Fault-Tolerant Aggregation by Flow Updating

Paulo Jesus, Carlos Baquero, and Paulo Sérgio Almeida

University of Minho (CCTC-DI)
Campus de Gualtar, 4710-057 Braga, Portugal
{pcoj, cbm, psa}@di.uminho.pt

Abstract. Data aggregation plays an important role in the design of scalable systems, allowing the determination of meaningful system-wide properties to direct the execution of distributed applications. In the particular case of wireless sensor networks, data collection is often only practicable if aggregation is performed. Several aggregation algorithms have been proposed in the last few years, exhibiting different properties in terms of accuracy, speed and communication tradeoffs. Nonetheless, existing approaches are found lacking in terms of fault tolerance. In this paper, we introduce a novel fault-tolerant averaging based data aggregation algorithm. It tolerates substantial message loss (link failures), while competing algorithms in the same class can be affected by a single lost message. The algorithm is based on manipulating flows (in the graph theoretical sense), that are updated using idempotent messages, providing it with unique robustness capabilities. Furthermore, evaluation results obtained by comparing it with other averaging approaches have revealed that it outperforms them in terms of time and message complexity.

1 Introduction

Traditional solutions based on centralized and tightly architected computation infrastructures are being challenged by new designs that amass the resources in highly distributed computing systems. Notable examples are found in large scale peer-to-peer systems, now in common use, and in the algorithms that will support the deployment of vast sensor networks.

In these settings, aggregation of data across large numbers of nodes plays a basal role in the design of scalable solutions [1]. Distributed aggregation along active nodes allows the efficient determination of meaningful global system properties, that can direct the actions of self-adaptive distributed algorithms.

Examples can be found when using estimates of the network size to direct the dimensioning of distributed hash table structures [2], when setting a quorum for voting algorithms [3], when estimates of the average system load are needed to direct local load-balancing decisions, or when an estimate of the total disk space in the network is required in a P2P sharing system.

Several aggregation algorithms have been introduced in the recent years, tackling the problem for different settings, and yielding different characteristics in terms of accuracy, time and communication tradeoffs. Traditional approaches relay on the existence of a specific aggregation structure (e.g. tree) [4–6], executing the aggregation process along a predefined routing topology. Another common class of distributed aggregation algorithms is based on averaging techniques [7–10]; Here, the values of a variable across all nodes are averaged iteratively. This kind of approaches are independent from the routing topology, often using a gossip-based communication scheme between peers.

Averaging techniques allow the derivation of different aggregation functions besides average (like counts, sums, and ranks), according to the combinations of input values. In particular, if the input value is set to 1 in a single node and to 0 in all remaining nodes, it is possible to derive an estimate of the network size, by averaging until a given level of convergence is reached, and later inspecting the inverse of the resulting value [8]. Other types of aggregation algorithms are also known, based on the application of *probabilistic* methods. This is the case of Extrema Propagation [11] and COMP [12], which reduce the computation of an aggregation function to the determination of the maximum/minimum of a collection of random numbers. These probabilistic techniques tend to emphasize speed, being less accurate than averaging techniques.

Specific aggregations, such as counting the number of nodes, are amenable to specialized probabilistic algorithms that can operate using properties of random walks, sample and re-sample techniques and other statistic tools [13–15].

Up to now, it has not been proposed any aggregation technique which is simultaneously accurate and tolerates faults (e.g. message loss) efficiently. In this paper, we introduce a novel averaging based aggregation technique: *Flow Updating*. This new algorithm tolerates quite easily high levels of message loss; a feature that was lacking in previous approaches, where message loss often implies "mass" loss in the amount subject to averaging. Moreover, even in lossless scenarios, our new technique achieves an improved convergence speed compared to previous approaches. We compare the new algorithm with two other established averaging algorithms (Push-Sum Protocol [7] and DRG [9]) in a common simulation environment and contrast the results.

The rest of this paper is organized as follows. The closest related work, in terms of averaging aggregation algorithms, is discussed in Section 2. *Flow Updating* will be described in Section 3. The evaluation of the proposed approach will be presented in Section 4, comparing it to other averaging algorithms, and discussing the obtained results. Finally, conclusions and future work directions will be drawn in Section 5.

2 Related Work

Unlike classical tree-based approaches (e.g. TAG [4]), some approaches that are independent from the routing strategy used to communicate have been proposed in the recent years. Commonly, these distributed aggregation algorithms are based on *averaging* techniques. Nodes start with a given real value, x_i, and an anti-entropy protocol is used to iteratively average the values between pairs of nodes. Eventually all values will converge to the same amount.

