Spanning Tree or Gossip for Aggregation: A Comparative Study

Lehel Nyers[1] and Márk Jelasity[2,⋆]

[1] University of Szeged, Hungary, and Subotica Tech, Subotica, Serbia
[2] MTA-SZTE Research Group on Artificial Intelligence, and University of Szeged, Hungary

Abstract. Distributed aggregation queries like average and sum can be implemented in several different paradigms including gossip and hierarchical approaches. In the literature, these two paradigms are routinely associated with stereotypes such as "trees are fragile and complicated" and "gossip is slow and expensive". However, a closer look reveals that these statements are not backed up by thorough studies. A fair and informative comparison is clearly needed. However, it is a very hard task, because the performance of protocols from the two paradigms depends on different subtleties of the environment and the implementation of the protocols. We tackle this problem by carefully designing the comparison study. We use state-of-the-art algorithms and propose the problem of monitoring the network size in the presence of churn as the ideal problem for comparing very different paradigms for global aggregation. Our experiments help us identify the most important factors that differentiate between gossip and spanning tree aggregation: the time needed to compute a truly global output, the properties of the underlying topology, and the sensitivity to dynamism. We demonstrate the effect of these factors in different practically interesting topologies and scenarios. Our results help us to choose the right protocol in the knowledge of the topology and dynamism patterns.

1 Introduction

Fully distributed aggregation is an important problem where we wish to execute queries such as sum, average, minimum, or maximum over unreliable networks (sensor networks, physical networks of routers, overlay networks, etc.), in which no central servers are directly accessible.

At least two paradigms are known for solving this problem. On the one hand, *gossip* algorithms were proposed to achieve large degrees of robustness. Gossip protocols do not rely on fixed topologies: nodes exchange information with random neighbors to implement a diffusion-like computation pattern, and as a result the system converges to a state where all the nodes know the query result. From the vast literature, here we focus on the adaptive approaches only. In [1], the authors propose the restarting technique to convert any one-shot algorithm into an adaptive one. Apart from restarting, other approaches have been proposed that focus on error correction through some form of bookkeeping at the nodes [2–5].

⋆ M. Jelasity was supported by the Bolyai Scholarship of the Hungarian Academy of Sciences. This work was supported by the European Union and the European Social Fund through project FuturICT.hu (grant no .: TAMOP-4.2.2.C-11/1/KONV-2012-0013).

F. Silva et al. (Eds.): Euro-Par 2014, LNCS 8632, pp. 379–390, 2014.

The other paradigm is *hierarchical aggregation*, a popular method in sensor networks [6]. It was also proposed for general process groups [7]. Tree-based aggregation remained unpopular in some areas, for example, peer-to-peer networks due to the widely held assumptions about its lack of robustness. There are a few notable exceptions: the Astrolabe framework [8], which is in fact only a virtual tree with completely unstructured gossip communication patterns behind it; the GAP protocol and its variants [5, 9–11] that actually build a spanning tree over a distributed network; and PRISM [12], a hierarchical approach that is built on top of a distributed hashtable, with a focus on detecting and signaling imprecise output.

Unfortunately, the literature is strongly influenced by stereotypes about both approaches like "spanning tree protocols are fragile" and "gossip protocols are slow and expensive". It is tacitly assumed that these statements have been conclusively settled. However, when surveying the literature, this does not turn out to be the case. In fact, we are not aware of any studies with a focus on a principled comparison among very different paradigms for aggregation. For example, Merrer et al [13] compare algorithms for size estimation, but they do not include spanning tree methods that are in the focus of our interest. Chitnis et al consider a very basic tree protocol that has no capabilities for reconfiguration, and briefly compare it with gossip [14]. The environment they consider is sensor networks. Due to the limited scope and suboptimal representatives of gossip and tree protocols, this work does not settle the problems we raised. Wuhib et al [5] propose an adaptive gossip protocol and compare it to GAP. Interestingly, GAP outperformed the gossip protocol in all scenarios examined (which were inspired by aggregation tasks in wired networks of routers). While this is a very nice result, it is far from complete due to the particular selection of the communication topology and the aggregation problem.

