Robust Aggregation Protocols for Large-Scale Overlay Networks*

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

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

Ozalp Babaoglu
Dept. of Computer Science
University of Bologna, Italy
babaoglu@cs.unibo.it

Abstract

Aggregation *refers to a set of functions that provide global information about a distributed system. These functions operate on numeric values distributed over the system and can be used to count network size, determine extremal values and compute averages, products or sums. Aggregation allows important basic functionality to be achieved in fully distributed and peer-to-peer networks. For example, in a monitoring application, some aggregate reaching a specific value may trigger the execution of certain operations; distributed storage systems may need to know the total free space available; load-balancing protocols may benefit from knowing the target average load so as to minimize the transfered load. Building on the simple but efficient idea of anti-entropy aggregation (a scheme based on the anti-entropy epidemic communication model), in this paper we introduce practically applicable robust and adaptive protocols for proactive aggregation, including the calculation of average, product and extremal values. We show how the averaging protocol can be applied to compute further aggregates like sum, variance and the network size. We present theoretical and empirical evidence supporting the robustness of the averaging protocol under different scenarios.*

1. Introduction

The latest generation of peer-to-peer (P2P) networks are typically self-organizing, large-scale distributed systems. Unlike many traditional distributed systems, however, neither a central authority nor a fixed communication topology are employed to control the various components. Instead, a dynamically changing overlay network is maintained and control is completely decentralized with "cooperation" links among nodes being created and deleted based on the requirements of the particular application. Such systems are attractive for several reasons, including the lack of single points of failure, the potential to scale to millions of nodes, and the fact that they allow the creation of relatively inexpensive distributed computing platforms.

The decentralized nature of such systems, however, presents certain drawbacks. P2P systems tend to be highly dynamic, with a continuous flow of nodes joining and leaving the network. Control and monitoring in such systems are difficult tasks: performing global computations requires orchestrating a huge number of nodes.

A useful building block for monitoring and control in P2P systems is *aggregation*, which is the collective name given to a set of functions that provide statistical information about the system [10]. These functions include finding extremal values of some property, computing averages and sums, etc. Aggregation can provide participants in a P2P network with important global information such as the size of the network or the average load in a network. Furthermore, aggregation can be used as a building block for obtaining more complex protocols. For example, the knowledge of the average load in a network can be exploited to implement near-optimal load-balancing schemes [6].

Aggregation protocols can be divided into two categories: *reactive* and *proactive*. Reactive protocols respond to specific queries issued by nodes in the network. The answers are returned directly to the issuer of the query [3]. Proactive protocols, on the other hand, continuously provide the value of some aggregate 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 values being aggregated, the output of the aggregation protocol should follow this change reasonably quickly. Proactive protocols are often useful when aggregation is used as a building block for completely decentralized protocols. For example, in the load-balancing scheme cited above, the knowledge of the average load is used by each node to decide when it can stop transferring load [6].

In this paper we introduce a robust and adaptive protocol for calculating aggregates in a proactive manner. The core of the protocol is a simple scheme [5] in which aggregation is performed in the style of an anti-entropy epidemic protocol, typically used for propagating updates in distributed databases [2]. Periodically, each node selects a random peer and communicates with it to bring the two states up-to-date. In our case, instead of resolving differences between databases, the elementary step consists of

* This work was partially supported by the FET unit of the European Commission through Project BISON (IST-2001-38923).
† also with MTA RGAI, SZTE, Szeged, Hungary

some aggregation-specific computation based on the values maintained by the two communicating peers.

Our contribution is 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 additional aggregates such as variances and products. Third, we present both theoretical and experimental evidence on the robustness of our protocol.

2. System Model

We consider a P2P 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 nodes may join at any time, and existing nodes may leave, either voluntarily or by *crashing*. Our protocol does not need any mechanism specific to leaves so crashes and voluntary leaves can be 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 [7]).

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 the *overlay network*. Given the large scale and the dynamicity of our envisioned system, neighborhoods are typically limited to small subsets of the entire network. Communication incurs unpredictable delays and is subject to failures. Single messages may be lost, links between pairs of nodes may break. Occasional performance failures of nodes (e.g., delay in receiving or sending a message in time) can be seen as 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 link failure and message loss. Some other aspects of the model that are outside of the focus of our present analysis—like clock drift and message delay—are discussed only informally in Section 4.

