Fault-Tolerant Aggregation for Dynamic Networks

Paulo Jesus Carlos Baquero Paulo Sérgio Almeida

Departamento de Informática (CCTC-DI)
Universidade do Minho
Braga, Portugal
{*pcoj, cbm, psa*}*@di.uminho.pt*

***Abstract*—Data aggregation is a fundamental building block of modern distributed systems. Averaging based approaches, commonly designated gossip-based, are an important class of aggregation algorithms as they allow all nodes to produce a result, converge to any required accuracy, and work independently from the network topology. However, existing approaches exhibit many dependability issues when used in faulty and dynamic environments. This paper extends our own technique, Flow Updating, which is immune to message loss, to operate in dynamic networks, improving its fault tolerance characteristics. Experimental results show that the novel version of Flow Updating vastly outperforms previous averaging algorithms; it self adapts to churn without requiring any periodic restart, supporting node crashes and high levels of message loss.**

***Keywords*-distributed algorithms, data aggregation, fault-tolerance, dynamic networks**

I. Introduction

With the advent of multi-hop ad-hoc networks, sensor networks and large-scale overlay networks, there is a demand for tools that can abstract meaningful system properties from given assemblies of nodes. In such settings, aggregation plays an essential role in the design of distributed applications [1], allowing the determination of network-wide properties like network size, total storage capacity, average load, and majorities. Although apparently simple, in practice aggregation has reveled itself to be a non-trivial problem in distributed settings, where no single element holds a global view of the whole system.

Distributed data aggregation becomes particularly difficult to achieve when faults are taken into account (i.e. message loss and node crashes), and especially if dynamic settings are considered (nodes arriving/leaving). Few have approached the problem under these settings [2], [3], [4], [5], [6], [7], proving to be hard to efficiently obtain accurate and reliable aggregation results in faulty and dynamic environments.

Classical approaches, like TAG [2], perform a tree-based aggregation where partial aggregates are successively computed from child nodes to their parents until the root of the aggregation tree is reached (requiring the construction of a specific routing topology). This kind of aggregation technique is often applied in practice to Wireless Sensor Network (WSN) [8]. Other tree-based aggregation approaches can be found in [3], and [9]. We should point out that, although being energy-efficient, the reliability of these approaches may be strongly affected by the inherent presence of single-points of failure in the aggregation structure.

Sampling techniques [4], [10], [5] are independent from the routing topology, but are not accurate. Their result is affected by an estimation error, even in fault-free scenarios, depending on the quality of the collected samples and the applied estimation method. Moreover, samples are collected at a single node and can incur in a considerable delay, especially if the probing message is lost. The estimation error can reach 20% in Sample & Collide [10], [4].

A useful class of high accuracy aggregation algorithms is based on *averaging* techniques [11], [6], [12], [13]. Such algorithms start from a set of input values spread across the network nodes, and iteratively average their values with neighbors. Eventually all nodes will converge to the same value and can estimate some useful metric. Averaging techniques allow the derivation of different aggregation functions besides average (like counting and summing), according to the initial combinations of input values. These techniques are thought to be robust and accurate (converge over time), when compared to other aggregation techniques, but in practice they exhibit relevant problems that have been overlooked, not supporting message loss nor node crashes (see [14] for more details).

The main contribution in this article is to extend our own averaging approach, *Flow Updating* [15], which is immune to message loss, in order to handle node arrival and departure/crash. We evaluate the new approach, and show that it is fault-tolerant and able to efficiently support network dynamism, exhibiting self-stabilizing properties.

The remainder of this paper is organized as follows. Section II describes a new version of the *Flow Updating* algorithm able to work in dynamic networks. In Section III, we evaluate the proposed approach using simulation. In Section IV we point to directions of further research. Finally, we make some concluding remarks in Section V.

II. Flow Updating for Dynamic Networks

Flow Updating [15] is a recent averaging based aggregation approach, which works for any network topology and tolerates faults. Like existing gossip-based approaches, it averages values iteratively during the aggregation process towards converging to the global network average. But unlike them, it is based on the concept of *flow*, providing unique fault-tolerant characteristics by performing idempotent updates.

Here we extend Flow Updating to perform robust data aggregation on dynamic networks. The extension is based on maintaining a dynamic mapping of flows according to the current set of neighbors: removing the entries relative to leaving (or crashing) nodes, and adding entries for newly arrived nodes. The averaging process in each node uses only the current set of neighbors. This straightforward modification allows Flow Updating to cope with node departure/crash and node arrival, extending its fault tolerance properties to these scenarios.