Our main contribution is that we propose a careful experimental design to shed light on the strengths and the weaknesses of both approaches. We experiment with competitive, state-of-the-art representatives of spanning tree and gossip protocols, and model different network environments. We identify the key aspects that determine the performance of the protocols in order to help application developers select the best solution in a given practical setting.

2 System Model

We assume that the system consists of N nodes that form a network with the help of reliable channels such as a TCP connections or physical links. Nodes communicate by exchanging messages over these channels only. Messages can be delayed. In addition, nodes can join and leave at any time. We assume the existence of a failure detector that sends a message to the node when a neighbor node fails. Leaving nodes and crashed nodes are treated identically. Leaving nodes can join again, and while offline, they retain their state. When they join again, they reconnect to their previous neighbors.

3 The Protocols in Our Comparison

The common problem these algorithms solve is the monitoring of aggregate values. That is, at any point in time t we have a network of $N(t)$ nodes all of which have a value. Let the set of values at time t be $A(t) = \{a_1(t), \ldots, a_{N(t)}(t)\}$. The task is

to continuously calculate (monitor) a global function $f(A(t))$. A given algorithm for solving this problem typically supports a well-defined set of aggregate functions f.

Due to lack of space we cannot present a complete description of all the protocols we examine in our experimental study. Instead, we describe the key ideas behind them, along with comments about our own implementation, where applicable. Our full implementation can also be downloaded.[1]

3.1 GAP (General Aggregation Protocol)

GAP is an adaptation of the classical self-stabilizing BFS construction algorithm of Dolev et al [15] that is based on message passing instead of shared tables. We implemented the version of GAP described in [9].

In GAP, there is a special node that acts as the root of the spanning tree. The root is fixed and guaranteed to remain available. The tree grows from the root as all the nodes discover their shortest path towards the root, starting with the neighbors of the root, and so on. GAP implicitly assumes a relatively stable underlying network. Each node in the network maintains a table that contains an entry for each neighbor and the node itself. Each table entry contains the level in the tree, and classifies the neighbor as parent, child, or peer. The parent of each node is always the neighbor with the minimal level (say, ℓ), and the node's own level is always $\ell + 1$. A table entry also contains the aggregate value in the subtree rooted in the neighbor. These values are used to calculate the node's own aggregate.

A node gets several types of messages related to changes in the topology (failed or new neighbors) or changes in the aggregate value (locally or in a subtree of a child node). When receiving a message, the node updates its own tables if necessary in such a way that the invariants of the tree structure and aggregate calculation are restored. Our implementation uses the "cache-like" policy [9] for maintaining the table, which means that table entries change only due to explicit messages and never due to predictions.

GAP can be implemented in a reactive or a proactive manner. In the former case, all changes are immediately reported to the neighbors. In the latter case, changes accumulate during a time period and are reported at once in a round-based fashion. We implemented the proactive round-based version, as it has better load balancing and generates fewer messages on average in dynamic environments.

The original publication of GAP did not mention that it is also important that the connections with neighbors need to preserve the order of the messages, otherwise inconsistent states can occur. This can be achieved with an appropriate transport layer, or at the application level as well.

3.2 Adaptive Gossip Protocols

We used the push-sum algorithm as a starting point [16]. In this algorithm (as in all gossip variants) the basic idea is that the nodes engage in a diffusive computation, during which nodes periodically send to each neighbor a proportion of the "mass" they store and also receive mass from neighbors. This way the nodes can collectively compute the average of all the values. Other aggregates, such as the network size can also be computed: if a single node has a value of 1, and all the other nodes have a value of 0 then the average is $1/N$, which can be used to recover the network size N.

[1] `http://peersim.sourceforge.net/`

The push-sum algorithm is by default a one-shot algorithm, unsuitable for monitoring. There are two approaches to achieve adaptivity. The first is the restart-based approach and the second is what we call the "bookkeeping" approach. We included in our set of algorithms a representative of both classes. In both cases, in each round a node with k neighbors sends one kth of its mass to each of the neighbors.