3. Aggregation: the Basic Idea

Each node in the network holds a numeric value. In a practical setting, this value can characterize any (dynamic) aspect of the node or its environment (e.g., the load at the node, temperature monitored by a sensor network). 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.

Our basic aggregation protocol is based on the epidemic-style push-pull scheme illustrated in Figure 1. Each node p executes two different threads. The *active* thread periodically initiates an *information exchange* with a peer node q selected randomly among its neighbors, by sending q 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 peers send and receive their states.

do forever
wait(δ time units)
$q \leftarrow$ GETNEIGHBOR()
send s_p to q
$s_q \leftarrow$ receive(q)
$s_p \leftarrow$ UPDATE(s_p, s_q)

(a) active thread

do forever
$s_q \leftarrow$ receive(*)
send s_p to sender(s_q)
$s_p \leftarrow$ UPDATE(s_p, s_q)

(b) passive thread

Figure 1. Protocol executed by node p.

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 UPDATE builds a new local state based on the previous one and the remote state received during the information exchange. The output of UPDATE depends on the specific function being implemented by the protocol. Here, we limit the discussion to AVERAGE, which computes the global average. Additional functions are described in Section 4.

To implement AVERAGE, each node stores a single numeric value representing the current estimate of the aggregation output. 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 distributes the initial sum equally among the two peers. So, the operation does not change the global average either; it only decreases the variance over all the estimates.

It is easy to see that the value at each node will converge to the true global average, as long as the underlying overlay network remains connected. In our previous work [5], we presented analytical results for the convergence speed of the averaging protocol. Let μ_i be the empirical mean and σ_i^2 be the empirical variance of the local estimates at cycle i:

$$\mu_i = \frac{1}{N}\sum_{k=1}^{N} a_{i,k}, \quad \sigma_i^2 = \frac{1}{N-1}\sum_{k=1}^{N}(a_{i,k} - \mu_i)^2 \qquad (1)$$

where $a_{i,k}$ is the estimate maintained at node $k = 1, \ldots N$ during cycle i and N is the number of nodes.

The *convergence factor* ρ_i, with $i \geq 1$, characterizes the speed of convergence for the aggregation protocol and is defined as $\rho_i = E(\sigma_i^2)/E(\sigma_{i-1}^2)$. If the (connected) overlay network topology is sufficiently random, it is possible to show that for $i \geq 1$, $\rho_i \approx 1/(2\sqrt{e})$. In other words, each cy-

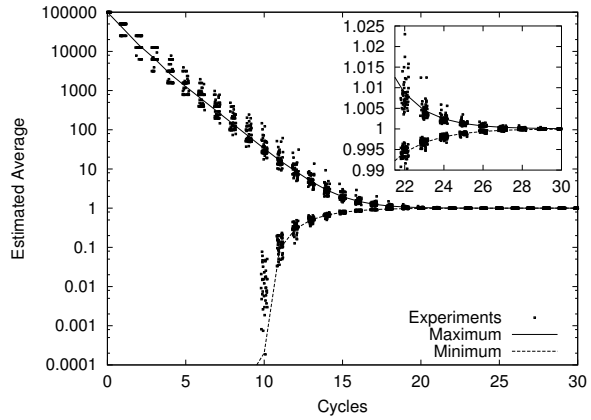


Figure 2. Behavior of protocol AVERAGE.

cle of the protocol reduces the expected variance of the local estimates by a factor $2\sqrt{e}$. From this result, it is clear that the protocol converges exponentially and very high precision estimates of the true average can be achieved in only a few cycles, irrespective of the network size, confirming the extreme scalability of our protocol.

Figure 2 illustrates the behavior of the protocol. The AVERAGE protocol was run on a simulated network composed of 10^5 nodes connected through a regular random overlay network, where each node knows exactly 20 neighbors. Initially, a single node has the value 10^5, while all others have zero as their local value (so that the global average is 1). We are interested in this *peak distribution* for two reasons: first, it will be the basis of our COUNT protocol presented in Section 4; and second, it is the most demanding scenario for testing robustness, since the single node holding the initial peak value constitutes a single point of failure.

