into disjoint clusters. To see this, one should consider the minimal value in the system. It can be proven that there is a positive probability in each cycle that either the number of instances of the minimal value decreases or the global minimum increases if there are different values from the minimal value (otherwise we are done because all values are equal). The idea is that if there is at least one different value, than at least one of the instances of the minimal values will have a neighbor with a different (thus larger) value and so it will have a positive probability to be matched with this neighbor.

In the following, we give basic theoretical results that characterize the speed of the convergence of the variance. We will show that each cycle results in a reduction of the variance by a constant factor, which provides exponential convergence. We will assume that no failures occur and that the starting point of the protocol is synchronized. Later in the paper, all of these assumptions will be relaxed.

3.2 Theoretical Analysis of Gossip-based Aggregation

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

We will treat the averaging protocol 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 $\mathbf{w}_0 = (w_{0,1} \dots w_{0,N})$. The elements of this vector correspond to the initial values at the nodes. We shall model this vector by assuming that $w_{0,1}, \dots, w_{0,N}$ are independent random variables with identical expected values and a finite variance.

The assumption of identical expected values is not as restrictive as it may seem. Too see this, observe that after any permutation of the initial values, the statistical behavior of the system

// vector $\mathbf{w}$ *is the input*
do N times
$(i, j) = \text{GETPAIR}()$
// perform elementary variance reduction step
$w_i = w_j = (w_i + w_j)/2$
return $\mathbf{w}$

Figure 2: Skeleton of global algorithm AVG used to model the distributed protocol of Figure 1.

remains unchanged since the protocol causes nodes to communicate in random order. This means that if we analyze the model in which we first apply a random permutation over the variables, we will obtain identical predictions for convergence. But if we apply a permutation, then we essentially transform the original vector of variables into another vector in which all variables have identical distribution, so the assumption of identical expected values holds.

In more detail, starting with random variables $w_{0,1}, \ldots, w_{0,N}$ with arbitrary expected values, after a random permutation, the new value at index i, denoted b_i, will have the distribution

$$P(b_i < x) = \frac{1}{N} \sum_{j=1}^{N} P(w_j < x) \tag{1}$$

since all variables can be shifted to any position with equal probability. That is, while obtaining an equivalent probability model as mentioned above, the distributions of random variables $b_0, \ldots, b_N$ are now identical. Note that the assumption of independence is technically violated (variables $b_0, \ldots, b_N$ are not independent), but in the case of large networks, the consequences will be insignificant.

When considering the network as a whole, one cycle of the averaging protocol can be seen as a variance reduction algorithm (let us call it AVG) which takes a vector $\mathbf{w}$ of length N as a parameter and produces a new vector $\mathbf{w}' = \text{AVG}(\mathbf{w})$ of the same length. In other words, AVG is a a single, central algorithm operating globally on the distributed state of the system, as opposed to the distributed protocol of Figure 1. This centralized view of the protocol serves to simplify our theoretical analysis of its behavior.

The consecutive cycles of the protocol result in a series of vectors $\mathbf{w}_1, \mathbf{w}_2, \ldots$, where $\mathbf{w}_{i+1} = \text{AVG}(\mathbf{w}_i)$. The elements of vector $\mathbf{w}_i$ are denoted as $\mathbf{w}_i = (w_{i,1} \ldots w_{i,N})$. Algorithm AVG is illustrated in Figure 2 and takes $\mathbf{w}$ as a parameter and modifies it in place producing a new vector. The behavior of our distributed gossip-based protocol can be reproduced by an appropriate implementation of GETPAIR. In addition, other implementations of GETPAIR are possible that do not necessarily map to any distributed protocol but are of theoretical interest. We will discuss some important special cases as part of our analysis.

We introduce the following empirical statistics for characterizing the state of the system in cycle i:

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

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

where $\overline{\mathbf{w}}_i$ is the target value of the protocol and σ_i^2 is a variance-like measure of homogeneity that characterizes the quality of local approximations. In other words, it expresses the deviation

of the local approximate values from the true aggregate value in the given cycle. In general, the smaller σ_i^2 is, the better the local approximations are, and if it is zero, then all nodes hold the perfect aggregate value.

The elementary variance reduction step (in which both selected elements are replaced by their average) is such that if we add the same constant C to the original values, then the end result will be the original average plus C. This means that for the purpose of this analysis, without loss of generality, we can assume that the common expected value of the elements of the initial vector $\mathbf{w}_0$ is zero (otherwise we can normalize with the common expected value in our equations without changing the behavior of the protocol in any way). The assumption serves to simplify our expressions. In particular, for any vector $\mathbf{w}$, if the elements of $\mathbf{w}$ are independent random variables with zero expected value, then

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

Furthermore, the elementary variance reduction step does not change the sum of the elements in the vector, so $\overline{\mathbf{w}}_i \equiv \overline{\mathbf{w}}_0$ for all cycles $i = 1, 2, \ldots$. This property is very important since it guarantees that the algorithm does not introduce any errors into the estimates for the average. This means that from now on we can focus on σ_i^2, because 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 $\overline{\mathbf{w}}_0$ 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 3.1 *Let $\mathbf{w}'$ be the vector that we obtain by replacing both w_i and w_j with $(w_i + w_j)/2$ in vector $\mathbf{w}$. If $\mathbf{w}$ contains uncorrelated random variables with expected value 0, then the expected value of the resulting variance reduction is given by*

$$E(\sigma_{\mathbf{w}}^2 - \sigma_{\mathbf{w}'}^2) = \frac{1}{2(N-1)}E(w_i^2) + \frac{1}{2(N-1)}E(w_j^2). \tag{5}$$

PROOF. Simple calculation using the fact that if w_i and w_j are uncorrelated, then

$$E(w_i w_j) = E(w_i)E(w_j) = 0. \tag{6}$$

□

In light of (4), an intuitive interpretation of this lemma is that after an elementary variance reduction step, both participating nodes will contribute only approximately half of their original contribution to the overall expected variance, provided they are uncorrelated. The assumption of uncorrelatedness is crucial to have this result. For example, in the extreme case of $w_i \equiv w_j$ (when this assumption is clearly violated) the lemma does not hold and the variance reduction is zero.

Keeping 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 $\mathbf{s}_0 \equiv (w_{0,1}^2 \ldots w_{0,N}^2)$ and $\mathbf{s}_i$ is produced in parallel with $\mathbf{w}_i$ using the same pair (i, j) returned by GETPAIR. In addition to performing the elementary averaging step on $\mathbf{w}_i$ (see Figure 2), we perform the step $s_i = s_j = (s_i + s_j)/4$ as well. This way, according to Lemma 3.1, $E(\overline{\mathbf{s}}_i)$ will emulate the evolution of $E(\sigma_i)$ with a high accuracy provided that each pair of values w_i and w_j selected by each call to GETPAIR are practically uncorrelated. Intuitively, this assumption can be expected to hold if the original values in $\mathbf{w}_0$ are uncorrelated and GETPAIR is "random enough" so as not to introduce significant correlations.

Working with $E(\overline{\mathbf{s}}_i)$ instead of $E(\sigma_i^2)$ is not only easier mathematically, but it also captures the dynamics of the system with 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 $\mathbf{w}_{i+1}$ from the input $\mathbf{w}_i$. In networking terms, ϕ_k denotes the number of state exchanges node k was involved in during cycle i.

Theorem 3.2 *If* 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 *have identical distributions,*

then we have

$$E(\overline{\mathbf{s}}_{i+1}) = E(2^{-\phi})E(\overline{\mathbf{s}}_i). \tag{7}$$

PROOF. We only give a sketch of the proof here. The basic idea is to think of $s_{i,k}$ as representing the quantity of some material. According to the definition of $s_{i,k}$, each time k is selected by GETPAIR we lose half of the material and the remaining material will be divided among the locations. Using assumption 2 of the theorem, we observe that it does not matter where a given piece of the original material ends up; it will have the same chance of losing its half as the proportion that stays at the original location. This 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 result. □

This Theorem will allow us to concentrate on the *convergence factor* that is defined as follows:

Definition 3.3 *The* convergence factor *between cycle i and $i+1$ is given by $E(\sigma_{i+1}^2)/E(\sigma_i^2)$.*

The convergence factor is an ideal measure to characterize the dynamics of the protocol because it captures the speed with which the local approximations converge towards the target value. Based on the reasoning we gave regarding $\mathbf{s}_i$, we expect that

$$E(\sigma_{i+1}^2) \approx E(2^{-\phi})E(\sigma_i^2) \tag{8}$$

will be true, if the correlation of the variables selected by GETPAIR is negligible. Note that this also means that, according to the theorem, the convergence factor depends only on the pair selection method. Most notably, it does not depend on network size, time, or the initial distribution of the values. Based on this observation, in the following we give explicit convergence factors through calculating $E(2^{-\phi})$ for specific implementations of GETPAIR and subsequently we verify the predictions of the theoretical model empirically.

3.2.1 Pair Selection: Perfect Matching

Let us begin by analyzing the optimal strategy for implementing GETPAIR. We will call this implementation GETPAIR_PM where PM stands for perfect matching. This implementation cannot be mapped to an efficient distributed 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 3.2 so it will serve as a reference for evaluating 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 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.

We can verify the assumptions of Theorem 3.2: (i) all nodes are selected the same constant number of times: exactly twice, 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 factor is given by

$$E(2^{-\phi}) = E(2^{-2}) = 1/4. \tag{9}$$

We now prove the optimality of this convergence factor under the assumptions of Theorem 3.2.

Lemma 3.4 *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. □

3.2.2 Pair Selection: Random Choice

Moving towards more practical implementations of GETPAIR, our next example is GETPAIR_RAND which simply returns a random pair of different nodes.

GETPAIR_RAND can easily be implemented as a distributed protocol, provided that GETNEIGHBOR returns a uniform random sample of the set of nodes. When iterating AVG, the waiting time between two consecutive selections 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 shall see, GETPAIR_RAND is not a very efficient pair selector. The purpose of discussing it is to illustrate the effect of relaxing the constraint of the original distributed protocol that requires each node to participate in at least one state exchange in each cycle.