Node departure, crash or arrival are modeled by a failure detector that gives for each node at each round the set of neighbors considered to be alive. The interesting thing is that the extension of Flow Updating will allow the use of practical implementations of failure detectors, that can be incorrect many times (falsely suspecting correct nodes or vice-versa) without compromising the correctness.

New nodes are immediately allowed to participate in the averaging process, and leaving or crashed nodes implicitly stop participating in it. The algorithm runs continuously, without requiring restarts in order to adapt to network changes/failures, and at each round simply makes use of the set of neighbors n_i given by the failure detector. No concept of *epoch* is required and the algorithm is always converging towards the average according to the current set of participants, allowing a fast self adaptation to network changes. Another advantage of Flow Updating is that it also allows the value to be aggregated v_i to change over time (e.g. a temperature). Again, the algorithm will converge to the aggregation of the most recent value at each node without requiring a restart. In the algorithm, presented in Figure 1, we call these values that can be read at each round, but are not updated by the algorithm, the *inputs*.

1: **inputs:**
2: v_i, value to aggregate
3: n_i, set of neighbors given by failure detector
4:
5: **state variables:**
6: flows: initially, $F_i = \{\}$
7:
8: **message-generation function:**
9: $\text{msg}_i(F_i, j) = (i, f, \text{est}(v_i, F_i))$
10: **with**
11: $f = \begin{cases} F_i(j) & \text{if } (j, \_) \in F_i \\ 0 & \text{otherwise} \end{cases}$
12:
13: **state-transition function:**
14: $\text{trans}_i(F_i, M_i) = F_i'$
15: **with**
16: $F = \{j \mapsto -f \mid j \in n_i \wedge (j, f, \_) \in M_i\} \cup$
17: $\{j \mapsto f \mid j \in n_i \wedge (j, \_, \_) \notin M_i \wedge (j, f) \in F_i\}$
18: $E = \{i \mapsto \text{est}(v_i, F)\} \cup$
19: $\{j \mapsto e \mid j \in n_i \wedge (j, \_, e) \in M_i\} \cup$
20: $\{j \mapsto \text{est}(v_i, F_i) \mid j \in n_i \wedge (j, \_, \_) \notin M_i\}$
21: $a = (\sum\{e \mid (\_, e) \in E\}) / |E|$
22: $F_i' = \{j \mapsto f + a - E(j) \mid (j, f) \in F\}$
23:
24: **estimation function:**
25: $\text{est}(v, F) = v - \sum\{f \mid (\_, f) \in F\}$
26:

Figure 1. Flow Updating algorithm for dynamic networks.

A. Algorithm

The algorithm is presented under the synchronous network model (as in Chapter 2 of [16]). Computation proceeds in synchronous rounds. At each round, first nodes look at their state and compute what messages are sent, through a *message-generation function*; then nodes take their state and the messages received and compute a new state, through a *state-transition function*. Each node needs only to be able to distinguish its neighbors, not requiring the use of globally unique identifiers.

The state of each node i consists of a mapping F_i from node ids to flows; it stores, for each current neighbor, the flow along the edge to that node.

A single type of message is sent, containing the self id i, the flow $F_i(j)$ to each current neighbor j, and the aggregate estimate (lines 9–11). When no flow value is available for a given neighbor, initially or when a new node starts participating, the value 0 is used. The estimate is computed by making use of the *estimation function* (line 25).

The state-transition function (lines 14-22) takes a state F_i and the set of messages M_i received by the node in the round, and returns a new state F_i'. We make use of some auxiliary variables to compute the new state: the flows F updated according to the messages received, and the estimates E (last received) used to compute the new average a. Looking at these values in more detail:

- F is a mapping from current neighbor ids to: the symmetric of the flow in messages, for those neighbors that sent messages successfully; the current value, if any, in the case of message loss.
- E is a mapping from node ids to estimates: for the self node i, according to the estimation function, using the newly updated flows in F; for neighbors whose