In the Figure, results for the first 30 cycles of the protocol are shown. The two curves represent the minimum and the maximum estimates of the average over all the nodes at the completion of each cycle. The curves correspond to averages over 50 independent experiments, whose results are shown as individual points in the figure.

4. Aggregation: A Practical Protocol

4.1. Automatic Restarting

The generic protocol described so far is not adaptive, as the aggregation takes into account neither the dynamicity of the network nor the variability of values. 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 aggregation output; then, the current 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.

To implement restarting, we divide the protocol execution in consecutive *epochs* of length Δ and start a new instance of the protocol in each epoch. Depending on the ratio between Δ and $\gamma\delta$, it is possible that different epochs of the protocol may be executing concurrently in the network. Thus, messages exchanged for a particular epoch have to be tagged with unique epoch identifiers.

4.2. Dynamic Membership

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 existing 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.

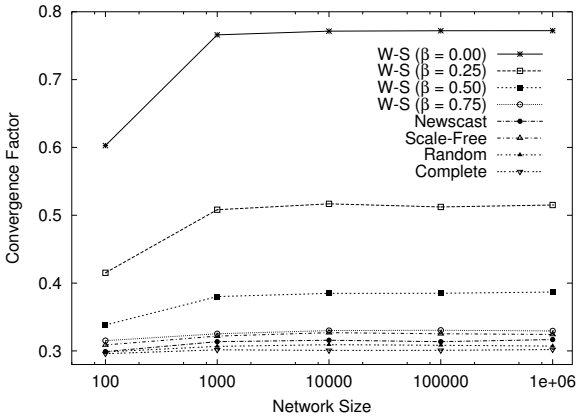
As for crashes, when a node initiates a exchange, it sets a timeout period to detect the failure of the contacted node. If the timeout expires before the message is received, the exchange step is skipped. The effect of these missing exchanges due to real (or presumed) failures on the final average will be discussed in Section 7.

4.3. Synchronization

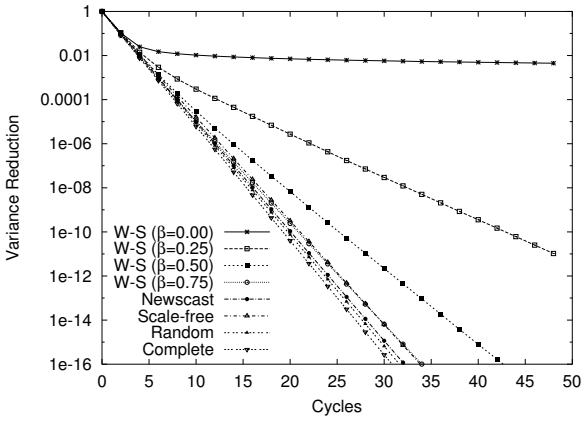
The protocol described so far is based on the assumption that cycles and epochs proceed in lock step at all nodes. In a large-scale distributed system, this assumption cannot be satisfied due to the unpredictability of message delays and the different drift rates of local clocks.

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 an 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 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 [2] and assuming that each message arrives within the timeout used during all communications, we can give a logarithmic



(a) Average convergence factor computed over a period of 20 cycles in networks of varying size. Each curve corresponds to a different topology and W-S stands for the Watt-Strogatz model.



(b) Variance reduction (normalized by the initial variance) for a network of 10^5 nodes. Each curve corresponds to a different topology and W-S stands for the Watt-Strogatz model.

Figure 3. Behavior of protocol AVERAGE.

bound on T_j for each epoch j. Considering also that typically many nodes will start the new epoch independently, 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. Although it would be interesting, further analysis of this mechanism is out of the scope of the present discussion. The effect of lost messages (i.e., those that time out) is discussed later.

4.4. Overlay Topology for Aggregation

The theoretical results mentioned 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. To explore deviation from the theoretically predicted behavior, Figure 3(a) illustrates the performance of aggregation for different topologies, by showing the average convergence factor over a period of 20 cycles, for network sizes ranging from 10^2 to 10^6 nodes. Figure 3(b) provides additional details. Here, the network size is fixed at 10^5 nodes. Instead of displaying the average convergence factor, the curves represent the actual variance reduction normalized with respect to the initial variance for the same set of topologies. Before going into the details of the topologies, we can already conclude that performance is independent of network size for all topologies, while it is highly sensitive to the particular topologies. Furthermore, the convergence factor is constant through the sequence of cycle, with non-random topologies being the only exceptions.