This kind of approaches tend to be very accurate, producing the correct result or converging to it along time. Compared with tree-based schemes, these algorithms remove the dependency from a specific routing topology, introducing more flexibility, and allowing the iterative calculation of the aggregation result at all network nodes (instead of a single node). Follows, a brief description of some of those algorithms.

2.1 Push-sum protocol

The push-sum protocol [7] is a gossip-based aggregation algorithm, which essentially consists of an iterative pairwise distribution of aggregated values throughout the network. At each round t, each node i maintains and propagates information of a pair of values $(s_{t,i}, w_{t,i})$, where $s_{t,i}$ represents the sum of the exchanged aggregates, and $w_{t,i}$ denotes the weight associated to this sum at the given time t and node i. In order to compute distinct aggregation functions, from the initial input value x_i of node i, one resorts to distinct initializations to the pair of values, $(s_{0,i}, w_{0,i})$ in each i. E.g. AVERAGE: $s_{0,i} = x_i$ and $w_{0,i} = 1$ for all nodes; SUM: $s_{0,i} = x_i$ for all nodes, only one node sets $w_{0,i} = 1$ and the remaining assume $w_{0,i} = 0$; COUNT: $s_{0,i} = 1$ for all nodes, only one with $w_{0,i} = 1$ and the others with $w_{0,i} = 0$.

The protocol works has follows: at each round, each node sends a pair of values corresponding to half of their current values $(s_{t,i}, w_{t,i})$ to a target node chosen uniformly at random, and to itself. The local values are updated with the correspondent sum of all the data received in the previous round. At each time t, the aggregation result can be estimated at each node by $s_{t,i}/w_{t,i}$. The accuracy of the produced estimate will tend to increase along each round, converging to the correct value.

As referred by the authors, the correctness of this algorithm relies on a fundamental property defined as the *mass conservation*: the global sum of all network estimates is always constant along time. When no messages are in transit, the value $\sum_i \frac{s_{t,i}}{w_{t,i}}$ is the same for any round t. The convergence to the true result will depend on the conservation of this property. Considering the crucial importance of this property, the authors assume the existence of a fault detection mechanism, that allows nodes to detect when a message did not reach its destination. In this situation, the "mass" is restored by sending the undelivered message to the node itself.

We should point out that, contrary to *indulgent* distributed algorithms in which an incorrect output from the failure detector (FD) merely postpones termination, assuming the use of a FD is problematic for realistic implementations, as FD inaccuracy means violating mass conservation.

2.2 Push-pull gossiping

A push-pull gossiping approach, similar to the previous one, is proposed in [16, 8]. This protocol benefits from the better convergence of push-pull interactions, as opposed to push only. Periodically, each node sends its current aggregated value to a random neighbor and waits for the response with the aggregate value of the target. The aggregation function is further applied to both values (sent and received), in order to determine the new estimation and update the local aggregate. Each time a node receives an aggregate from a neighbor, it sends back its current value, and afterwards computes the new aggregate, using the received and sent value as inputs.

Unlike push-sum, the protocol does not use weight variables, imposing greater atomicity requirements on the interaction between node pairs.

2.3 DRG (Distributed Random Grouping)

A different approach based on a distributed random grouping (DRG) was proposed in [9]. DRG was designed to take advantage of the broadcast nature of wireless transmission, in which all nodes within radio range will be prone to hear a transmission. This algorithm defines three different working modes for each node: *leader*, *member*, and *idle* mode.

According to the defined modes, one could divide the execution of the algorithm in three main steps. First, each node in idle mode independently decides to become a group leader (according to a predefined probability), and consequently broadcasts a Group Call Message (GCM) to all its neighbors, subsequently waiting for members. Second, all nodes in idle mode respond to the first received GCM with a Joining Acknowledgment (JACK) tagged with their aggregated value, updating their state mode accordingly to become members of that group. Finally, after gathering the group members values from all received JACKs, the leader computes the group aggregate and broadcast a Group Assignment Message (GAM) with the result, returning to idle mode afterwards. Each group member waits for the leader GAM, not responding to any other request until then, to update its local state (setting its local value with the received group aggregate and returning to idle mode).

The execution of this scheme along time creates distributed random groups that coordinate in-group aggregation. Since groups overlap over time, the estimation will convergence at all nodes to the desired network wide global aggregate. The performance of this algorithm is highly influenced by its capacity to create aggregation groups (quantity and size of groups), which is defined by the predefined probability of a node becoming leader.