Restarted push-sum. The key idea is that the algorithm is run in *epochs* of some fixed length, after which the gossip protocol is restarted automatically in a distributed way [1]. In effect, the restart mechanism takes a snapshot of the system at the beginning of the epoch that involves the nodes that were live at that time, and then the aggregate of this snapshot is computed during the epoch. After the completion of the epoch, the computed aggregate value is used as the output of the algorithm, hence the output is delayed by roughly two epoch lengths at most. Depending on the topology of the network, epochs can be rather short (as few as 20 rounds) due to the quick convergence of gossip.

LiMoSense, a bookkeeping approach. Instead of restarting, a gossip protocol might attempt to repair the state of the nodes as a reaction to failure. This can be achieved if some variant of bookkeeping for the underlying gossip algorithm (e.g. push-sum) is implemented that makes it possible to "undo" those computations that had to do with a failed node, or that makes it possible to repair message drop failure by comparing books with neighbors. The main design goal of such protocols is the classical requirement of self-stabilization, that is, to be able to eventually converge after failures and dynamism stop. A state-of-the-art representative of such protocols is LiMoSense [2]. We use this protocol in our comparison study.

3.3 Common Properties

When comparing different paradigms, we should focus on application areas and systems where the paradigms in question all are feasible and have a similar cost. In other words, there are systems that are *obviously* suitable only for one or the other algorithm. Here, we do not focus on these obvious cases.

First, the system is assumed to have a *special stable node* that is guaranteed to remain available in the network. GAP crucially relies on such a node to act as the root of the tree for tree building and maintenance. Such a node is not critical for gossip but—given that due to GAP we need to assume a stable root—gossip protocols can and will take advantage of it too. For example, when calculating the network size, the node that has the initial value of 1 can be the root (see Section 4.1). Note that GAP does not rely on the root for reading out the value: it can be modified to propagate the global aggregate to all the nodes.

Second, all the protocols are *round based* with a period (round length) of Δ. They generate a very *similar amount of traffic* in each round: each node sends one message to each neighbor in each round. In the case of GAP this can be substantially reduced, but only when the network becomes static and there is no failure either. This is because no messages need to be sent if there is no change in the aggregated value or in the underlying topology. In our implementation, GAP broadcasts in each round even if there is seemingly no change. The reason is that—since we work with systems that constantly change—this results in a negligible amount of extra traffic, and it solves a subtle issue of the original algorithm related to churn.

4 Experimental Setup and Methodology

We used the PeerSim [17] simulator with the event-based engine in all our experiments.

4.1 Network Size as the Aggregation Problem of Choice

Calculating the average of distributed values is often the baseline problem used to evaluate generic distributed aggregation algorithms. This, however, is rather problematic because the performance then depends crucially on the distribution of the values. If the distribution is concentrated around the average, then one cannot differentiate between the ability of an algorithm to provide real global results and between the local sampling effect, that is, when the average of local samples is similar to the global average by pure chance. This is true in the case of both gossip and spanning tree algorithms.

It is vital that here we wish to compare the *global* behavior of the algorithms, that is, how they behave in scenarios where they need to consider the entire data set. The performance of such global tasks can be considered a *worst case*, which can only improve when local neighborhoods already offer a good approximation of a given query. Of course it is of interest how certain algorithms react to specific distributions, and one could even develop algorithms that explicitly exploit specific known distributions, if such prior knowledge is available. However, without prior knowledge getting a quick result due to local sampling is just a matter of chance, so when comparing very different generic paradigms, we consider a robust worst case analysis more informative and preferable.

Our choice is the network size estimation problem. For this problem, the spanning tree approach counts each node according to the tree hierarchy: all nodes have a value of 1, and the tree calculates the sum. The gossip protocols here will calculate the average in a network of N nodes were the initial value is 0 at all nodes except the root, where it is 1, which gives $1/N$ as a result [1]. In both cases, the point is that the problem is clearly global, where a useful answer is available only after the algorithm has globally converged.

4.2 Network Topologies

Our protocols need undirected topologies, so where the original topology definition is directed, it has to be understood with the directionality of the edges dropped. All networks are of size $N = 1000$ unless otherwise stated.