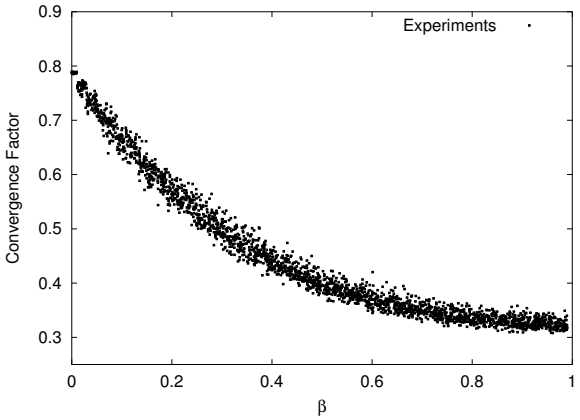
All the topologies examined in this section (with the exception of NEWSCAST) are static—the neighbor set of each node is fixed. While static topologies are unrealistic in the presence of node joins and crashes, we still consider them due to their theoretical importance and the fact that our protocol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

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 Barabasi-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.

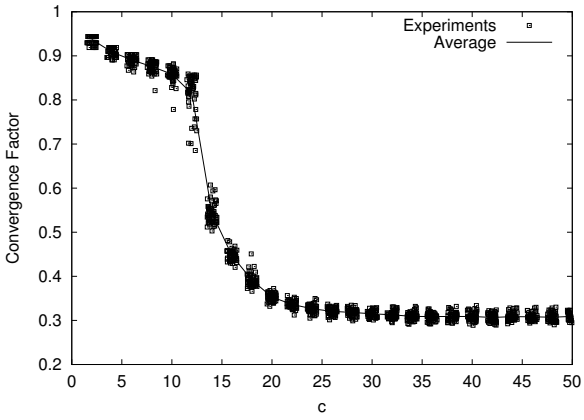
The remaining topologies are realistic *small-world* topologies that are often used to model different natural and artificial phenomena [1, 13]. The first class of these topologies (the Watts-Strogatz model [14]) is built starting from a regular ring lattice. The ring lattice is built by connecting the 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 probabilityβ. 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.

Figure 4(a) focuses on the Watts-Strogatz model showing the convergence factor as a function of β ranging from 0 (complete order) to 1 (complete disorder). Although there is no sharp phase transition, we observe that increased randomness results in a better convergence factor.

Scale-free topologies form the other class of realistic small world topologies. In particular, the WWW, the Internet and P2P networks such as Gnutella [9] 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 Barabasi 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 ex-



(a) Convergence factor for Watts-Strogatz graphs with a variable β parameter.



(b) Convergence factor for NEWSCAST graphs with a variable c parameter.

Figure 4. Behavior of the AVERAGE protocol in Watts-Strogatz and NEWSCAST graphs.

isting contact node is picked randomly with a probability proportional to its degree (number of neighbors). The results are encouraging, as the observed convergence factors are similar to those obtained in random graphs.

Dynamic topologies From the above results, it is clear that the topology of the overlay should be as random as possible. Furthermore, in dynamic systems, there must be mechanisms in place that preserve this property. To achieve this goal, we propose to use NEWSCAST [4], a decentralized membership protocol based on an epidemic scheme similar to the one described in Figure 1.

In NEWSCAST, the overlay is generated by a continuous exchange of neighbor sets, where each neighbor is associated with its 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 get replaced by new information. This feature allows the protocol to "repair" the overlay topology by forgetting information about crashed neighbors, which by definition do not inject their identifier.

The resulting topology has a very low diameter [4]. Figure 4(b) show the performance of aggregation over a NEWSCAST network of 10^5 nodes, with c varying between 2 and 50. According to our experimental results, choosing $c = 30$ is already sufficient to obtain fast convergence for aggregation. Furthermore, the same value is sufficient for very stable and robust connectivity.