This algorithm is vulnerable to message loss between coordinators and neighbors, partial fixes to avoid the possibility of nodes waiting forever may incur in violating mass conservation.

2.4 Further considerations

Averaging aggregation algorithms depend on the mass conservation principle to converge to a correct result. Consequently, the robustness of these algorithms is strongly related to their ability to preserve the global mass of the system. The loss of a partial aggregate (mass) may result in the subtraction of the lost value from the global mass, and convergence to an incorrect value.

A few approaches have recently introduced some practical concerns about aggregation robustness. This is the case of G-GAP [10], that tackles the mass conservation problem, extending the push-synopses protocol [7] in order to provide accurate estimates in the presence of node failures. Despite their effort, G-GAP only supports discontinuous failures of adjacent nodes within a short time period.

In this paper, we introduce an aggregation algorithm that fully overcomes the mass conservation issue under link failures. Apart from its robustness properties, our technique also exhibits good performance. We compare it, in failure-free scenarios, with a representative set of the existing averaging approaches, namely the push-sum protocol [7] and DRG [9].

Push-pull gossiping [16, 8] is not considered in the comparison, since simulations exhibited a violation of mass conservation. We found out that this is due to message interleaving resulting from the natural concurrency in the distributed system model we adopted. This means it does not work, even under no failures, in a realistic model. Fixes can be devised, towards making a pair of push and pull messages to behave as if they are atomic (as implied by the papers presenting the mechanism). However, it means some substantial changes (including preventing deadlock), and we would not be making a comparison with the original algorithm. Even so, we found out the corrected algorithm to be even slower and it does make an unfair comparison.

3 Flow Updating

3.1 System Model

We model a distributed system as a connected undirected graph $G(\mathcal{V}, \mathcal{E})$, in which the set of vertices $\mathcal{V}$ represent the network computation nodes, and the set of edges $\mathcal{E}$ correspond to bidirectional communication links. We consider only a fixed topology. We define $\mathcal{D}_i$ as the set of adjacent nodes of i in the communication graph, and denote its size as $|\mathcal{D}_i|$ which corresponds to the node *degree*. The existence of global unique identifiers to distinguish nodes is not considered, nor required. We only assume that each node is able to distinguish its neighbors.

We consider the execution of the aggregation algorithms in a synchronous model (as in Chapter 2 of [17]). Each round, executed in lockstep, is composed of two steps: message generation, where each node uses its local state to compute and send messages to its neighbors; and state transition, where each node uses its local state and the received messages to compute the new state.

We use this model in order to provide a fair comparison of the simulated algorithms. For example, each iteration in DRG, consisting of three phases with different kinds of messages sent in each one, will be three rounds.

We do not consider node failures, only link failures, in the algorithm and its evaluation. However, in Section 4.3 we briefly discuss why the *Flow Updating* algorithm is suitable to be adapted to cope with node failures and be used in asynchronous systems.

3.2 Key idea

Flow Updating is a novel averaging based aggregation algorithm, which enables the computation of aggregation functions (e.g. AVERAGE, COUNT or SUM) over a distributed system. It works independently from the network communication topology, and it is robust against message loss, a common fault in relevant application scenarios, such as Wireless Sensor Networks (WSN).

This algorithm departs from current approaches, that send "mass" in messages (with message loss implying mass loss) and keep the current mass value in a variable. The key idea is to use the *flow* concept from graph theory (which serves as an abstraction for many things like water flow or electric current; see Chapter 6 of [18]), and instead of storing in each node the current average in a variable, compute it from the initial value and the contribution of the flows along edges to the neighbors:

$$a_i = v_i - \sum_{j \in \mathcal{D}_i} f_{ij}. \tag{1}$$

This can be read as: the current average in a node is the initial input value less the flows from the node to each neighbor. Here we are not concerned with classic network flow concepts like capacity or trying to maximize flows; we focus on exploring the symmetry property of the flow along an edge:

$$f_{ij} = -f_{ji}. \tag{2}$$

This says that the flow from node i to node j is the symmetrical of the flow from node j to node i (see Figure 1(a)). The essence of the algorithm is: each node i stores the flow f_{ij} to each neighbor j; node i sends flow f_{ij} to j in a message; a node j receiving f_{ij} updates its variable f_{ji} with $-f_{ij}$. Messages simply update flows, being idempotent; the value in a subsequent message overwrites the previous one, it does not add to the previous value.