Like for GETPAIR_PM, the assumptions of Theorem 3.2 hold: (i) for all nodes the same sampling probability applies at each step and (ii) all indeces have exactly the same probability to be selected after each elementary variance reduction step, irrespective of having been selected already or not.

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

$$P(\phi = j) = \frac{2^j}{j!} e^{-2}. \tag{10}$$

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

$$E(2^{-\phi}) = \sum_{j=0}^{\infty} 2^{-j} \frac{2^j}{j!} e^{-2} = e^{-2} \sum_{j=0}^{\infty} \frac{1}{j!} = e^{-2} e = e^{-1}. \tag{11}$$

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

3.2.3 Pair Selection: a Distributed Solution

Building on the results we have so far, it is possible to analyze our original protocol described in Figure 1.

In order to simulate this fully distributed version, the implementation of pair selection will return random pairs such that in each execution of AVG (that is, in each cycle), each node is guaranteed to be a member of at least one pair. This can be achieved by picking a random permutation of the nodes and pairing up each node in the permutation with another random node, thereby generating N pairs. We call this algorithm GETPAIR_DISTR. As we shall see, this protocol is not only implementable in a distributed way, its performance is also superior to that of GETPAIR_RAND although of course not matching GETPAIR_PM which is optimal.

It can be verified that this algorithm also satisfies the assumption of Theorem 3.2. Random variable ϕ can be approximated as $\phi = 1+\phi'$ where ϕ' has the Poisson distribution with parameter 1, that is, for $j > 0$

$$P(\phi = j) = P(\phi' = j - 1) = \frac{1}{(j-1)!}e^{-1}. \tag{12}$$

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

$$E(2^{-\phi}) = \sum_{j=1}^{\infty} 2^{-j}\frac{1}{(j-1)!}e^{-1} = \frac{1}{2e}\sum_{j=1}^{\infty}\frac{2^{-(j-1)}}{(j-1)!} = \frac{1}{2e}\sqrt{e} = \frac{1}{2\sqrt{e}}. \tag{13}$$

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

3.2.4 Empirical Results for Convergence of Aggregation

We ran AVG using GETPAIR_RAND and GETPAIR_DISTR for several network sizes and different initial distributions. For each parameter setting 50 independent experiments were performed.

Recall, that theory predicts that the average convergence factor is independent of the actual initial distribution of node values. To test this, we initialized the nodes in two different ways. In the *uniform* scenario, each node is assigned an initial value uniformly drawn from the same interval. In the *peak* scenario, one randomly selected node is assigned a non-zero value and the rest of the nodes are initialized to zero.

Note that in the case of the peak scenario, methods that approximate the average based on a small random sample (that is, statistical sampling methods) are useless: one has to know all the values to calculate the average. Also, for a fixed variance, we have the largest difference between any two values. In this sense this scenario represents a worst case scenario. Last but not least, the peak initialization has important practical applications as well as we discuss in Section 5.

The results are shown if Figures 3 and 4. Figure 3 confirms our prediction that convergence is independent of network size and that the observed convergence factors match theory with very high accuracy. Note that smaller convergence *factors* result in faster convergence.

The only difference between peak and uniform scenarios is that variance of the convergence factor is higher for the peak scenario. Note that our theoretical analysis does not tackle the question of convergence factor variance. We can see however that the average convergence factor is well predicted and after a few cycles the variance is decreased considerably.

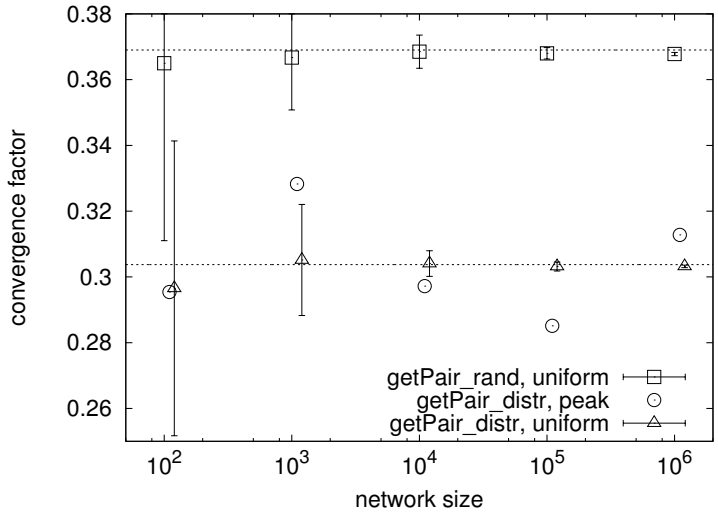


Figure 3: Convergence factor σ_1^2/σ_0^2 after one execution of AVG as a function of network size. For the peak distribution, error bars are omitted for clarity (but see Figure 4). Values are averages and standard deviations for 50 independent runs. Dotted lines correspond to the two theoretically predicted convergence factors: $e^{-1} \approx 0.368$ and $1/(2\sqrt{e}) \approx 0.303$.

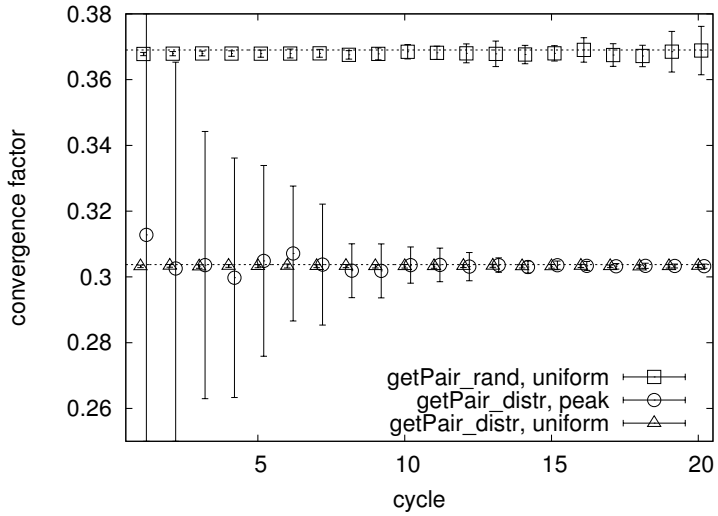


Figure 4: Convergence factor $\sigma_i^2/\sigma_{i-1}^2$ for network size $N = 10^6$ for different iterations of algorithm AVG. Values are averages and standard deviations for 50 independent runs. Dotted lines correspond to the two theoretically predicted convergence factors: $e^{-1} \approx 0.368$ and $1/(2\sqrt{e}) \approx 0.303$.

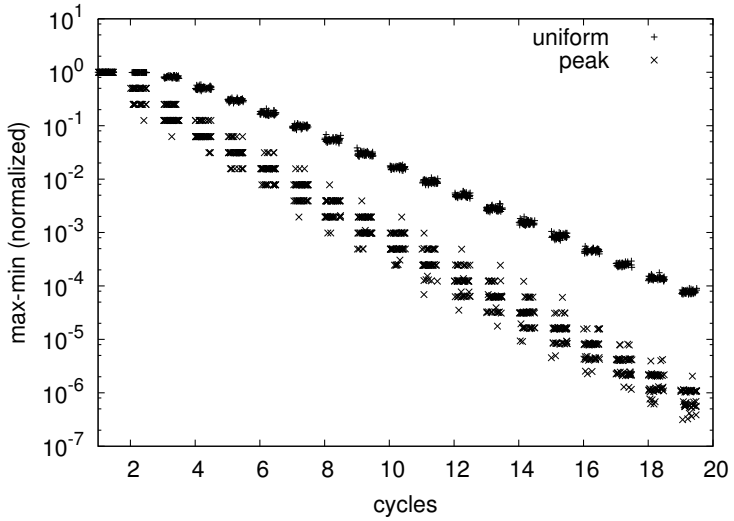


Figure 5: Normalized difference between the maximum and minimum estimates as a function of cycles with network size $N = 10^6$. All 50 experiments are plotted as a single point for each cycle with a small horizontal random translation.

Finally, to illustrate the "exponentially decreasing variance" result in a less abstract manner, Figure 5 shows the difference between the maximum and minimum estimates in the system for both the peak and uniform initialization scenarios. Note that although the expected variance $E(\sigma_i)$ decreases at the predicted rate, in the peak distribution scenario, the difference decreases faster. This effect is due to the highly skewed nature of the distribution of estimates in the peak scenario. In both cases, the difference between the maximum and minimum estimates decreases exponentially and after as few as 20 cycles the initial difference is reduced by several orders of magnitude. This means that after a small number of cycles all nodes, including the outliers, will possess very accurate estimates of the global average.

3.2.5 A Note on our Figures of Merit

Our approach for characterizing the quality of the approximations and convergence is based on the variance measure σ defined in (3) and the convergence factor, which describes the speed at which the expected value of σ decreases. To understand better what our results mean, it helps to compare it with other approaches to characterizing the quality of aggregation.

First of all, since we are dealing with a continuous process, there is no end result in a strict sense. Clearly, the figures of merit depend on how long we run the protocol. The variance measure σ_i characterizes the average *accuracy* of the approximates in the system in the give cycle. In our approach, apart from averaging the accuracy over the system, we also average it over different runs, that is, we consider $E(\sigma_i)$. This means that an individual node in a specific run can have rather different accuracy. In this paper we have not considered the distribution of the accuracy (only the mean accuracy as described above), which depends on the initial distribution of the values. However, Figure 5 suggests that our approach is robust to the initial distribution.

Another frequently used measure is *completeness* [8]. This measure is defined under the assumption that the aggregate is calculated based on the knowledge of a subset of the values (ideally, based on the entire set, but due to errors this cannot always be achieved). It gives the percentage of the values that were taken into account. In our protocol this measure is difficult to interpret because at all times a local approximate can be thought of as a weighted average of

the entire set of values. Ideally, all values should have equal weight in the approximations of the nodes (resulting in the global average value). To get a similar measure, one could characterize the distribution of weights as a function of time, to get a more fine-grained idea of the dynamics of the protocol.

4 A Practical Protocol for Gossip-based Aggregation

Building on the simple idea presented in the previous section, we now complete the details so as to obtain a full-fledged solution for gossip-based aggregation in practical settings.