4.5. Cost Analysis

Both the communication cost and time complexity of our algorithms 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 and 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. Other Aggregation Functions

In the following, we briefly outline how our protocol for averaging can be easily modified to compute several other interesting aggregation functions.

MIN and MAX: To obtain the maximum or minimum value among those maintained by 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. Existing results about epidemic-style broadcasting [2] are applicable.

COUNT: We base this protocol 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.

However, implementing this function in distributed system where new nodes may appear and existing ones may crash may not be possible. An alternative approach consists in enabling multiple nodes to start concurrent instances of the averaging protocol. Each concurrent instance is lead by a different node. Messages and data related to an instance are tagged with a unique identifier (e.g., the address of the leader). Each node maintains a map M associating a leader id 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.

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 cost 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.

SUM: Two concurrent aggregation protocols are run, one to estimate the size of the network, the other to estimate the average of the values to be summed. Size and average are multiplied in order to obtain an estimate of the sum.

GEOMETRICMEAN and PRODUCT: In order to compute the geometric mean and the product of the values contained in the network, the same approach to compute the arithmetic mean and the sum may be used. Rather than returning the arithmetic mean of the two local values, method UPDATE(a, b) now returns $\sqrt{ab}$. After one cycle, the product of the two local values remains unchanged, but the variance over the set of values decreases such that the local estimates converge toward the global geometric mean. As before, once the geometric mean is known with sufficient precision, the result of a concurrent COUNT protocol can be used to obtain the product.

VARIANCE: We run two instances of the averaging protocol to compute $\overline{a}$, the average of the initial values and $\overline{a^2}$, the average of the squares of the initial values. Then, an estimate for the variance can be obtained as $\overline{a^2} - \overline{a}^2$.

6. Theoretical Results on Benign Failure

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, nodes come and go 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, P_f, of the nodes crash. Given N^* nodes initially, $P_f N^*$ nodes are removed (without replacement).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.

Let us begin with some simple observations. Using the notations in (1) in our failure model the expected value of μ_i and σ_i^2 will stay the same independently of P_f since the model is completely symmetric. The variance reduction rate also remains the same since it does not rely on any particular network size. So the only interesting thing is the variance of μ_i, which characterizes the expected error of the approximation of the average. We will describe the variance of μ_i as a function of P_f.

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

$$Var(\mu_i) = \frac{P_f}{N(1-P_f)} E(\sigma_0^2) \frac{1 - \left(\frac{\rho}{1-P_f}\right)^i}{1 - \frac{\rho}{1-P_f}}. \qquad (2)$$

Proof. Let us take the decomposition $\mu_{i+1} = \mu_i + d_i$. Random variable d_i is independent of μ_i so

$$\text{Var}(\mu_{i+1}) = \text{Var}(\mu_i) + \text{Var}(d_i). \qquad (3)$$

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

$$\begin{aligned} \text{Var}(d_i) \quad &= E((\mu_i - \mu_{i+1})^2) = \frac{P_f}{N_i(1-P_f)} E(\sigma_i^2) \\ &= \frac{P_f}{1-P_f} E(\sigma_0^2) \frac{\rho^i}{N(1-P_f)^i}. \end{aligned} \qquad (4)$$

Now, from (3) we see that $\text{Var}(\mu_i) = \sum_{j=0}^{i-1} \text{Var}(d_j)$ which gives the desired formula when substituting (4). □

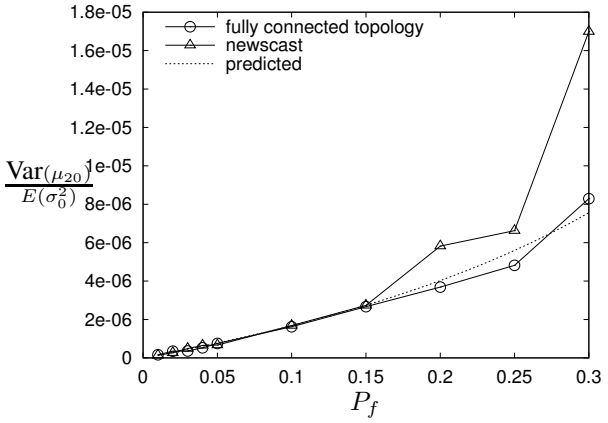


Figure 5. Effects of node crashes on the variance computed by AVERAGE at cycle 20.