If the symmetry property of flows holds, the sum of the averages for all nodes (the global mass) will remain constant:

$$\sum_{i \in \mathcal{V}} a_i = \sum_{i \in \mathcal{V}} (v_i - \sum_{j \in \mathcal{D}_i} f_{ij}) = \sum_{i \in \mathcal{V}} v_i. \tag{3}$$

The intuition is that if a message is lost the symmetry is temporarily broken, but as long as a successful messages arrives, it re-establishes the symmetry (see Figure 1(b) and 1(c)). What really happens, due to interleaving, is that the symmetry may never hold but f_{ij} converges to $-f_{ji}$, and the global mass converges to the sum of the input values of all nodes.

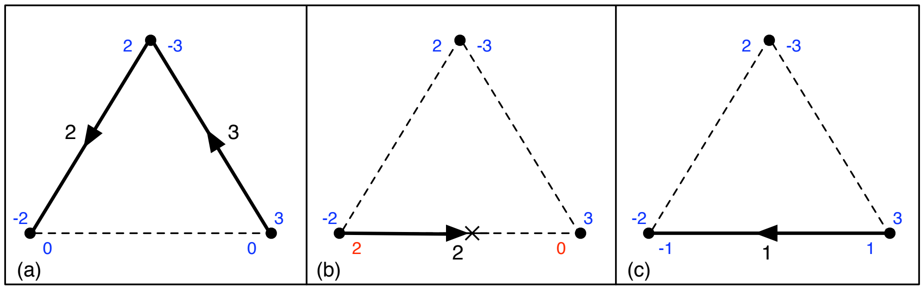


Fig. 1. Key concept of *Flow Updating*, illustrated by arbitrary flow exchanges between 3 nodes, considering a temporary link failure in (b).

3.3 Algorithm

Algorithm 1 shows *Flow Updating*, according to the defined system model (Section 3.1). In this algorithm, the local state of each node i will keep a variable v_i with the local input value, the individual flows f_{ij} toward its neighbors, and the estimated aggregates e_{ij} of the neighborhood. Initially, at each node i the neighbors flows and estimates are set to zero ($f_{ij} = 0$ and $e_{ij} = 0$ for all $j \in \mathcal{D}_i$), and v_i to the local input value to aggregate.

```
state variables:
    $f_{ij}$, $\forall j \in \mathcal{D}_i$, flows, initially $f_{ij} = 0$
    $e_{ij}$, $\forall j \in \mathcal{D}_i$, estimates, initially $e_{ij} = 0$
    $v_i$, input value

message-generation function:
    $\mathrm{msg}(i, j) = (f_{ij}, e_{ij})$, $\forall j \in \mathcal{D}_i$

state-transition function:
    forall $(f_{ji}, e_{ji})$ received do
        $f_{ij} \leftarrow -f_{ji}$
        $e_{ij} \leftarrow e_{ji}$
    $e_i \leftarrow \frac{\left(v_i - \sum_{j \in \mathcal{D}_i} f_{ij}\right) + \sum_{j \in \mathcal{D}_i} e_{ij}}{|\mathcal{D}_i| + 1}$
    forall $j \in \mathcal{D}_i$ do
        $f_{ij} \leftarrow f_{ij} + (e_i - e_{ij})$
        $e_{ij} \leftarrow e_i$
```

Algorithm 1: *Flow Updating* algorithm (at each node i).

In the message generation step a pair of values (flow and estimate) is created for each neighbor j, to be sent by i. An individual flow value f_{ij} is assigned and send to each neighbor, while the same estimate e_i is send by i to all its neighbors (both calculated and locally stored at the end of the previous round).

In the state transition step, each node starts by setting the local values associated to the sender of each received message with the ones (estimate and symmetric flow) within the message pair addressed to him. Notice that, different estimates (and corresponding flows) may be received from different neighbors. Thereafter, each node computes a new prediction of the aggregation value e_i by averaging the received estimates and the one locally calculated by (1), and updates its state accordingly, in order to produce the new result. To do so, to the flow f_{ij} is added the difference between the new estimate e_i and the received estimate from j; the estimates e_{ij} of all neighbors j are set directly with the new foreseen estimate e_i. The newly computed state will be used in the next round to generate and send the proper data values to all neighbors, in order to lead them to the same estimate. The iterative execution of this algorithm across all the network allows the convergence of the value estimated at each node to the correct global average of the input values.

```
state variables:
    $f_{ij}$, $\forall j \in \mathcal{D}_i$, flows, initially $f_{ij} = 0$
    $e_{ij}$, $\forall j \in \mathcal{D}_i$, estimates, initially $e_{ij} = 0$
    $v_i$, input value
    $k$, chosen neighbor

message-generation function:
    $\mathrm{msg}(i, k) = (f_{ik}, e_{ik})$

state-transition function:
    forall $(f_{ji}, e_{ji})$ received do
        $f_{ij} \leftarrow -f_{ji}$
        $e_{ij} \leftarrow e_{ji}$
    $e_i \leftarrow \frac{\left(v_i - \sum_{j \in \mathcal{D}_i} f_{ij}\right) + \sum_{j \in \mathcal{D}_i} e_{ij}}{|\mathcal{D}_i| + 1}$
    $k \leftarrow chooseNeighbor(\mathcal{D}_i)$;
    $f_{ik} \leftarrow f_{ik} + (e_i - e_{ik})$
    $e_{ik} \leftarrow e_i$
```