4.1 Automatic Restarting

The generic protocol described so far is not adaptive, as the aggregation takes into account neither the dynamicity in the network nor the variability in values that are being aggregated. To provide up-to-date estimates, the protocol must be periodically *restarted*: at each node, the protocol is terminated and the current estimate is returned as the aggregation output; then, the current local values are used to re-initialize the estimates and aggregation starts again with these fresh initial values.

To implement termination, we adopt a very simple mechanism: each node executes the protocol for a predefined number of cycles, denoted as γ, depending on the required accuracy of the output and the convergence factor that can be achieved in the particular overlay topology adopted (see the convergence factor given in Section 3).

To implement restarting, we divide the protocol execution in consecutive *epochs* of length $\Delta = \gamma\delta$ (where δ is the cycle length) and start a new instance of the protocol in each epoch. We also assume that messages are tagged with an epoch identifier that will be applied by the synchronization mechanism as described below.

4.2 Coping with Churn

In a realistic scenario, nodes continuously join and leave the network, a phenomenon commonly called churn. When a new node joins the network, it contacts a node that is already participating in the protocol. Here, we assume the existence of an out-of-band mechanism to discover such a node, and the problem of initializing the neighbor set of the new node is discussed in Section 4.4.

The contacted node provides the new node with the next epoch identifier and the time until the start of the next epoch. Joining nodes are not allowed to participate in the current epoch; this is necessary to make sure that each epoch converges to the average that existed *at the start* of the epoch. Continuously adding new nodes would make it impossible to achieve convergence.

As for node crashes, when a node initiates an exchange, it sets a timeout period to detect the possible failure of the other node. If the timeout expires before the message is received, the exchange step is skipped. The effect of these missing exchanges due to real (or presumed) failures on the final average will be discussed in Section 7. Note that self-healing (removing failed nodes from the system) is taken care of by the NEWSCAST protocol, which we propose as the implementation of method GETNEIGHBOR (see Sections 4.4.2 and 7).

4.3 Synchronization

The protocol described so far is based on the assumption that cycles and epochs proceed in lock step at all nodes. In a large-scale distributed system, this assumption cannot be satisfied due to the unpredictability of message delays and the different drift rates of local clocks.

Given an epoch j, let T_j be the time interval from when the first node starts participating in epoch j to when the last node starts participating in the same epoch. In our protocol as it stands, the length of this interval would increase without bound given the different drift rates of local clocks and the fact that a new node joining the network obtains the next epoch identifier and start time from an existing node, incurring a message delay.

To avoid the above problem, we modify our protocol as follows. When a node participating in epoch i receives an exchange message tagged with epoch identifier j such that $j > i$, it stops participating in epoch i and instead starts participating in epoch j. This has the effect of propagating the larger epoch identifier (j) throughout the system in an epidemic broadcast fashion forcing all (slow) nodes to move up to the new epoch. In other words, the start of a new epoch acts as a synchronization point for the protocol execution forcing all nodes to follow the pace being set by the nodes that enter the new epoch first. Informally, knowing that push-pull epidemic broadcasts propagate super-exponentially [3] and assuming that each message arrives within the timeout used during all communications, we can obtain a logarithmic bound on T_j for each epoch j. More importantly, typically many nodes will start the new epoch independently with a very small difference in time, so this bound can be expected to be sufficiently small, which allows picking an epoch length, Δ, such that it is significantly larger that T_j. A more detailed analysis of this mechanism would be interesting but is out of the scope of the present discussion. The effect of lost messages (i.e., those that time out) however, is discussed later.

4.4 Importance of Overlay Network Topology for Aggregation

The theoretical results described in Section 3 are based on the assumption that the underlying overlay is "sufficiently random". More formally, this means that the neighbor selected by a node when initiating communication is a uniform random sample among its peers. Yet, our aggregation scheme can be applied to generic connected topologies, by selecting neighbors from the set of neighbors in the given overlay network. This section examines the effect of the overlay topology on the performance of aggregation.

All of the topologies we examine (with the exception of NEWSCAST) are static—the neighbor set of each node is fixed. While static topologies are unrealistic in the presence of churn, we still consider them due to their theoretical importance and the fact that our protocol can in fact be applied in static networks as well, although they are not the primary focus of the present discussion.

4.4.1 Static Topologies

All topologies considered have a regular degree of 20 neighbors, with the exception of the complete network (where each node knows every other node) and the Barabási-Albert network (where the degree distribution is a power-law). For the random network, the neighbor set of each node is filled with a random sample of the peers.

The Watts-Strogatz and scale-free topologies represent two classes of realistic *small-world* topologies that are often used to model different natural and artificial phenomena [1, 28]. The Watts-Strogatz model [29] is obtained from a regular ring lattice. The ring lattice is built by connecting the nodes in a ring and adding links to their nearest neighbors until the desired node degree is reached. Starting with this ring lattice, each edge is then randomly *rewired* with probability β. Rewiring an edge at node n means removing that edge and adding a new edge connecting n to another node picked at random. When $\beta = 0$, the ring lattice remains unchanged, while when $\beta = 1$, all edges are rewired, generating a random graph. For intermediate values of β, the structure of the graph lies between these two extreme cases: complete order and complete disorder.

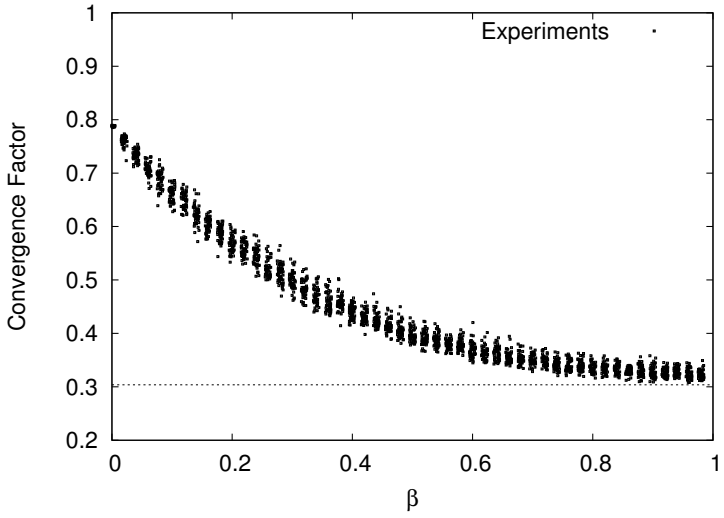


Figure 6: Convergence factor for Watts-Strogatz graphs as a function of parameter β. The dotted line corresponds to the theoretical convergence factor for peer selection through random choice: $1/(2\sqrt{e}) \approx 0.303$.

Figure 6 focuses on the Watts-Strogatz model showing the convergence factor as a function of β ranging from 0 to 1. Although there is no sharp phase transition, we observe that increased randomness results in a lower convergence factor (faster convergence).

Scale-free topologies form the other class of realistic small world topologies. In particular, the Web graph, Internet autonomous systems, and P2P networks such as Gnutella [23] have been shown to be instances of scale-free topologies. We have tested our protocol over scale-free graphs generated using the preferential attachment method of Barabási and Albert [1]. The basic idea of preferential attachment is that we build the graph by adding new nodes one-by-one, wiring the new node to an existing node already in the network. This existing contact node is picked randomly with a probability proportional to its degree (number of neighbors).

Let us compare all the topologies described above. Figure 7 illustrates the performance of aggregation for different topologies by plotting the average convergence factor over a period of 20 cycles, for network sizes ranging from 10^2 to 10^6 nodes. Figure 8 provides additional details. Here, the network size is fixed at 10^5 nodes. Instead of displaying the average convergence factor, the curves illustrate the actual variance reduction (values are normalized so that the initial variance for all cases is 1) for the same set of topologies. We can conclude that performance is independent of network size for all topologies, while it is highly sensitive to the topology itself. Furthermore, the convergence factor is constant as a function of time (cycle), that is, the variance is decreasing exponentially, with non-random topologies being the only exceptions.

4.4.2 Dynamic Topologies

From the above results, it is clear that aggregation convergence benefits from increased randomness of the underlying overlay network topology. Furthermore, in dynamic systems, there must be mechanisms in place that preserve this property over time. To achieve this goal, we propose to use NEWSCAST [13, 10], which is a decentralized membership protocol based on a gossip-based scheme similar to the one described in Figure 1.

In NEWSCAST, the overlay is generated by a continuous exchange of neighbor sets, where each

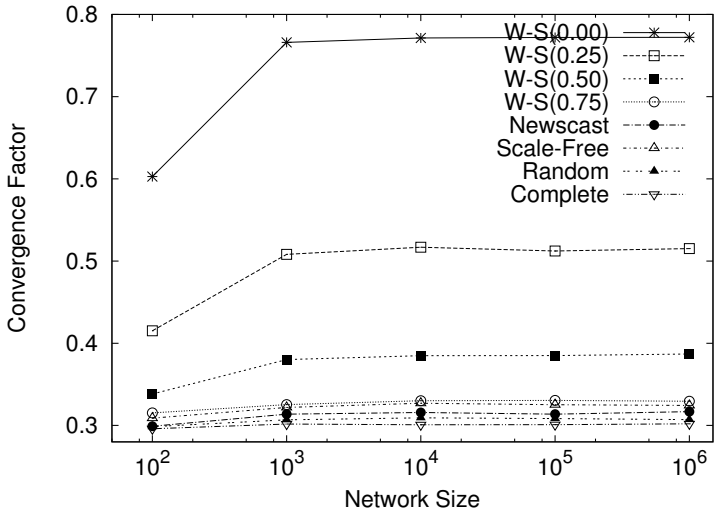


Figure 7: Average convergence factor computed over a period of 20 cycles in networks of varying size. Each curve corresponds to a different topology where W-S(β) stands for the Watts-Strogatz model with parameter β.