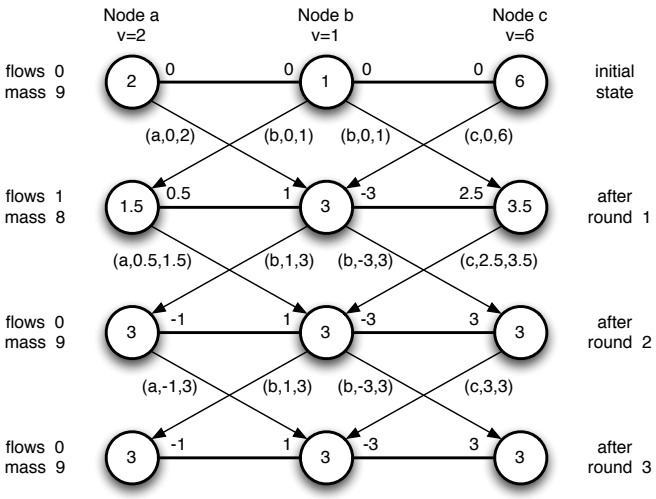

Figure 2. Example of a simple *Flow Updating* run, without faults.

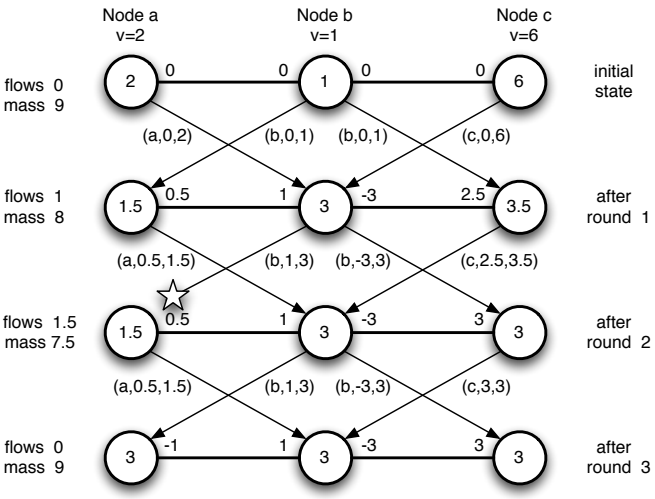

Figure 3. Example of a *Flow Updating* run with a message loss.

messages arrived, the estimate sent; otherwise, the estimate according to the estimation function, using the flows at the beginning of the round (this is the estimate sent to all neighbors at the beginning of the round).

- a is simply the average of the estimates in the mapping E, and represents the new estimate towards which the node will lead its neighbors to converge in the next round.

Finally, the new mapping F_i' is calculated by adjusting each flow in F so that the estimates move towards a: the estimate for node i in the beginning of next round will be a; each neighbor would compute a as its estimate at the end of the next round if it did not receive other messages from its own neighbors (e.g. if it has node i as its only neighbor).

B. Example Run

We now illustrate the execution of the algorithm in a simple path network of three nodes, $\{a, b, c\}$, with two links, $\{(a, b), (b, c)\}$. First we consider a run, in Figure 2, where no failures occur. Rounds start executing from an initial state where each node estimate is the input value (2, 1, and 6, respectively), and no flows are registered, defaulting to 0. When starting round 1 nodes generate the messages to their neighbors and send them (e.g. node a sends message $(a, 0, 2)$ to node b, illustrated by a diagonal arrow). Then, they inspect their state and messages received and apply the *state-transition* function. In the second row in the figure we illustrate final values at the end of round 1. Flows are shown close to the node, in the direction of the respective neighbor and are according to F_i'. The estimates, depicted in the center of the node circle, reflect $\text{est}(v_i, F_i')$. For each round we also indicate the sum of flows across all nodes (labeled *flows*) and the the sum of all node estimates (labeled *mass*). One can see that when the system converges flows add up to 0, the mass depicts the sum of the input values, and all nodes estimate the same value.

In this particular run, in Figure 2, the system converges after round 2. After this round the flows are skew symmetric and all nodes correctly estimate an average of 3. This fast and precise convergence in a short number of rounds is possible, in flow update, due to the simplicity of this network and the absence of cycles in the graph. However, even in this simple path graph, other averaging algorithms (e.g. [11] and [6]) would take a large number of rounds to reach a high level of accuracy.

We now consider the effect of loosing one message. In Figure 3 a message from node b to a sent in round 2 is lost. In this case, we assume that the failure detector in a does not mark node b as failed (one still has $n_a = \{b\}$), and thus one can observe the impact of a single message loss. We now describe the actions in node a at round 2. According to the algorithm (line 17), if a neighbor is considered active by the failure detector but its message has not arrived, then the stored flow value is used (in this case 0.5). Similarly, the algorithm (line 20) uses the estimate at the beginning of the round for nodes whose messages have not arrived (in this case 1.5); this corresponds to the estimate at the end of the previous round, which is also the value sent to neighbors in the current round. Since no message is received by node a, the same estimate is used for both nodes to compute the new average, keeping it unchanged as $(1.5 + 1.5)/2 = 1.5$. Since node a estimate is unaltered, so is the flow towards node b $(0.5 + 1.5 - 1.5 = 0.5)$. Overall, the impact of this single lost message was simply to delay convergence to the end of round 3.