Algorithm 2: Unicast version of *Flow Updating* (at each node i).

Notice that, in practice, when broadcast is supported by the physical communication medium, all the messages generated at each round by a node can be sent in a single transmission to all neighbors. If broadcast is not physically supported, the messages are individually transmitted to each adjacent node. According to this, and in order to supply an impartial evaluation of this algorithm, when compared with others that do not take advantage of message broadcast (e.g. push-sum protocol), we defined an unicast version of *Flow Updating*. The differences of this variation of the algorithm are depicted by Algorithm 2, mainly consisting in the addition of a function $chooseNeighbor(\mathcal{D}_i)$ to choose a specific target k from the set of neighbors of node i. In the state transition process, only the flow and estimate corresponding to the chosen node k will be updated, instead of all neighbor nodes. Afterwards, in the next round, a single message will be generated and sent to the previously chosen node k.

Several heuristics can be used to implement the function $chooseNeighbor(\mathcal{D}_i)$. For instance, the node can be simply picked up uniformly at random from the set of neighbors $\mathcal{D}_i$, or it can be chosen in accordance to a specific criteria, taking advantage of the neighbors data locally available. In particular, in the evaluated unicast version of the algorithm, we consider a criteria in which the neighbor possessing the estimate with the greater discrepancy relatively to the averaged estimate e_i will be selected in each round.

Considering the system settings in which the algorithm is executed, the choice of the correct heuristic to select the target neighbor in each round may improve the overall performance of the aggregation process. Some simulations comparing both of the the referred heuristics have evidenced an improved performance of the unicast version of the algorithm using the latter criteria, instead of using a naif random choice. This optimization study is out of the scope of this paper.

4 Evaluation

4.1 Simulation Setup

We prepared a simulation environment compliant with the system model enunciated in Section 3.1, in order to allow comparisons between Flow Updating and two established approaches: the push-sum protocol [7] and DRG [9]. We evaluated all the aggregation algorithms under strictly identical simulation settings (same network topologies and initial distribution of input values), aiming for an impartial and fair comparison between them. Two different network topologies were taken into account for simulation purposes: *random* and *2D/mesh*. The random network fits the Erdős–Rényi model [19] and consists on a connected network in which all nodes are randomly linked to each other (according to a predefined average connection degree). The 2D/mesh network defines a connected network in which the communication links are established according to geographical proximity, modeling an approximation to the connectivity in WSN. Nodes are spread uniformly at random and links are set within a given fixed radius. Independently from the topology, along the algorithms execution the network remains static (no link changes, and no nodes arriving or leaving).

In order to evaluate the robustness of *Flow Updating*, we consider that each message sent in each round can be lost according to a predefined probability.

The same aggregation function is computed by all algorithms: COUNT, which can be used to determine the network size of the system. Shown results correspond to the average value obtained from 50 repetitions of the execution of the same algorithm under identical simulation settings, using different generated networks with the same characteristics in each repetition. Two main metrics are used to evaluate each aggregation algorithm: speed and overhead. The first criteria defines how fast, in number of rounds, a given accuracy is reached. Accuracy is expressed by the normalized RMSE (Root Mean Square Error) of the estimate when contrasted to the target value. The second criteria is defined in terms of the number of messages required to compute the aggregation result with the desired accuracy. This message overhead can be interpreted as an approximation to energy expenditure in WSN, since message transmission is often the dominating factor in those settings.

4.2 Results

The first scenario corresponds to a random network with 1000 nodes ($n = 1000$) and an average connection degree d approximately equal to $\log n$ ($= 3$). The same network size ($n = 1000$) and degree ($d \approx \log n$) are considered in the second scenario, but a different network topology is used: 2D/mesh. We choose to use a degree of value $\log n$, since it is the degree value that nodes must have in order to keep the network connected with constant probability, considering that all nodes fail with a probability of 0.5 [20]. Notice that all specific parameters of the analyzed algorithms (e.g. the probability to become leader in DRG) have been tuned to provide them with the best performance in each scenario.