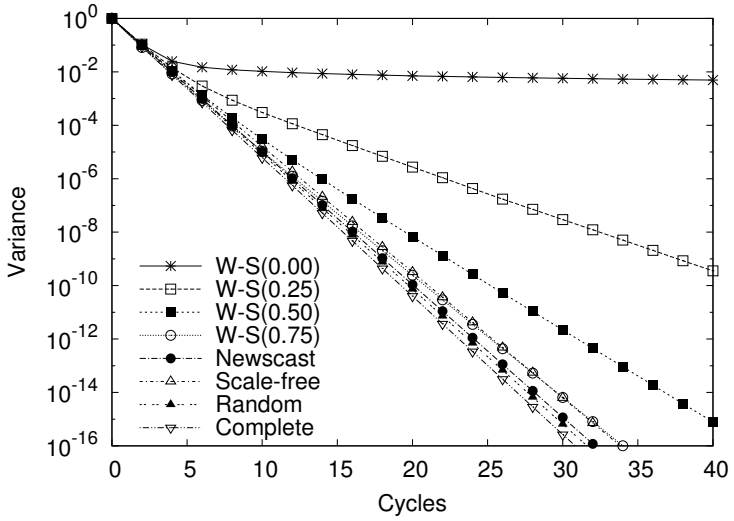


Figure 8: Variance reduction for a network of 10^5 nodes. Results are normalized so that all experiments result in unit variance initially. Each curve corresponds to a different topology where W-S(β) stands for the Watts-Strogatz model with parameter β.

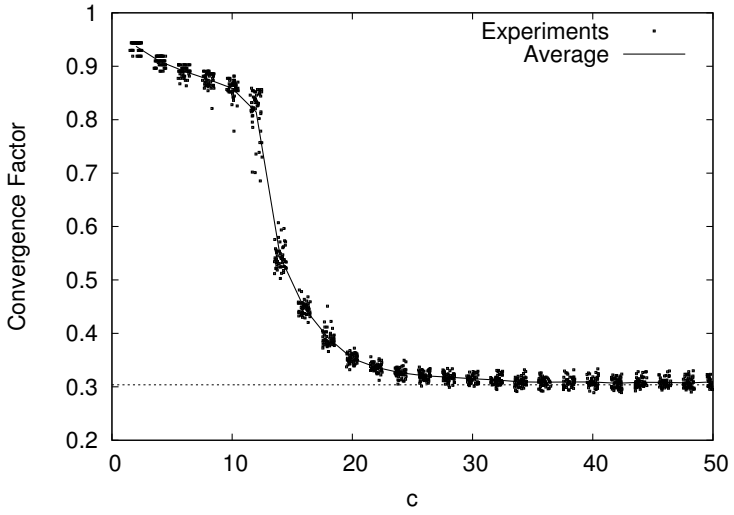


Figure 9: Convergence factor for NEWSCAST graphs as a function of parameter c. The dotted line corresponds to the theoretical convergence factor for peer selection through random choice: $1/(2\sqrt{e}) \approx 0.303$.

element consists of a node identifier and a timestamp. These sets have a fixed size, which will be denoted by c. After an exchange, participating nodes update their neighbor sets by selecting the c node identifiers (from the union of the two sets) that have the freshest timestamps. Nodes belonging to the network continously inject their identifiers in the network with the current timestamp, so old identifiers are gradually removed from the system and are replaced by newer information. This feature allows the protocol to "repair" the overlay topology by forgetting information about crashed neighbors, which by definition cannot inject their identifiers.

The resulting topology has a very low diameter (each node is reachable from any other node through very few links) [13, 10]. Figure 9 shows the performance of aggregation over a NEWSCAST network of 10^5 nodes, with c varying between 2 and 50. From these experimental results, choosing $c = 30$ is already sufficient to obtain fast convergence for aggregation. Furthermore, this same value for c is sufficient for very stable and robust connectivity [13, 10]. Figures 7 and 8 provide additional evidence that applying NEWSCAST with $c = 30$ already results in performance very similar to that of a random network.

4.5 Cost Analysis

Both the communication cost and time complexity of our scheme follow from properties of the aggregation protocol and are inversely related. The cycle length, δ defines the time complexity of convergence. Choosing a short δ will result in proportionally faster convergence but higher communication costs per unit time. It is possible to show that if the overlay is sufficiently random, the number of exchanges for each node in δ time units can be described by the random variable $1 + \phi$ where ϕ has a Poisson distribution with parameter 1. Thus, on the average, there are two exchanges per node (one initiated by the node and the other one coming from another node), with a very low variance. Based on this distribution, parameter δ must be selected to guarantee that, with very high probability, each node will be able to complete the expected number of exchanges before the next cycle starts. Failing to satisfy this requirement results in a violation of our theoretical assumptions. Similarly, parameter γ must be chosen appropriately, based on the desired accuracy

of the estimate and the convergence factor ρ characterizing the overlay network. After γ cycles, we have $E(\sigma_\gamma^2)/E(\sigma_0^2) = \rho^\gamma$ where $E(\sigma_0^2)$ is the expected variance of the initial values. If ϵ is the desired accuracy of the final estimate, then $\gamma \geq \log_\rho \epsilon$. Note that ρ is independent of N, so the time complexity of reaching a given precision is $O(1)$.

5 Aggregation Beyond Averaging

In this section we give several examples of gossip-based aggregation protocols to calculate different aggregates. With the exception of minimum and maximum calculation, they are all built on averaging. We also briefly discuss the question of dynamic queries.

5.1 Examples of Supported Aggregates

5.1.1 Minimum and maximum

To obtain the maximum or minimum value among the values maintained by all nodes, method UPDATE(a, b) of the generic scheme of Figure 1 must return $\max(a, b)$ or $\min(a, b)$, respectively. In this case, the global maximum or minimum value will be effectively broadcast like an epidemic. Well-known results about epidemic broadcasting [3] are applicable.

5.1.2 Generalized means

We formulate the general mean of a vector of elements $\mathbf{w} = (w_0, \ldots, w_N)$ as

$$f(\mathbf{w}) = g^{-1}\left(\frac{\sum_{i=0}^{N} g(w_i)}{N}\right) \tag{14}$$

where function f is the mean function and function g is an appropriately chosen local function to generate the mean. Well known examples include $g(x) = x$ which results in the average, $g(x) = x^n$ which defines the nth power mean (with $n = -1$ being the harmonic mean, $n = 2$ the quadratic mean, etc.) and $g(x) = \ln x$ resulting in the geometric mean (nth root of the product). To compute the above general mean, UPDATE(a, b) returns $g^{-1}[(g(a) + g(b))/2]$. After each exchange, the value of f remains unchanged but the variance over the set of values decreases so that the local estimates converge toward the general mean.

5.1.3 Variance and other moments

In order to compute the nth *raw* moment which is the average of the nth power of the original values, $\overline{w^n}$, we need to initialize the estimates with the nth power of the local value at each node and simply calculate the average. To calculate the nth central moment, given by $\overline{(w - \overline{w})^n}$, we can calculate all the raw moments in parallel up to the nth and combine them appropriately, or we can proceed in two sequential steps first calculating the average and then the appropriate central moment. For example, the variance, which is the 2nd central moment, can be approximated as $\overline{w^2} - \overline{w}^2$.

5.1.4 Counting

We base counting on the observation that if the initial distribution of local values is such that exactly one node has the value 1 and all the others have 0, then the global average is exactly $1/N$ and thus the network size, N, can be easily deduced from it. We will use this protocol, which we call COUNT, in our experiments.

Using a probabilistic approach, we suggest a simple and robust implementation of this scheme without any need for leader election: we allow multiple nodes to randomly start concurrent instances of the averaging protocol, as follows. 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). Each node maintains a map M associating a leader identifier with an average estimate. When nodes n_i and n_j maintaining the maps M_i and M_j perform an exchange, the new map M (to be installed at both nodes) is obtained by merging M_i and M_j in the following way:

$$\begin{aligned} M \quad = \quad & \{(l, e/2) \mid e = M_i(l) \in M_i \wedge l \notin D(M_j)\} \cup \\ & \{(l, e/2) \mid e = M_j(l) \in M_j \wedge l \notin D(M_i)\} \cup \\ & \{(l, (e_i + e_j)/2 \mid e_i = M_i(l) \wedge e_j = M_j(l)\}, \end{aligned}$$

where $D(M)$ corresponds to the domain (key set) of map M and e_i is the current estimate of node n_i. In other words, if the average estimate for a certain leader is known to only one node out of the two nodes that participate in an exchange, the other node is considered to have an estimate of 0.

Maps are initialized in the following way: if node n_l is a leader, the map is equal to $\{(l, 1)\}$, otherwise the map is empty. All nodes participate in the protocol described in the previous section. In other words, even nodes with an empty map perform random exchanges. Otherwise, an approach where only nodes with a non-empty set perform exchanges would be less effective in the initial phase while few nodes have non-empty maps.

Clearly, the number of concurrent protocols in execution must be bounded, to limit the communication costs involved. A simple mechanism that we adopt is the following. At the beginning of each epoch, each node may become leader of a run of the aggregation protocol with probability P_{lead}. At each epoch, we set $P_{\text{lead}} = C/\hat{N}$, where C is the desired number of concurrent runs and $\hat{N}$ is the estimate obtained in the previous epoch. If the systems size does not change dramatically within one epoch then this solution ensures that the number of concurrently running protocols will be approximately Poisson distributed with the parameter C.

5.1.5 Sums and products

Two concurrent aggregation protocols are run, one to estimate the size of the network, the other to estimate the average or the geometric mean, respectively. The size and the means together can be used to compute the sum or the product of the initial local values.

5.1.6 Rank statistics

Although the examples presented above are quite general, certain statistics appear to be difficult to calculate in this framework. Statistics that have a definition based on the index of values in a global ordering (often called *rank statistics*) fall into this category. While certain rank statistics like the minimum and maximum (see above) can be calculated easily, others, including the median, are more difficult. Extending our results in this direction is an active area of our research [19].

5.2 Dynamic Queries

Although in this paper we target applications where the same query is calculated continuously and proactively in a highly dynamic large network, having a fixed query is not an inherent limitation of the approach. The aggregate value being calculated is defined by method UPDATE and the semantics of the state of the nodes (the parameters of method UPDATE). These components can

be changed throughout the system at any time, using for example an extension of the restarting technique discussed in Section 4, where in a new epoch not only the start of the new epoch is being propagated through gossip but a new query as well.

Typically, our protocol will provide aggregation service for an application. The exact details of the implementation of dynamic queries (if necessary) will depend on the specific environment, taking into account efficiency and performance constraints and possible sources of new queries.