Now we consider an execution where a node fails permanently and this is detected by the neighbor node. In this case, in Figure 4, node c fails at round 1 and this is detected by its neighbor node b, from round 2 onwards. After this failure

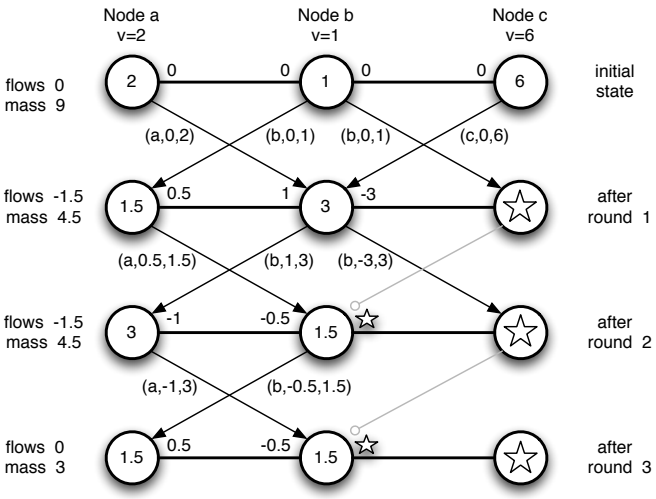

Figure 4. Example of a *Flow Updating* run with a node crash/departure.

nodes a and b should no longer converge to a common estimate of 3 but instead to 1.5, the average of the input values at the remaining nodes. In node b, at round 2 node c no longer belongs to n_b ($n_b = \{a\}$), thus according to the algorithm (lines 16-17) its previous flow does not get into the mapping F. Likewise, its estimate is not incorporated in E and the target average, line 21, now only considers nodes a and b. This changes in node b will indirectly lead node a to adjust its flows; convergence of the two remaining nodes to the new target will be at the end of round 3.

One should also note that, if node c becomes active again, sends messages to b, and eventually is again considered an active neighbor in n_b, then this leads to a rebalancing of the average that again considers the input value in c. This plasticity allows the algorithm to quickly adapt to changes in active membership due to churn, as well as message loss.

III. Evaluation

In this section, we provide some experimental results to evaluate *Flow Updating* under demanding faulty scenarios, with both churn and message loss. We also compare it with an existing average based technique intended to be able to operate in dynamic networks, more specifically *Push-Pull Gossiping* [6]. For that purpose, we use a custom high level simulator, operating in synchronous rounds, to compute the COUNT aggregation function (determination of network size). Convergence speed depends on the initial data distribution across the network; COUNT represents an extreme scenario where only one node starts with the value 1 and all others with 0. We chose to use this aggregation function for evaluation because it is the one with the worst performance. The algorithm will perform better when computing an AVERAGE of uniformly distributed input values. SUM will have the same performance as COUNT, as it is computed by combining AVERAGE and COUNT.

We consider two different network topologies: random (generated according to the Erdős–Rényi model [17]), and 2D/mesh (geographical networks, with random uniform node placement, where communication links are established according to a predefined radio communication range, an approximation to the topologies occurring in WSN). The results for each scenario are drawn from 30 trials of the execution of the algorithms under identical settings. In each trial different randomly generated networks with the same characteristics (topology, size and average degree) are used. All networks considered in every scenario start with the same size ($n = 1000$), and the same approximated average connection degree ($d \approx 2 \log n$). Notice that according to [18], $\log n$ is the degree 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. However, we choose to be more conservative and use a value twice higher, in order to avoid significant network partitioning when testing failure of one quarter of the nodes.

We start by considering a random network scenario, when subject to both drastic and continuous changes of the network membership. For this purpose, we successively applied a sudden departure (catastrophic crash) and arrival of 25% of the initial nodes, followed by an arrival and departure of the same portion of nodes at a constant rate (10 nodes per round). For a matter of clarity, a stability period of 50 rounds is introduced between each network change.