We performed simulations with $N = 10^5$ to validate these results (see Figure 5). For each value of P_f, the empirical data is based on 100 independent experiments whereas the prediction is obtained from (2) 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 damage. See also Figure 6(b), which depicts the whole distribution of the estimates with NEWSCAST, not only the normalized variance.

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 μ_i. This is optimal 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. 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.

In [5] it was proven that $\rho = 1/e$ 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. Note that assuming the protocol described in Section 3—where $\rho = 1/(2\sqrt{e})$—this assumption is not true, because there, it is guaranteed that each node participates in at least one variance reduction step, the one initiated actively by the node. In this more 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" (less pairwise exchanges are performed in a unit time interval). In the limit case when P_d is close to 1 the randomness assumption described above (when $\rho = 1/e$) is fulfilled with a 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

$$\rho_d = (\frac{1}{e})^{1-P_d} = e^{P_d - 1} \tag{5}$$

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

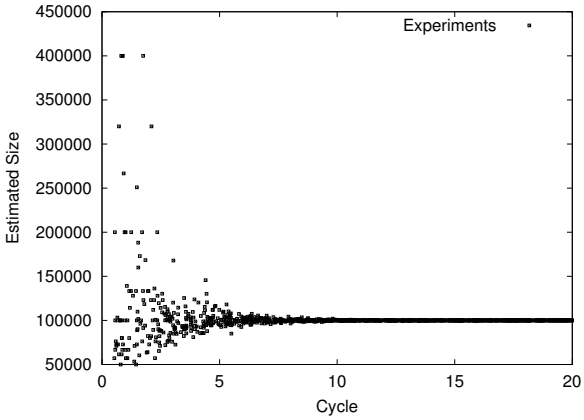
7. Simulation Results on Benign Failures

To complement the theoretical analysis, we have performed numerous experiments based on simulation. In all experiments, we have used NEWSCAST as the underlying overlay network. The reason for this choice is twofold: first, we want to show empirical results in a realistic overlay network that can actually be built in a decentralized way; second, NEWSCAST is known to be robust and capable of maintaining a sufficiently random network in the failure scenarios we are analyzing [4].

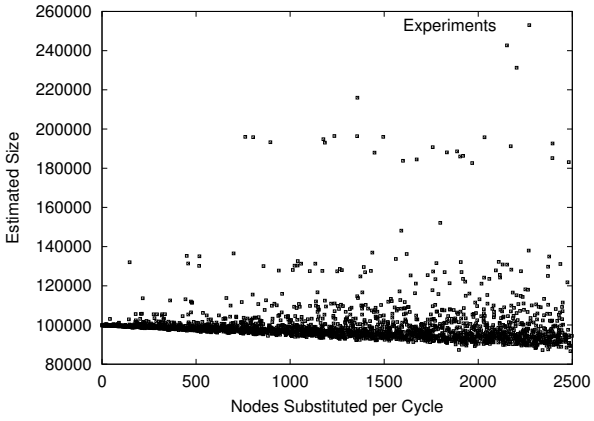
Furthermore, all our experiments were performed with the COUNT aggregation protocol since it represents a worst-case being the most sensitive to failures (both node crashes and message omissions). During the first few cycles of an epoch when only a few nodes have a local estimate other than 0, their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

All experiments in the paper are performed with PEERSIM, a simulator developed by us and optimized for our aggregation protocol [6]. 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 3(a)).

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 estimations 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. The result of each experiment is shown in the figure as a dot to illustrate the entire distribution. A small random factor is added to the x-coordinates so as to separate results having similar



(a) 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.



(b) Network size estimation with protocol COUNT in a network of constant size subject to a continuous flux of nodes joining and crashing. At each cycle, a variable number of nodes crash and are substituted by the same number of new nodes.

Figure 6. Effects of node crashes on the COUNT protocol in a NEWSCAST network.

y-coordinates. When applicable, averages computed over all experiments are also shown as curves.

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 crashing 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 6(a), 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.