6 Theoretical Results for Benign Failures

6.1 Crashing Nodes

The result on convergence discussed in Section 3 is based on the assumption that the overlay network is static and that nodes do not crash. When in fact in a dynamic environment, there may be significant churn with nodes coming and going continuously. In this section we present results on the sensitivity of our protocols to dynamism of the environment.

Our failure model is the following. Before each cycle, a fixed proportion, say P_f, of the nodes crash.[1] Given N^* nodes initially, $P_f N^*$ nodes are removed (without replacement) as the ones that actually crash. We assume crashed nodes do not recover. Note that considering crashes only at the beginning of cycles corresponds to a worst-case scenario since the crashed nodes render their local values inaccessible when the variance among the local values is at its maximum. In other words, the more times a node communicates with other nodes, the better it approximates the correct global average (on average), so removing it at a latter stage does not disturb the end result as much as removing it at the beginning. Also recall that we are interested in the average at the beginning of the current epoch as opposed to the real-time average (see Section 4.1).

Let us begin with some simple observations. Using the notations in (3) in our failure model the expected value of $\overline{\mathbf{w}}_i$ and σ_i^2 will stay the same independently of P_f since the model is completely symmetric. The convergence factor also remains the same since it does not rely on any particular network size. So the only interesting measure is the variance of $\overline{\mathbf{w}}_i$, which characterizes the expected error of the approximation of the average. We will describe the variance of $\overline{\mathbf{w}}_i$ as a function of P_f.

Theorem 6.1 *Let us assume that $E(\sigma_{i+1}^2) = \rho E(\sigma_i^2)$ and that the values $w_{i,1}, \ldots, w_{i_N}$ are pairwise uncorrelated for $i = 0, 1, \ldots$ Then $\overline{\mathbf{w}}_i$ has a variance*

$$Var(\overline{\mathbf{w}}_i) = \frac{P_f}{N(1-P_f)} E(\sigma_0^2) \frac{1 - \left(\frac{\rho}{1-P_f}\right)^i}{1 - \frac{\rho}{1-P_f}}. \tag{15}$$

PROOF. Let us take the decomposition $\overline{\mathbf{w}}_{i+1} = \overline{\mathbf{w}}_i + d_i$. Random variable d_i is independent of $\overline{\mathbf{w}}_i$ so

$$\text{Var}(\overline{\mathbf{w}}_{i+1}) = \text{Var}(\overline{\mathbf{w}}_i) + \text{Var}(d_i). \tag{16}$$

This allows us to consider only $\text{Var}(d_i)$ as a function of failures. Note that $E(d_i) = 0$ since $E(\overline{\mathbf{w}}_i) = E(\overline{\mathbf{w}}_{i+1})$. Then, using the assumptions of the theorem and the fact that $E(d_i) = 0$ it can be proven that

$$\text{Var}(d_i) = E((\overline{\mathbf{w}}_i - \overline{\mathbf{w}}_{i+1})^2) = \frac{P_f}{N_i(1-P_f)} E(\sigma_i^2) = \frac{P_f}{1-P_f} E(\sigma_0^2) \frac{\rho^i}{N(1-P_f)^i}. \tag{17}$$

Now, from (16) we see that $\text{Var}(\overline{\mathbf{w}}_i) = \sum_{j=0}^{i-1} \text{Var}(d_j)$ which gives the desired formula when substituting (17). □

[1] Recall that we do not distinguish between nodes leaving the network voluntarily and those that crash.

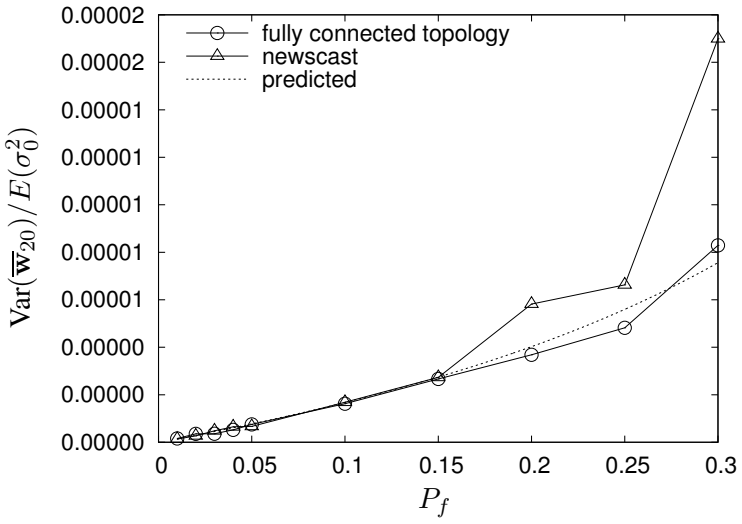


Figure 10: Effects of node crashes on the variance of the average estimates at cycle 20.

The results of simulations with $N = 10^5$ to validate this analysis are shown in Figure 10. For each value of P_f, the empirical data is based on 100 independent experiments whereas the prediction is obtained from (15) with $\rho = 1/(2\sqrt{e})$. The empirical data fits the prediction nicely. Note that the largest value of P_f examined was 0.3 which means that in each cycle almost one third of the nodes is removed. This already represents an extremely severe scenario. See also Section 7.1, where we present additional experimental analysis using NEWSCAST.

If $\rho > 1 - P_f$ then the variance is not bounded, it grows with the cycle index, otherwise it is bounded. Also note that increasing network size decreases the variance of the approximation $\overline{\mathbf{w}}_i$. This is good news for scalability, as the larger the network, the more stable the approximation becomes.

6.2 Link Failures

In a realistic system, links fail in addition to nodes crashing. This represents another important source of error, although we note that from our point of view node crashes are more important because we model leaves as crashes, so in the presence of churn crash events dominate all other types of failure.

Let us adopt a failure model in which an exchange is performed only with probability $1 - P_d$, that is, each link between any pair of nodes is down with probability P_d. This model is adequate because we focus on short term link failures. For long term failures it is not sufficient to model failure as a probability, and long term failures can hardly be modeled as independent either. Besides, long term link failure in an *overlay* network means long term partitioning in the underlying physical network (because if the physical network was connected, normally the routing service could still function), and thus the overlay network is also partitioned. In such a partitioned topology our protocol will simply calculate an aggregate value local to each partitioned cluster.

In Section 3.2 it was proven that $\rho = 1/e$ (where ρ is the convergence factor) if we assume that during a cycle for each particular variance reduction step, each pair of nodes has an equal probability to perform that particular variance reduction step. For the protocol described in Figure 1 we have proven that $\rho = 1/(2\sqrt{e})$. For this protocol the uniform randomness assumption does not hold since the protocol guarantees that each node participates in at least one variance re-

duction step—the one initiated actively by the node. In the random model however, it is possible for example that a node does not participate in a given cycle at all.

Consider that a system model with $P_d > 0$ is very similar to a model in which $P_d = 0$ but which is "slower" (fewer pairwise exchanges are performed in a unit time interval). In the limit case when P_d is close to 1, the uniform randomness assumption described above (when $\rho = 1/e$) is fulfilled with high accuracy.

This motivates our conclusion that the performance can be bounded from below by the model where $P_d = 0$, and $\rho = 1/e$ instead of $1/(2\sqrt{e})$, and which is $1/(1 - P_d)$ times slower than the original system in terms of wall clock time. That is, the upper bound on the convergence factor can be expressed as

$$\rho_d = (\frac{1}{e})^{1-P_d} = e^{P_d - 1} \tag{18}$$

which gives $\rho_d^{1/(1-P_f)} = 1/e$. Since the factor $1/e$ is not significantly worse than $1/(2\sqrt{e})$, we can conclude that practically only a proportional slowdown of the system is observed. In other words, link failures do not result in any loss of approximation quality or increased unreliability.

6.3 Conclusions

We have examined two sources of random failures: node crashes and link failures. In the case of node crashes, an exact relationship was given between the proportion of failing nodes and the expected loss in accuracy of the average estimation. We have seen that the protocol can tolerate relatively large amounts of node crashes and still provide reasonable estimates. We have also shown that performance degrades gracefully with increasing link failure probability.

7 Simulation Results for Benign Failures

To complement the theoretical analysis, we have performed numerous experiments based on simulation. In all experiments, we used NEWSCAST as the underlying overlay network to implement function GETNEIGHBOR in Figure 1. As a result, we need no unrealistic assumptions about the amount of information available at the nodes locally.

Furthermore, all our experiments were performed with the COUNT protocol since it is the aggregation example that is most sensitive to failures (both node crashes and message omissions) and thus represents a worst-case. During the first few cycles of an epoch when only a few nodes have a local estimate other than 0, their removal from the network due to failures can cause the final result of COUNT to diverge significantly from the actual network size.

All of experimental results were obtained through PEERSIM, a simulator developed by us and optimized for aggregation protocols [12, 22]. Unless stated otherwise, all simulations are performed on networks composed of 10^5 nodes. We do not present results for different network sizes since they display similar trends (as predicted by our theoretical results and confirmed by Figure 7).

The size of the neighbor sets maintained and exchanged by the NEWSCAST protocol is set to 30. As discussed in Section 4.4, this value is large enough to result in convergence factors similar to those of random networks; furthermore, as our experiments confirm, the overlay network maintains this property also in the face of the node crash scenarios we examined. Unless explicitly stated, the size estimates and the convergence factor plotted in the figures are those obtained at the end of a single epoch of 30 cycles. In all figures, 50 individual experiments were performed for all parameter settings. When the result of each experiment is shown in a figure (e.g., as a dot) to illustrate the entire distribution, the x-coordinates are shifted by a small random value so as to separate results having similar y-coordinates.