For a matter of convenience, to reduce the space consumed by graphics, we depict unicast and broadcast algorithms in the same figure. Nevertheless, in order to perform a correct analysis of the obtained results, approaches using different communication assumptions should be observed separately. Concretely, the push-sum protocol should be contrasted with the unicast version of *Flow Updating*, and DRG with the broadcast version. The simulation results obtained for the outlined network topologies (random and 2D/mesh) are respectively depicted by Figure 2 and Figure 3.

In the random network scenario, the two versions of *Flow Updating* clearly outperform both competitors, both in terms of convergence speed Figure 2(a) (from 4 to more than 10 times faster), and resource consumption (sending considerably less messages) Figure 2(b).

As expected, the 2D/mesh topology penalizes the convergence rate in all algorithms. Again, both *Flow Updating* variations (unicast and broadcast) reach better results than the ones obtained by the other aggregation algorithms (Figure 3). The broadcast version of *Flow Updating* shows the best overall results both in speed and overhead.

We now concentrate on the behavior of *Flow Updating* under message loss, since under this failure pattern the other algorithms (without extensions) do not converge to the correct value. Here we compare three levels of message loss affecting transmitted messages, 0% (no loss), 20% and 40%. This values are compared in a random network running the unicast version and in a 2D/mesh network using the broadcast version of the algorithm. These combinations should

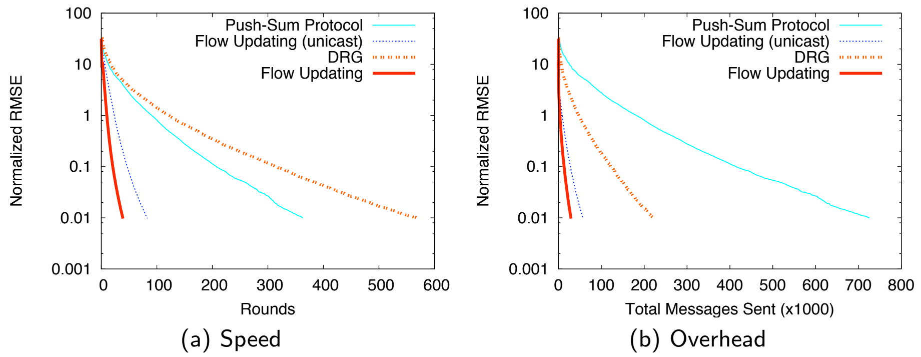


Fig. 2. Random networks, $n = 1000$, and $d \approx 3$ $(\log n)$.

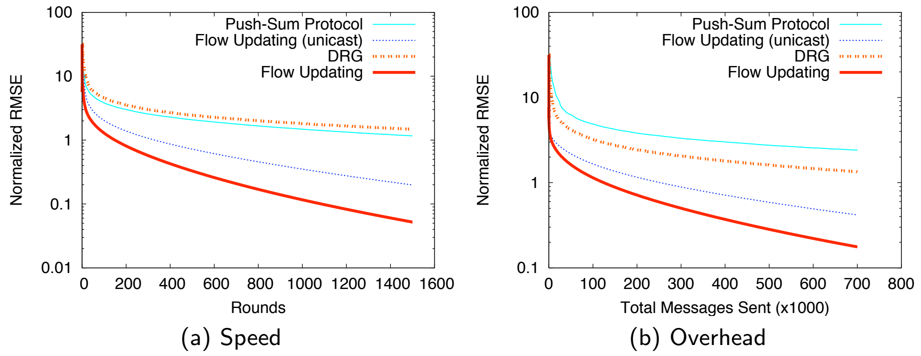
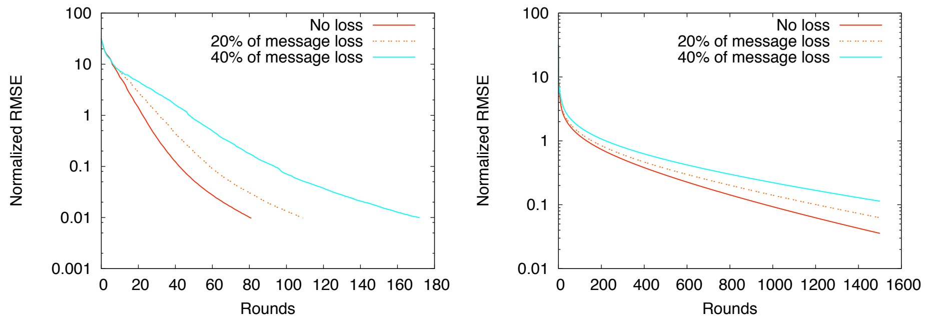


Fig. 3. 2D/mesh networks, $n = 1000$, and $d \approx 3$ $(\log n)$.

reflect the more commonly available communication capabilities in each topology. Although not shown in the figures, we confirmed that the broadcast version is always better than the unicast, so that when possible broadcast should be preferred.