First, we compare *Flow Updating* (FU) with *Push-Pull Gossiping*[6] (PPG), and *Push-Pull Ordered Wait* [14] (PPOW is a fix of PPG that solves its atomicity problems), in the described random dynamic network scenario without message loss. PPG implements a restart mechanism to cope with churn, starting a new instance of the algorithm after a predefined number of rounds (epoch), and prevents new nodes from participating in the current epoch. Similarly to PPG, PPOW was extended with a restart mechanism, but instead of delaying the participation of new nodes to the next epoch, joining nodes are allowed to immediately participate. This modification was applied since it yielded more favorable results to PPOW in all performed experiments.

Figure 5 shows the results obtained, using an epoch length of 50 rounds. We can observe that an overestimate is produced by PPG due to its atomicity problems, even without network changes (e.g. between round 0 and 50), which is solved by PPOW that converges to the expected value. Most important, these results expose the effect of the restart mechanism, that introduces an undesirable delay to respond to network change. Note that, this delay is also observed even if only the estimate at the end of each epoch are considered as valid (points at the end of each PPG and PPOW epoch, every 50 rounds). In the particular case of PPG, the delay is present in both node departure and arrival. However, in PPOW the response time to changes is reduced in the case of nodes arrival by allowing joining nodes to

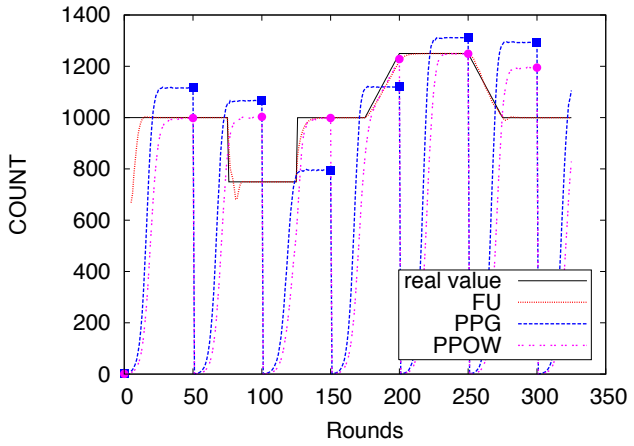

Figure 5. Comparison of Flow Updating (FU), Push-Pull Gossiping (PPG) and Push-Pull Ordered Wait (PPOW): Average of estimates in a dynamic random network, with no message loss.

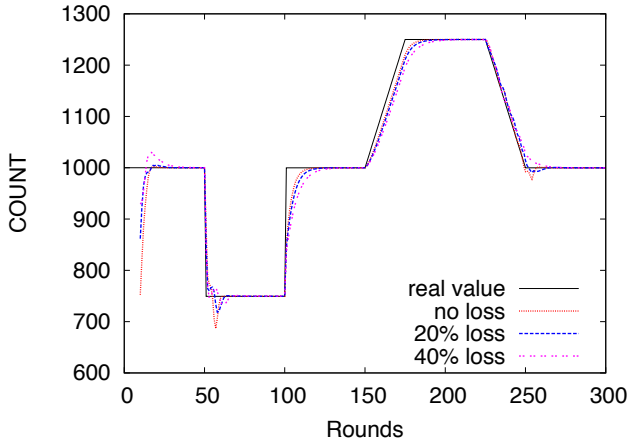

Figure 6. Average of estimates in a dynamic random network with message loss.

immediately participate in the current epoch.

The restart mechanism introduces a trade-off between the response time to network change and the accuracy of the push-pull algorithms, preventing them from following the network change with high accuracy. In contrast, FU is able to closely follow the network changes without requiring any restart mechanism. FU clearly outperforms the other approaches (PPG and PPOW) which are unable to adapt to network changes. For this reason, the remainder of the evaluation focuses exclusively on Flow Updating.

We now evaluate the behavior of FU on the same dynamic random network. But besides churn, we also consider that each received message (at each round) can be lost according to a given probability. Figure 6 shows more clearly that the mean of the estimates produced by FU closely follows the network changes. Moreover, from Figures 6 and 7, we observe that message loss, even in considerable amounts

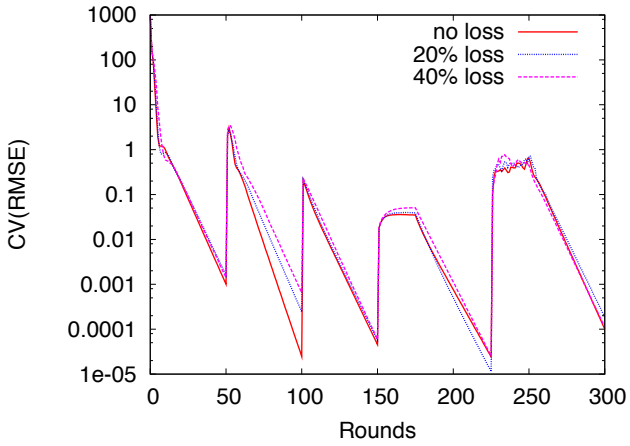

Figure 7. Coefficient of variation of the RMSE in a dynamic random network with message loss.