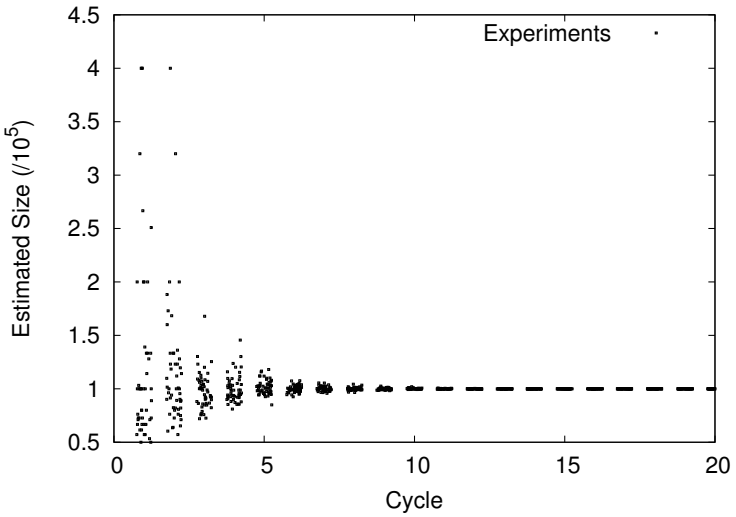


Figure 11: Network size estimation with protocol COUNT where 50% of the nodes crash suddenly. The x-axis represents the cycle of an epoch at which the "sudden death" occurs.

7.1 Node Crashes

The crash of a node may have several possible effects. If the crashed node had a value smaller than the actual global average, the estimated average (which should be $1/N$) will increase and consequently the reported size of the network N will decrease. If the crashed node has a value larger than the average, the estimated average will decrease and consequently the reported size of the network N will increase.

The effects of a crash are potentially more damaging in the latter case. The larger the removed value, the larger the estimated size. At the beginning of an epoch, relatively large values are present, obtained from the first exchanges originated by the initial value 1. These observations are confirmed by Figure 11, that shows the effect of the "sudden death" of 50% of the nodes in a network of 10^5 nodes at different cycles of an epoch. Note that in the first cycles, the effect of crashing may be very harsh: the estimate can even become infinite (not shown in the figure), if all nodes having a value different from 0 crash. However, around the tenth cycle the variance is already so small that the damaging effect of node crashes is practically negligible.

A more realistic scenario is a network subject to churn. Figure 12 illustrates the behavior of aggregation in such a network. Churn is modeled by removing a number of nodes from the network and substituting them with new nodes at each cycle. According to the protocol, the new nodes do not participate in the ongoing approximation epoch. However this scenario is not fully equivalent to a continuous node crashing scenario because these new nodes do participate in the NEWSCAST network and so they are contacted by participating nodes. These contacts are refused by the new nodes which results in an additional effect similar to link failure.

The size of the network is constant, while its composition is dynamic. The plotted dots correspond to the average estimate computed over all nodes that still participate in the protocol at the end of a single epoch (30 cycles), that is, that were originally part of the system at the start of the epoch. Note that although the average estimate is plotted over all nodes, in cycle 30 the estimates are practically identical as Figure 8 confirms. Also note that 2,500 nodes crashing in a cycle means that 75% of the nodes ($(30 \times 2500)/10^5$) are substituted during the epoch, leaving 25% of the nodes that make it until the end of the epoch.

The figure demonstrates that (even when a large number of nodes are substituted during an

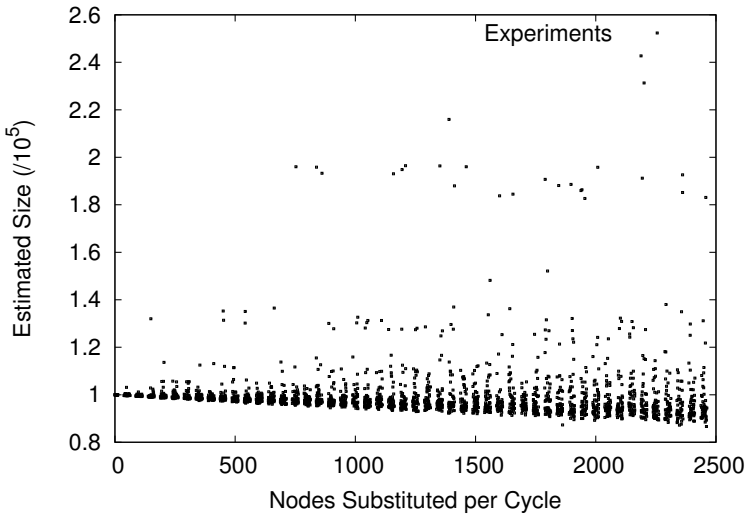


Figure 12: Network size estimation with protocol COUNT in a network of constant size subject to churn. The x-axis is the churn rate which corresponds to the number of nodes that crash at each cycle and are substituted by the same number of new nodes.

epoch) most of the estimates are included in a reasonable range. This is consistent with the theoretical result discussed in Section 6.1, although in this case we have an additional source of error: nodes are not only removed but replaced by new nodes. While the new nodes do not participate in the epoch, they result in an effect similar to link failure, as new nodes will refuse all connections that belong to the currently running epoch. However, the variance of the estimate continues to be described by the results in Section 6.1 because according to Sections 6.2 and 7.2, link failures do not change the estimate, only slows down convergence. Since an epoch lasts 30 cycles, this time is enough for convergence even beside the highest fluctuation rate. See also Figure 10 for the variance of the estimates plotted against the theoretical prediction.

The above experiment can be considered as a worst case analysis since the level of churn was much higher than it could be expected in a realistic scenario, considering that an epoch lasts for a relatively short time. We have repeated our experiments on the well-known Gnutella trace described in [24] to validate our results on a more realistic churn scenario as well. Figure 13 illustrates the simulation results. Only a short time window is shown (where the churn rate is particularly variable) to illustrate the accuracy of the approach better. We can observe that the approximation is accurate (with a one epoch delay), and the standard deviation is low as well. In this particular trace, during one epoch approximately 5% of the nodes are replaced. This is a relatively low rate and as we have seen earlier, the protocol can withstand much higher churn rates. Noted that the figure illustrates only the fluctuations in the network size as a result of churn and not the actual churn rate itself.

7.2 Link Failures and Message Omissions

Figure 14 shows the convergence factor of COUNT in the presence of link failures. As discussed earlier, in this case the only effect is a proportionally slower convergence. The theoretically predicted upper bound of the convergence factor (see (18)) indeed bounds the average convergence factor, and—as predicted—it is more accurate for higher values of P_d.

Apart from link failures that interrupt communication between two nodes in a symmetric way,

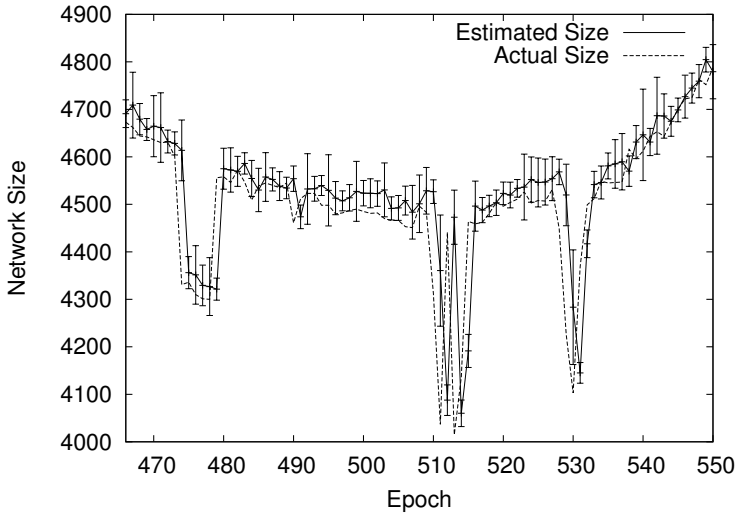


Figure 13: Network size estimation with protocol COUNT in the presence of churn according to a Gnutella trace [24]. 50 experiments were run to calculate statistics (mean and standard deviation), each epoch consisted of 30 cycles, each cycle lasted for 10 seconds.

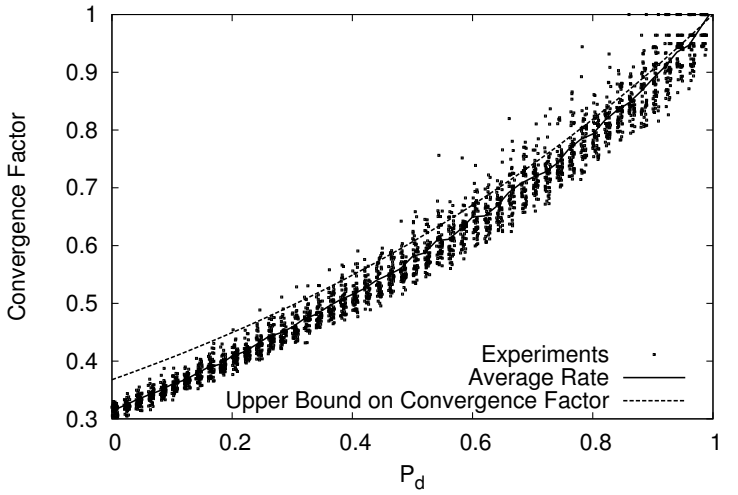


Figure 14: Convergence factor of protocol COUNT as a function of link failure probability.

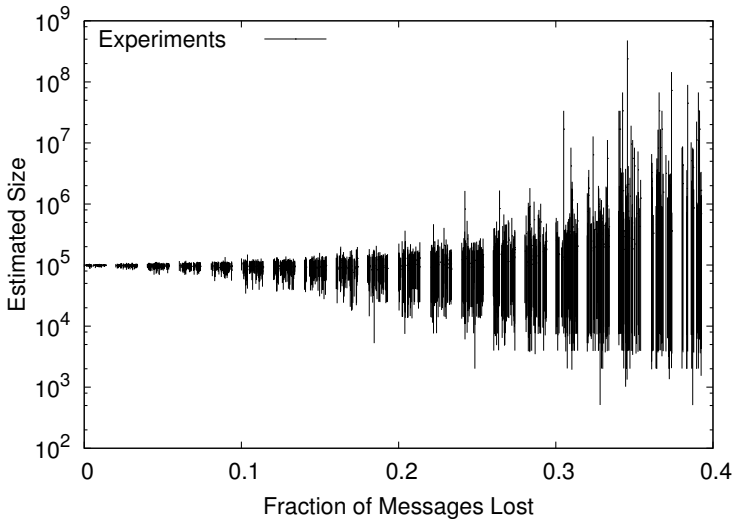


Figure 15: Network size estimation with protocol COUNT as a function of lost messages. The length of the bars illustrate the distance between the minimal and maximal estimated size over the set of nodes within a single experiment.

it is also possible that single messages are lost. If the message sent to initiate an exchange is lost, the final effect is the same as with link failure: the entire exchange is lost, and the convergence process is just slowed down. But if the message lost is the response to an initiated exchange, the global average may change (either increasing or decreasing, depending on the value contained in the message).