Figure 4 shows a degradation of the performance of the algorithm proportional to each fault rate. The results show that message loss does not prevent convergence of the estimate, it only increases the time and messages needed to reach it.

It is curious to observe that even under the occurrence of high amounts of message loss *Flow Updating* can still outperform the classical algorithms operating under no message loss. Notice that, comparing the results of Figure 2(a) against Figure 4(a) for the random network scenario, and the results of Figure 3(a) versus Figure 4(b) for the 2D/mesh network, even considering a substantial amount of faults (40%) on both versions of *Flow Updating*, they outperform the other approaches without faults.



(a) Random network (unicast version) (b) 2D/mesh network (broadcast version)

Fig. 4. *Flow Updating* fault tolerance – $n = 1000$, and $d \approx 3$ $(\log n)$.

4.3 Discussion

The obtained results reveal a significantly greater performance of *Flow Updating* on all evaluated scenarios, both in terms of speed and overhead, when compared to well-known approaches. Most important, *Flow Updating* is naturally robust against message loss, due to its flow exchange scheme that keeps the aggregation inputs unchanged along the execution of the algorithm.

Fault tolerance has not been a key concern in the design of aggregation by averaging. Typically, existing approaches require the use of additional mechanisms in order to tolerate faults. For instance, G-GAP [10] extended the push-sum protocol by explicitly acknowledging mass and computing recovery shares, in order to support discontinuous failures of adjacent nodes within a short time period. As we pointed out, failure detection is trickier in this setting, since wrong assessments will lead to deviations in the total mass and ruin convergence. Additionally, the use of acknowledgment messages and timeouts leads to extra consumption of resources and time, adding to the overall overhead of the underlying algorithm even if no faults occur. In contrast, *Flow Updating* is per se tolerant to message loss faults, and its performance is only affected when faults do occur.

In presenting and evaluating *Flow Updating* we have focused on message loss (transient link failures). However, the basic algorithm can be trivially extended to cope with both permanent link failures and node failures. In fact, even though it can be impossible in a distributed system to distinguish between these types of failure, *Flow Updating* can tolerate both without having to make a distinction. The principle is quite simple: if a link/neighbor is suspected to have permanently failed (e.g. because no message arrived for some rounds), the entries (flow and estimate) regarding that neighbor are removed from the state. For all purposes the flow will converge as if that link or node did not exist. If the suspicion turns out to be wrong because a message arrives, the state can be simply augmented using the message content. Even if the network becomes partitioned, each partition will be aggregated independently (in the case of counting, only the partition having the 1 initial value will be counted).

A similar strategy could also be applied, in order to cope with dynamic changes of the network structure, adding or removing the local neighbors information whenever a node is arriving or leaving the neighborhood. Even in this case, the flow values should adjust and all the estimates should converge to a correct value, continuously adapting to reach the network equilibrium. The tolerance exhibited by *Flow Updating* when working in adverse scenarios prone to considerable amounts of message loss, suggest that the algorithm can be successfully used in practice on asynchronous and dynamic systems. We leave the analysis and study of *Flow Updating* in this kind of setting for future work.

5 Conclusion

The main contribution of this paper consists on the introduction of a new distributed data aggregation approach: *Flow Updating*. An empirical analysis of the proposed algorithm is provided, comparing it with existing approaches under identical simulation settings. *Flow Updating* was shown to be robust against message loss, overcoming the problem of "mass" loss verified on existing averaging algorithms, and can be easily extended to tolerate permanent link failures or node failures. Moreover, the results obtained reveal that our algorithm performs better than its peers, requiring less time and communication resources.

Flow Updating allows the accurate computation of aggregates at all nodes, converging to the exact result along time. The algorithm execution is independent from the network routing topology. The "flow" exchange scheme implemented by this algorithm enables the execution of idempotent update operations, which is the key to its unique robustness capabilities.