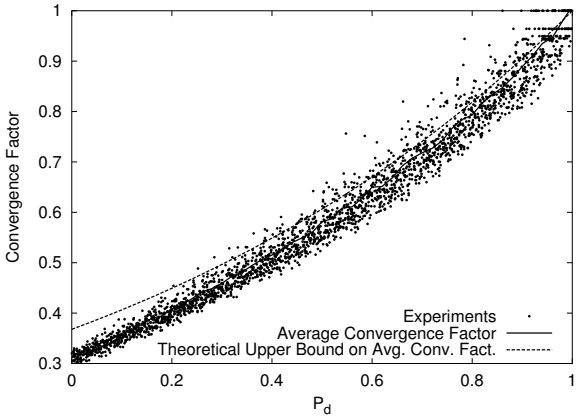
A more realistic scenario is a network characterized by continuously leaving and joining nodes. Figure 6(b) illustrates the behavior of aggregation in such a network. At each cycle, a variable number of nodes are removed from the network and are substituted with completely new nodes. In this way, the 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 (recall that joining nodes cannot participate until the start of the next epoch) at the end of a single epoch (30 cycles). Note that although the average estimate is plotted, in cycle 30 the estimates are practically identical at all nodes as Figure 3(b) 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 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 is still the same as according to Sections 6.2 and 7.2 link failure does 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 5 for the estimate varianceplotted against the theoretical prediction.

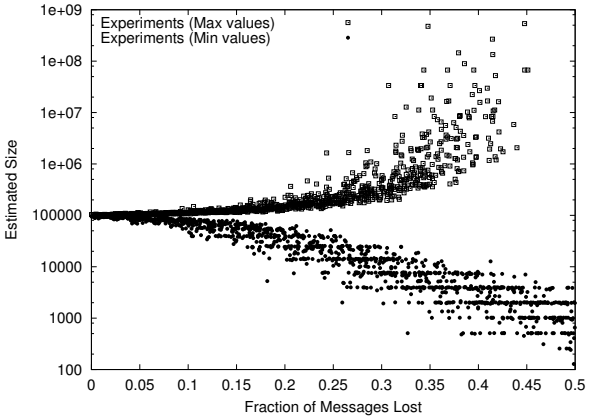
7.2. Link Failures and Message Omissions

Figure 7(a) 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 (5)) 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, it is also possible that single messages are lost. If the message sent to initiate an

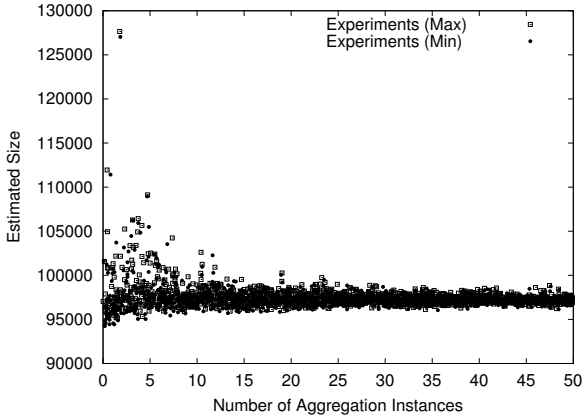


(a) Convergence factor of protocol COUNT as a function of link failure probability.

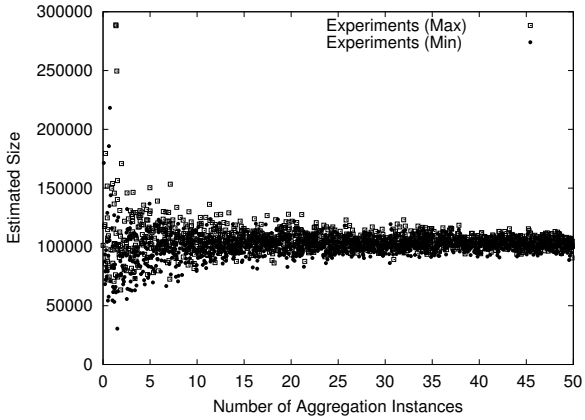


(b) Network size estimation with protocol COUNT as a function of lost messages.

Figure 7. Effects of communication failures on the COUNT protocol in a NEWSCAST network.



(a) Network size estimation with multiple instances of protocol COUNT. 1000 nodes crash at the beginning of each cycle.