The effect of message omissions is illustrated in Figure 15. The given percentage of all messages (initiated or response) was dropped. For each experiment, both the maximum and the minimum estimates over the nodes in the network are shown, represented by the ends of the bars. As can be seen, when a small percentage of messages are lost, estimations of reasonable quality can be obtained. Unfortunately, when the number of messages lost is higher, the results provided by aggregation can be larger or smaller by several orders of magnitude. In this case, however, it is possible to improve the quality of estimations considerably by running multiple concurrent instances of the protocol, as explained in the next section.

7.3 Increasing Robustness Using Multiple Instances of Aggregation

To reduce the impact of "unlucky" runs of the aggregation protocol that generate incorrect estimates due to failures, one possibility is to run multiple concurrent instances of the aggregation protocol. To test this solution, we have simulated a number t of concurrent instances of the COUNT protocol, with t varying from 1 to 50. At each node, the t estimates that are obtained at the end of each epoch are ordered. Subsequently, the $\lfloor t/3 \rfloor$ lowest estimates and the $\lfloor t/3 \rfloor$ highest estimates are discarded, and the reported estimate is given by the average of the remaining results.

Figure 16 shows the results obtained by applying this technique in a system where 1000 nodes per cycle are substituted with new nodes, while Figure 17 shows the results in a system where 20% of the messages are lost. Recall that even though in the node crashing scenario the number of nodes participating in the epoch decreases, the correct estimation is 10^5 as the protocol reports network size at the *beginning* of the epoch.

The results are quite encouraging; by maintaining and exchanging just 20 numerical values

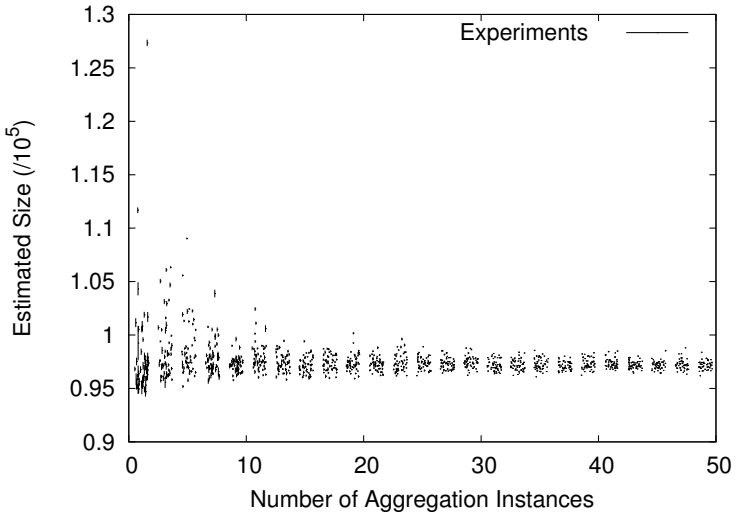


Figure 16: Network size estimation with multiple instances of protocol COUNT. 1000 nodes crash at the beginning of each cycle. The length of the bars correspond to the distance between the minimal and maximal estimates over the set of all nodes within a single experiment.

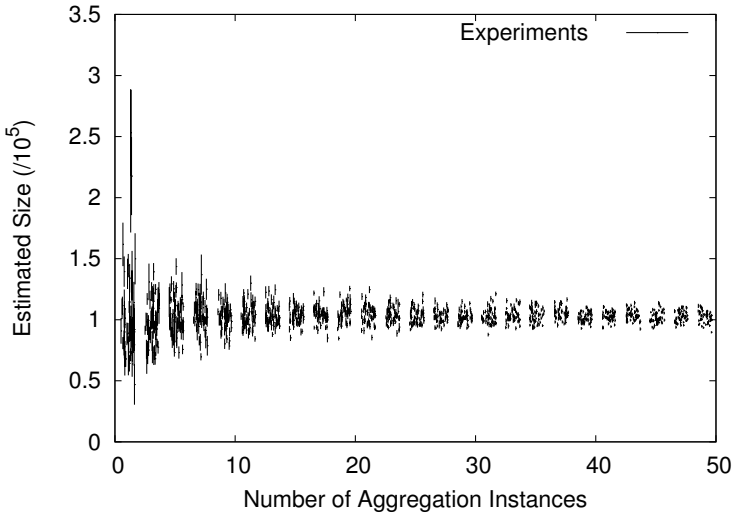


Figure 17: Network size estimation with protocol COUNT as a function of concurrent protocol instances. 20% of messages are lost. The length of the bars correspond to the distance between the minimal and maximal estimates over the set of all nodes within a single experiment.

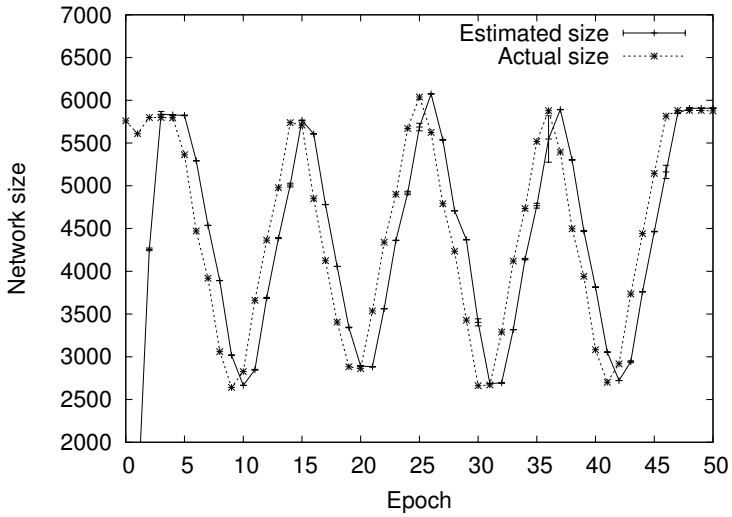


Figure 18: The estimated size (as provided by COUNT) and the actual size of a network oscillating between 2500 and 6000 nodes (approximately). Standard deviation of estimated size is displayed using vertical bars.

(resulting in messages of still only a few hundreds of bytes), the accuracy that may be obtained is very high, especially considering the hostility of the scenarios tested. It can also be observed that the estimate is very consistent over the nodes (the bars are short) in the crash scenario (as predicted by our theoretical results), and using multiple instances the variance of the estimate over the nodes decreases significantly even in the message omission scenario, so the estimate is sufficiently representative at every single node.

8 Experimental Results on PlanetLab

In order to validate our analytical and simulation results, we implemented the COUNT protocol and deployed it on PlanetLab [2]. PlanetLab is an open, globally distributed platform for developing, deploying and accessing planetary-scale network services. At the time of this writing, more than 170 academic institutions and industrial research labs are members of the PlanetLab consortium, providing more than 400 nodes for experimentation.

A summary of the experimental results obtained on PlanetLab is illustrated in Figure 18. During the experiment, 300 machines belonging to the PlanetLab testbed were used. Each machine was running up to 20 virtual nodes, each participating as a distinct entity. In other words, the maximum size of our emulated network was 6000 virtual nodes, distributed over five continents. The size of the network was made to oscillate between 2500 and 6000 nodes during the experiment. Virtual nodes were removed and added using a central scheduler that randomly picked nodes from the network to produce the oscillation effect shown in the figure. The number of concurrent protocol instances was 20 (see Section 7.3), and parameter c of NEWSCAST was $c = 30$. The length of a cycle is 5 seconds, while the number of cycles in an epoch is 30 (that is, the length of an epoch is approximately 2.5 minutes). Several experiments were run, all of them starting at 02:00 Central European Time during workdays. All of them produced results similar to those shown in the figure. The communication mechanism of our implementation is based on UDP. This choice is motivated by the fact that in a network based on NEWSCAST, interactions between nodes are short-lived, so establishing a TCP connection is relatively expensive. On the other

hand, the protocol can tolerate message omissions. The observed message omission rate during our experiments varied between 3% and 8%.

The figure shows two curves, one representing the real size of the network at the beginning of a given epoch, and the other representing the estimated size, averaged over all nodes in the network. The (very small) standard deviation of the estimates over all nodes is also illustrated using vertical bars. These experiments further confirm the validity and practicality of our mechanisms.

9 Related Work

Since our work overlaps with a large number of fields, including gossip-based and epidemic protocols, load balancing, aggregation and network size estimation (in both overlay and wireless ad hoc networks), we restrict our discussion to the most relevant publications from each area.

Protocols based on epidemic and gossiping metaphors have found numerous practical applications. Examples include database replication [3] and failure detection [27]. A recently completed survey by Eugster et al. provides an excellent introduction to the area [5]. Note that our approach applies gossiping only as the communication model (periodic information exchange with random peers). Strictly speaking, nothing is "gossiped", the dynamics of the system is closer to a diffusion process. This is why, for example, theoretical results on epidemic spreading are not directly relevant here.

The load balancing protocol presented in [7] builds on the idea of generating a matching in the network topology and balancing load along the edges in the matching. Although the basic idea is similar, our work assumes a random overlay network (that we provide using NEWSCAST) and does not require the communications to take place in a matching in this network. Recall however that we have shown that the matching is the optimal case for our protocol; fortunately random pair selection has similar performance as well.

There are a number of general purpose systems for aggregation that offer a database abstraction (supporting queries about the state of the system) and that are based on structured (typically hierarchical) topologies. Perhaps the best-known example of this approach is Astrolabe [26], and more recently, SDIMS [30]. In these systems a hierarchical architecture is deployed which reduces the cost of finding the aggregates and enables the execution of complex database queries. However, maintenance of the hierarchical topology introduces additional overhead, which can be significant if the environment is very dynamic. Our gossip-based aggregation protocol is substantially different. Although the class of aggregates that it can compute is fairly general, and dynamic queries can also be implemented, it is not a general purpose system: it is extremely simple, lightweight, and targeted for unstructured, highly dynamic environments. Furthermore, our protocol is proactive: the updated results of aggregation are known to all nodes continuously.