Averaging based aggregation is of special relevance when seeking high accuracy estimates of a given global property in a distributed system. The proposed approach can be used on overlay networks with unicast, and is particularly efficient when broadcast capabilities are present. The broadcast version of the algorithm can bring a robust implementation of aggregation by averaging to wireless sensor networks, where a high level of message loss can be expected.

References

1. Robbert Van Renesse. The importance of aggregation. *Future Directions in Distributed Computing, Lecture Notes in Computer Science*, 2584:87–92, 2003.
2. Ion Stoica, Robert Morris, David Karger, M Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 149–160, Aug 2001.
3. Ittai Abraham and Dahlia Malkhi. Probabilistic quorums for dynamic systems. *Distributed Computing, Springer Berlin/Heidelberg*, 18(2):113–124, Dec 2005.
4. Samuel Madden, Michael Franklin, Joseph Hellerstein, and Wei Hong. TAG: a Tiny AGgregation service for ad-hoc sensor networks. *ACM SIGOPS Operating Systems Review*, 36(SI):131–146, Dec 2002.

5. J Li, K Sollins, and D Lim. Implementing aggregation and broadcast over distributed hash tables. *ACM SIGCOMM Computer Communication Review*, 35(1):81–92, 2005.
6. Yitzhak Birk, Idit Keidar, Liran Liss, Assaf Schuster, and Ran Wolff. Veracity radius: capturing the locality of distributed computations. *PODC '06: Proceedings of the twenty-fifth annual ACM symposium on Principles of distributed computing*, Jul 2006.
7. D Kempe, A Dobra, and J Gehrke. Gossip-based computation of aggregate information. *Foundations of Computer Science, 2003. Proceedings. 44th Annual IEEE Symposium on*, pages 482– 491, 2003.
8. M Jelasity, A Montresor, and O Babaoglu. Gossip-based aggregation in large dynamic networks. *ACM Transactions on Computer Systems (TOCS)*, 2005.
9. Jen-Yeu Chen, G Pandurangan, and Dongyan Xu;. Robust computation of aggregates in wireless sensor networks: Distributed randomized algorithms and analysis. *IEEE Transactions on Parallel and Distributed Systems*, 17(9):987 – 1000, Sep 2006.
10. Fetahi Wuhib, Mads Dam, Rolf Stadler, and Alexander Clemm. Robust monitoring of network-wide aggregates through gossiping. *10th IFIP/IEEE International Symposium on Integrated Network Management*, pages 226 – 235, 2007.
11. Carlos Baquero, Paulo Sérgio Almeida, and Raquel Menezes. Fast estimation of aggregates in unstructured networks. In *International Conference on Autonomic and Autonomous Systems (ICAS)*, Valencia, Spain, Apr 2009. IEEE Computer Society.
12. D Mosk-Aoyama and D Shah. Computing separable functions via gossip. *PODC '06: Proceedings of the twenty-fifth annual ACM symposium on Principles of Distributed Computing*, pages 113–122, 2006.
13. D Kostoulas, D Psaltoulis, Indranil Gupta, K Birman, and Al Demers. Decentralized schemes for size estimation in large and dynamic groups. *Fourth IEEE International Symposium on Network Computing and Applications*, pages 41–48, 2005.
14. Laurent Massoulié, Erwan Merrer, Anne-Marie Kermarrec, and Ayalvadi Ganesh. Peer counting and sampling in overlay networks: random walk methods. *PODC 06: Proceedings of the twenty-fifth annual ACM symposium on Principles of Distributed Computing*, Jul 2006.
15. A Ganesh, A Kermarrec, E Le Merrer, and L Massoulié. Peer counting and sampling in overlay networks based on random walks. *Distributed Computing*, 20(4):267–278, 2007.
16. M Jelasity and A Montresor. Epidemic-style proactive aggregation in large overlay networks. *24th International Conference on Distributed Computing Systems*, 2004.
17. Nancy A. Lynch. *Distributed Algorithms.* Morgan Kaufmann Publishers Inc., 1996.
18. Reinhard Diestel. *Graph Theory*, volume 173 of *Graduate Texts in Mathematics.* Springer, 3 edition, 2005.
19. Paul Erdős and Alfréd Rényi. On the evolution of random graphs. *Publications of the Mathematical Institute of the Hungarian Academy of Sciences*, 5:17–61, 1960.
20. M. Frans Kaashoek and David R. Karger. Koorde: A simple degree-optimal distributed hash table. In M. Frans Kaashoek and Ion Stoica, editors, *IPTPS*, volume 2735 of *Lecture Notes in Computer Science*, pages 98–107. Springer, 2003.