(b) Network size estimation with protocol COUNT as a function of concurrent protocol instances. 20% of messages are lost.

Figure 8. Effects of failures on protocol COUNT with multiple concurrent instances.

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 7(b). 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. 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 order of magnitudes. 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 in the range between 1 and 50. At each node, the t estimates 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 8(a) shows the results obtained by applying this technique in a system where 1000 nodes per cycle are substituted with new nodes, while Figure 8(b) 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 (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.

8. Related Work

Protocols based on the analogy with the spreading of epidemics have found many applications. Examples include database replication [2] and failure detection [12].

The idea of aggregation was pionereed by Van Renesse in Astrolabe [10, 11]. In Astrolabe, a hierarchical architecture is deployed. While this approach reduces the cost of finding the aggregates and enables the execution of complex, database-style queries, the maintenance of the hierarchical topology introduces additional overhead. Our anti-entropy aggregation is substantially different, since it is extremely simple, lightweight, and targeted to unstructured, highly dynamic environments. Furthermore, our protocol is proactive: the result of aggregation is known to all nodes.

Kempe et al. [8] propose an aggregation protocol similar to ours. The main difference is that their protocol is based on push-only gossiping mechanisms. Furthermore, their discussion is limited to the theoretical version of the protocol, while this paper discusses at length the practical details needed for a real implementation and assesses its performance in a faulty environment.

9. Conclusions

In this paper we have presented a full-fledged proactive aggregation protocol and we have demonstrated its adaptivity and robustness to benign failure through theoretical an experimental analysis.

We have seen that a reasonably good approximation can be obtained even if 75% of the nodes crash during the calculation of the aggregate. Furthermore, the protocol is totally insensitive to link failure, which results only in proportional slowdown but no approximation error. In the case of message loss we did not see this extreme and exceptional robustness, but the protocol is still reliable under usual, reasonable levels of message loss. However, as was shown, robustness to message loss can be greatly improved by the inexpensive and simple extension of running multiple instances of the protocol at the same time and calculate the final approximation based on the output of the parallel instances. In the case of crash and link failure our experimental results are supported by theoretical analysis as well.

In short, based on the presented results we can conclude that our aggregation protocol is robust to benign failure and can cope with an extremely high level of dynamism. The fact that our experiments were carried out using the worst case peak distribution further confirms the practical applicability of the protocol.

References

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

[2] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry. Epidemic Algorithms for Replicated Database Management. In *Proc. of PODC'87*, Vancouver, Aug. 1987. ACM.

[3] I. Gupta, R. van Renesse, and K. P. Birman. Scalable fault-tolerant aggregation in large process groups. In *Proc. of DSN'01*, Göteborg, Sweden, 2001.

[4] M. Jelasity, W. Kowalczyk, and M. van Steen. Newscast computing. Technical Report IR-CS-006, Vrije Universiteit Amsterdam, Department of Computer Science, Amsterdam, The Netherlands, Nov. 2003.

[5] M. Jelasity and A. Montresor. Epidemic-Style Proactive Aggregation in Large Overlay Networks. In *Proc. of the 24th International Conference on Distributed Computing Systems (ICDCS 2004)*, Tokyo, Japan, 2004.

[6] M. Jelasity, A. Montresor, and O. Babaoglu. A modular paradigm for building self-organizing peer-to-peer applications. In *Proc. of the 1st Internation Workshop on Engineering Self-Organizing Applications (ESOA'03)*, Melbourne, Australia, 2003.

[7] M. Jelasity, A. Montresor, and O. Babaoglu. Towards secure epidemics: Detection and removal of malicious peers in epidemic-style protocols. Technical Report UBLCS-2003-14, University of Bologna, Bologna, Italy, Nov. 2003.

[8] D. Kempe, A. Dobra, and J. Gehrke. Computing aggregate information using gossip. In *Proc. of FOCS'03*, 2003.

[9] M. Ripeanu, I. Foster, and A. Iamnitchi. Mapping the gnutella network: Properties of large-scale peer-to-peer systems and implications for system design. *IEEE Internet Computing Journal*, 6(1), 2002.

[10] 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*, LNCS 2584. Springer, 2003.

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

[12] 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*. Springer, 1998.

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

[14] D. J. Watts and S. H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393, 1998.