20% and 40%, only slightly affects convergence speed and the ability of the algorithm to cope with churn.

Curiously, in some situations the algorithm even benefits from message loss, increasing its convergence speed (e.g. 20% loss in round 175 to 225 of Figure 7). We found out that it is possible to increase the convergence speed of the algorithm by "deactivating" some communication links. This deactivation also provides a considerable reduction on the number of messages required to reach a given accuracy level. In some cases, message loss reproduces this effect. We are currently exploring mechanisms to control this feature [19], although a thorough study of this additional source of convergence speedups is left for future work.

Figure 7 shows the coefficient of variation of the root mean square error[1] along time, allowing the observation of the global accuracy variation due to network dynamism. This metric compares each individual estimate against the actual network size, as perceived by an external observer that can inspect the whole network in 0 rounds. This is a very demanding metric, since in any actual distributed algorithm nodes would have a delay proportional to diameter rounds before knowing the network size.

The results confirm the fast convergence of the algorithm during stability periods, and show expected accuracy decreases (increase of the CV(RMSE)) resulting from network changes. Brutal changes lead to momentary perturbations which are rapidly reduced, while continuous changes will provoke an accuracy reduction that persists during the continuous churn time period. In this particular case, for the considered churn rate (10 nodes per round), the arrival of nodes will increase the global error from less than 0.01% to about 3.5%, and node departures will increase it from

[1]Root of the mean squared differences between the estimate e_i at each node i and the correct result e, divided by the correct result: $\frac{1}{e}\sqrt{\frac{1}{n}\sum_{i=1}^{n}(e_i - e)^2}$

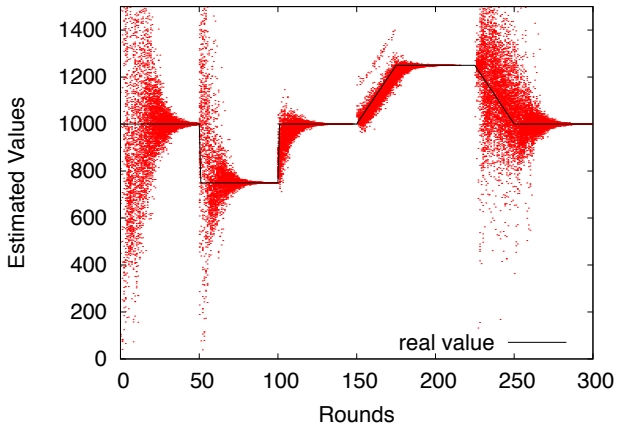

Figure 8. Estimates distribution in a dynamic random network with 20% of message loss.

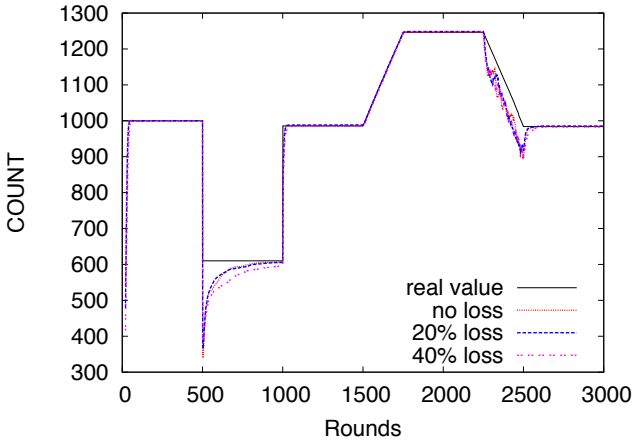

Figure 9. Average of estimates in a dynamic 2D/mesh network with message loss.

less than 0.01% to about 50%. Node departure (or crashes) induce higher perturbations than node arrivals; in both cases the higher the number of nodes involved the bigger is the impact on node estimation accuracy. The effect of churn on each node estimate is clearly depicted by Figure 8, which shows the distribution of individual estimates along time, considering 20% of message loss[2].