The protocol presented in [8] suggests the so called Grid Box hierarchies to process queries in a structured fashion, which (compared to our protocol) involves increased message sizes and more complicated (so more vulnerable) execution which involves a logarithmic number of phases to calculate a single value. On the other hand, the overall approach is similar in the sense that all nodes are equivalent (run the same algorithm) and they all learn the end result.

Kempe et al. [15] propose an aggregation protocol similar to ours: it is based on gossiping and is tailored to work on random topologies. The main difference with the present work is that they consider push-only gossiping mechanisms, which results in a slightly more complicated (though still very simple) protocol. The complication comes from the fact that in a push-only approach some nodes attract more "weight" due to their more central position, so a normalization factor needs to be kept track of as well. Besides, other difficulties arise in practical settings if the directed graph used to push messages is not strongly connected. In our case the effective communication topology is undirected so we need only weak connectivity to allow the protocol

to work. Furthermore, their discussion is limited to theoretical analysis, while we consider the practical details needed for a real implementation and evaluate their performance in unreliable and dynamic environments through simulations.

Related work targeted specifically to network size estimation should also be mentioned. A typical approach is to sample some property of the system which is random but depends on network size and so can be used to apply maximum likelihood estimation or a similar technique. This approach was followed in [20] in the context of multicasting. Another, probabilistic and localized technique is described in [9] where a logical ring is maintained and all nodes estimate network size locally based on the estimates of their neighbors. Unlike these approaches, our protocol provides the exact size in the absence of failures (assuming also that size is an integer which limits the necessary numeric precision) with very low cost and the approximation continues to be very accurate in highly unreliable and dynamic environments.

In principle, aggregation (even in the presence of malicious failures) could be achieved as follows: nodes run a protocol solving the agreement problem [21] (or the weaker approximate agreement problem [4, 6]) with their local values as the input. This suggests that the problems of aggregation and agreement are related. However, agreement protocols are designed for relatively small scale systems where the main problem is to deal with Byzantine failure. Agreement protocols are typically round based, requiring each node to communicate with every other node in a given interval of time (round). While the problem itself is similar, this approach is clearly not practical in the highly dynamic and extremely large scale settings we have in mind.

Finally, aggregation is an important problem in wireless and ad hoc networks as well. For instance, [17] represents a reactive approach where queries are propagated through the system and the answer propagates back to the source node (see the distinction between reactive and proactive approaches in the Introduction). The approach introduced in [16] is similar to ours. It is assumed that the network is a one-hop network (so all nodes can directly communicate with any other node), and a protocol is described that can manage the matching process that implements neighbor selection in this environment.

10 Conclusions

We have presented a full-fledged proactive aggregation protocol and have demonstrated several desirable properties including low cost, rapid convergence, robustness and adaptivity to network dynamics through theoretical an experimental analysis.

We proved that in the case of average calculation, the variance of the approximation of the average decreases exponentially fast, independently of network size. This results suggests both efficiency and scalability. We demonstrated that the method can be applied to calculate a number of aggregates beside the average. These include the maximum and minimum, geometric and harmonic means, network size, sum and product. We proved theoretically that the protocol is not sensitive to node crashes, which confirms our approach of not introducing a leave protocol, but instead handling leaves as crashes. Link failures were also shown to only slightly slow down convergence.

The protocol was simulated on top of several different topologies, including random graphs, the complete graph, small-world networks like the Watts-Strogatz and Barabási-Albert topologies, and a dynamic adaptive unstructured network: NEWSCAST. It was demonstrated that the protocol is efficient on all of these topologies that have a small diameter.

We tested the robustness of the protocol in several failure scenarios. We have seen that very accurate estimates for the aggregate values can be obtained even if 75% of the nodes crash during the running of the protocol. Furthermore, it was confirmed empirically that the protocol is unaffected by link failures, which result only in a proportional slowdown but no loss in accuracy.

Effects of single messages being lost are more severe but for reasonable levels of message loss, the protocol continues to provide highly-accurate aggregate values. Robustness to message loss can be greatly improved by the inexpensive and simple extension of running multiple instances of the protocol concurrently and calculating the final estimate based on the results of the concurrent instances. For node crashes and link failures, our experimental results are supported by theoretical analysis. Finally, the empirical analysis of the protocol was completed with emulations on PlanetLab that confirmed our theoretical and simulation results.

Acknowledgements

This work was partially supported by the Future and Emerging Technologies unit of the European Commission through Projects BISON (IST-2001-38923) and DELIS (IST-001907).

We would like to give credit to Wojtek Kowalczyk, whose ideas significantly contributed to the formulation of the basic averaging protocol and served as a valuable inspiration for the theoretical discussion. We would also like to thank Toni Binci for implementing the protocol for the PlanetLab environment. Finally, we are grateful to the anonymous reviewers for their insightful and constructive suggestions.

References

[1] A.-L. Barabási. *Linked: the new science of networks*. Perseus, Cambridge, Mass., 2002.

[2] A. Bavier, M. Bowman, B. Chun, D. Culler, S. Karlin, S. Muir, L. Peterson, T. Roscoe, T. Spalink, and M. Wawrzoniak. Operating system support for planetary-scale services. In *Proceedings of the First Symposium on Network Systems Design and Implementation (NSDI'04)*, pages 253–266. USENIX, 2004.

[3] 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 *Proceedings of the 6th Annual ACM Symposium on Principles of Distributed Computing (PODC'87)*, pages 1–12, Vancouver, British Columbia, Canada, August 1987. ACM Press.

[4] D. Dolev, N. Lynch, S. Pinter, E. Stark, and W. Weihl. Reaching approximate agreement in the presence of faults. *Journal of the ACM*, 33(3):499–516, July 1986.

[5] P. T. Eugster, R. Guerraoui, A.-M. Kermarrec, and L. Massoulié. Epidemic information dissemination in distributed systems. *IEEE Computer*, 37(5):60–67, May 2004.

[6] A. Fekete. Asynchronous approximate agreement. *Information and Computation*, 115(1):95–124, November 1994.

[7] B. Ghosh and S. Muthukrishnan. Dynamic load balancing by random matchings. *Journal of Computer and System Sciences*, 53(3):357–370, December 1996.

[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. IEEE Computer Society.

[9] K. Horowitz and D. Malkhi. Estimating network size from local information. *Information Processing Letters*, 88(5):237–243, 2003.

[10] M. Jelasity, R. Guerraoui, A.-M. Kermarrec, and M. van Steen. The peer sampling service: Experimental evaluation of unstructured gossip-based implementations. In H.-A. Jacobsen, editor, *Middleware 2004*, volume 3231 of *Lecture Notes in Computer Science*. Springer-Verlag, 2004.

[11] M. Jelasity, A. Montresor, and O. Babaoglu. Detection and removal of malicious peers in gossip-based protocols. In *FuDiCo II: S.O.S.*, Bertinoro, Italy, June 2004. `http://www.cs.utexas.edu/users/lorenzo/sos/`.

[12] M. Jelasity, A. Montresor, and O. Babaoglu. A modular paradigm for building self-organizing peer-to-peer applications. In G. Di Marzo Serugendo, A. Karageorgos, O. F. Rana, and F. Zambonelli, editors, *Engineering Self-Organising Systems*, volume 2977 of *Lecture Notes in Artificial Intelligence*, pages 265–282. Springer, 2004.

[13] 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, October 2002.

[14] J. Joseph and C. Fellenstein. *Grid Computing*. Prentice Hall, 2003.

[15] D. Kempe, A. Dobra, and J. Gehrke. Gossip-based computation of aggregate information. In *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science (FOCS'03)*, pages 482–491. IEEE Computer Society, 2003.

[16] M. Kutyłowski and D. Letkiewicz. Computing average value in ad hoc networks. In B. Rovan and P. Vojtáš, editors, *Mathematical Foundations of Computer Science (MFCS'2003)*, number 2747 in Lecture Notes in Computer Science, pages 511–520. Springer, 2003.

[17] S. Madden, R. Szewczyk, M. J. Franklin, and D. Culler. Supporting aggregate queries over ad-hoc wireless sensor networks. In *Fourth IEEE Workshop on Mobile Computing Systems and Applications (WMCSA'02)*, pages 49–58, Callicoon, New York, 2002. IEEE Computer Society.

[18] 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.

[19] A. Montresor, M. Jelasity, and O. Babaoglu. Decentralized ranking in large-scale overlay networks. Technical Report UBLCS-2004-18, University of Bologna, Dept. of Computer Science, Bologna, Italy, December 2004. `http://www.cs.unibo.it/pub/TR/UBLCS/2004/2004-18.pdf`.

[20] M. Nekovee, A. Soppera, and T. Burbridge. An adaptive method for dynamic audience size estimation in multicast. In B. Stiller, G. Carle, M. Karsten, and P. Reichl, editors, *Group Communications and Charges: Technology and Business Models*, number 2816 in Lecture Notes in Computer Science, pages 23–33. Springer, 2003.

[21] M. Pease, R. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *Journal of the ACM*, 27(2):228–234, 1980.

[22] PeerSim. http://peersim.sourceforge.net/.

[23] M. Ripeanu, A. Iamnitchi, and I. Foster. Mapping the gnutella network. *IEEE Internet Computing*, 6(1):50–57, 2002.

[24] S. Saroiu, P. K. Gummadi, and S. D. Gribble. Measuring and analyzing the characteristics of Napster and Gnutella hosts. *Multimedia Systems Journal*, 9(2):170–184, August 2003.

[25] 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.

[26] 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):164–206, May 2003.

[27] 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.

[28] D. J. Watts. *Small Worlds: The Dynamics of Networks Between Order and Randomness*. Princeton University Press, 1999.

[29] D. J. Watts and S. H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393:440–442, 1998.

[30] P. Yalagandula and M. Dahlin. A scalable distributed information management system. In *Proceedings of ACM SIGCOMM 2004*, pages 379–390, Portland, Oregon, USA, 2004. ACM Press.