The previous simulation scenarios are now applied using 2D/mesh network topologies. Since the convergence speed of these kind of networks is much slower (see [15]), a bigger stability period (500 rounds) and slower churn rate (1 node per round) were considered. Results are presented in Figures 9 and 10. In these settings, the behavior of *Flow Updating* is similar to the one previously described for random networks, although a deeper contrast between the effect of node arrival and departure is observed. Namely,

[2]The graphic of the distribution of node estimates in a scenario without message loss is very similar to the case of 20% faults.

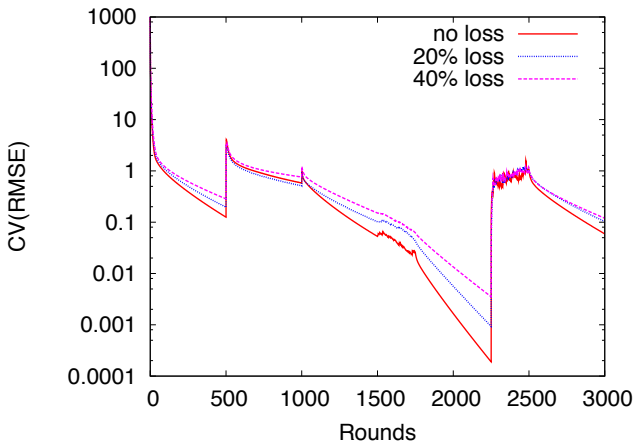

Figure 10. Coefficient of variation of the RMSE in a dynamic 2D/mesh network with message loss.

the perturbation introduced by a sudden (round 1000) or continuous (round 1500 to 1750) arrival of nodes is very small. On the contrary, node departure/crash have a greater impact in this kind of networks.

Node departure/crash breaks the flows established between nodes, and can result in the removal of links connecting different clusters, breaking the equilibrium in the whole network. This may lead to a global rearrangement of flows across the network, in order to reach a new equilibrium state. On the other hand, new nodes will simply provide new links (alternative paths), without breaking existing ones, leading to a smaller adjustment of the existing flows in order to converge to the new aggregate.

Overall experimental results show that *Flow Updating* can provide accurate aggregation results in demanding dynamic and faulty networks. It allows all nodes to continuously adjust their estimates according to network changes, due to node arrival/departure and crashes, quickly converging to the current network aggregate average, even with very high levels of message loss.

IV. Future Work

Our level of abstraction in the simulation follows the practice in the analysis of aggregation algorithms [11], [20], [12], [21], considering faults and churn. We concentrated on network dynamism and message loss, but assumed perfect failure detection. Further work will be assessing how practical failure detectors impact convergence speed.

Another aspect will be evaluating variations of Flow Updating for asynchronous systems. The algorithm could start a new round after receiving messages from a subset of the neighbors. In the extreme we could have an event driven approach were the algorithm reacts to single messages received. In these scenarios it will be interesting to evaluate a possible tradeoff between number of messages and convergence speed.

V. Conclusions

Average-based aproaches constitute an important segment in aggregation algorithms due to their independence from network topology and convergence to any desired precision. Our previous work on averaging by Flow Updating, in static settings, already introduced fault tolerance, achieving up to an order of magnitude improvement in convergence speed without increasing the message load [15].

Here we bring attention to the vulnerabilities of state of the art averaging techniques when faced with failures and dynamic environments. It is our belief that these shortcomings are not easy to fix and that actual mass exchange must give way to idempotent flow management, in order to address these demanding scenarios.

We introduce a dynamic version of Flow Updating where entries for neighbor nodes are added or removed according to the current participants. This simple design adapts to abrupt changes of network membership and tracks continuous variations of network size with a good level of accuracy. Evaluation showed that Flow Updating clearly outperforms previous strategies, and unlike them it adapts in a continuous fashion without requiring protocol restarts.

Finally we show that, even in dynamic settings, Flow Updating is hardly affected by a message loss rate of 40%, and one can expect it to tolerate even higher losses.

References

[1] R. V. Renesse, "The importance of aggregation," *Future Directions in Distributed Computing*, vol. 2584, pp. 87–92, 2003.

[2] S. Madden, M. Franklin, J. Hellerstein, and W. Hong, "TAG: a Tiny AGgregation service for ad-hoc sensor networks," *ACM SIGOPS Operating Systems Review*, vol. 36, no. SI, pp. 131–146, 2002.

[3] J. Li, K. Sollins, and D. Lim, "Implementing aggregation and broadcast over distributed hash tables," *ACM SIGCOMM Computer Communication Review*, vol. 35, no. 1, pp. 81–92, 2005.

[4] A. Ganesh, A. Kermarrec, E. L. Merrer, and L. Massoulié, "Peer counting and sampling in overlay networks based on random walks," *Distributed Computing*, vol. 20, no. 4, pp. 267–278, 2007.

[5] D. Kostoulas, D. Psaltoulis, I. Gupta, K. Birman, and A. Demers, "Decentralized schemes for size estimation in large and dynamic groups," in *Proc. 4th IEEE International Symposium on Network Computing and Applications*, 2005, pp. 41–48.

[6] M. Jelasity, A. Montresor, and O. Babaoglu, "Gossip-based aggregation in large dynamic networks," *ACM Transactions on Computer Systems (TOCS)*, vol. 23, no. 3, pp. 219–252, 2005.

[7] O. Kennedy, C. Koch, and A. Demers, "Dynamic approaches to in-network aggregation," in *Proc. 25th IEEE International Conference on Data Engineering (ICDE)*, 2009, pp. 1331–1334.

[8] S. Madden, R. Szewczyk, M. Franklin, and D. Culler, "Supporting aggregate queries over ad-hoc wireless sensor networks," in *Proc. 4th IEEE Workshop on Mobile Computing Systems and Applications*, 2002, pp. 49–58.

[9] Y. Birk, I. Keidar, L. Liss, A. Schuster, and R. Wolff, "Veracity radius: capturing the locality of distributed computations," in *Proc. 25th annual ACM symposium on Principles of Distributed Computing (PODC)*, 2006.

[10] L. Massoulié, E. Merrer, A.-M. Kermarrec, and A. Ganesh, "Peer counting and sampling in overlay networks: random walk methods," in *Proc. 25th annual ACM symposium on Principles of Distributed Computing (PODC)*, 2006.

[11] D. Kempe, A. Dobra, and J. Gehrke, "Gossip-based computation of aggregate information," in *Proc. 44th Annual IEEE Symposium on Foundations of Computer Science*, 2003, pp. 482–491.

[12] J.-Y. Chen, G. Pandurangan, and D. Xu, "Robust computation of aggregates in wireless sensor networks: Distributed randomized algorithms and analysis," *IEEE Trans. Parallel Distrib. Syst.*, vol. 17, no. 9, pp. 987–1000, 2006.

[13] F. Wuhib, M. Dam, R. Stadler, and A. Clemm, "Robust monitoring of network-wide aggregates through gossiping," in *Proc. 10th IFIP/IEEE International Symposium on Integrated Network Management*, 2007, pp. 226–235.

[14] P. Jesus, C. Baquero, and P. S. Almeida, "Dependability in aggregation by averaging," in *Proc. Simpósio de Informática (INForum)*, Lisboa, Portugal, Sep. 2009.

[15] ——, "Fault-tolerant aggregation by flow updating," in *Proc. 9th IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS)*, ser. Lecture Notes in Computer Science, vol. 5523. Lisbon, Portugal: Springer, Jun. 2009, pp. 73–86.

[16] N. A. Lynch, *Distributed Algorithms*. Morgan Kaufmann Publishers Inc., 1996.

[17] P. Erdős and A. Rényi, "On the evolution of random graphs," *Publications of the Mathematical Institute of the Hungarian Academy of Sciences*, vol. 5, pp. 17–61, 1960.

[18] M. F. Kaashoek and D. R. Karger, "Koorde: A simple degree-optimal distributed hash table," in *Proc. 2nd International Workshop on Peer-to-Peer Systems (IPTPS)*, ser. Lecture Notes in Computer Science, vol. 2735. Springer, 2003, pp. 98–107.

[19] P. Jesus, C. Baquero, and P. S. Almeida, "Using less links to improve fault-tolerant aggregation," 4th Latin-American Symposium on Dependable Computing (LADC), 2009, [Fast Abstract].

[20] M. Jelasity and A. Montresor, "Epidemic-style proactive aggregation in large overlay networks," in *Proc. 24th International Conference on Distributed Computing Systems*, 2004, pp. 102–109.

[21] E. L. Merrer, A.-M. Kermarrec, and L. Massoulié, "Peer to peer size estimation in large and dynamic networks: A comparative study," in *Proc. 15th IEEE International Symposium on High Performance Distributed Computing*.