NewsCast. A dynamic topology defined in [18]. In a nutshell, without describing NEWSCAST in detail, each node will have a new set of random neighbors in each cycle using the same cycle period as the aggregation protocol. The number of neighbors is $k = 30$. The motivation is that gossip protocols are often implemented over such dynamic topologies so that nodes can communicate with random samples from the network in each cycle, as assumed in theoretical discussions of gossip protocols.

Random k-out. A static topology in which every node connects to a set of k random neighbors. After dropping directionality, the average degree is $2k$. The motivation is that randomly sampled, but static, topologies have been proposed recently as the optimal choice in commercial P2P platforms over the Internet [19].

Binary Tree. An undirected balanced binary tree is formed. In our experiments the root node of the aggregation protocols is placed at different levels of the tree from 0 (the root of the binary tree) to $\lceil \log_2 N \rceil$, the leaf level of the tree. We include this artificial topology to be able to illustrate a major difference between gossip and spanning tree approaches.

Barabasi-Albert (BA). To test heavy tailed degree distributions, we include the BA network that is constructed incrementally. New nodes connect to old nodes already in the network according to the preferential attachment rule, that is, with a probability proportional to the degree of the old node [20]. New nodes get $k = 2$ edges when they are added to the topology. In our experiments the root node of the aggregation protocols is placed at nodes with different degrees in this topology.

4.3 Failure and Churn Scenarios

We used the same model of message delay in all experiments: each message is delayed by a uniform random time drawn from the interval $[0, 0.2\Delta]$. Our preliminary experiments revealed very little sensitivity to message delay in all the protocols, so we do not focus on this aspect. We consider no message drop failures. This is because in most scenarios that are reasonable for a spanning tree the underlying topologies in question are static, so it is feasible to apply a reliable transport layer such as TCP.

The protocols require a failure detector. We assume a timeout-based detector with a timeout of 5Δ in all experiments. Our preliminary experiments suggested very little sensitivity to this parameter as well, so we keep it fixed throughout the study.

Node churn was modeled based on statistics from a BitTorrent trace [21] as well as known empirical findings [22]. We draw the online session length for each node independently from a log-normal probability distribution with two different parameter settings. The first setting that we call *fast churn* is $\mu = 3$ and $\sigma^2 = 1$, which results in a mean of ~ 33. The unit of the resulting online session lengths is the communication period Δ. This—considering that Δ can be expected to be in the range of seconds—is a rather short session length so it represents a very dynamic scenario. The second set of parameters that we call *slow churn* is $\mu = 6$ and $\sigma^2 = 2$, which results in a mean of ~ 1096.

Offline session lengths are determined implicitly by fixing the number of nodes that are online at the same time. The ratio of online nodes was set to a range of values from $\alpha = 1$ to $\alpha = 0.2$. As stated previously, nodes that re-join the network retain the state they had when leaving the network.

4.4 Evaluation Methodology and Metrics

We are interested in static behavior. As mentioned above, we assume a constant churn pattern with a static expected network size αN, where α is the ratio of online nodes in the scenario in question. In this setting, we expect a good monitoring algorithm to consistently signal $\hat{N}(t) = \alpha N$ as the approximated network size in cycle t.

To measure how close a given algorithm is to this optimal behavior, we run each scenario 10 times for 10,000 cycles, and collect statistics of the absolute error $|\alpha N - \hat{N}(t)|/(\alpha N)$ over the last 9,000 cycles in each run. We ignore the first 1,000 cycles in order to allow the system to reach an equilibrium state. We plot the average and the standard deviation (with error bars).

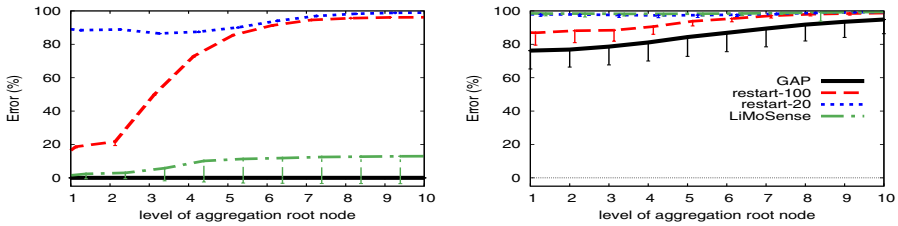

Fig. 1. Binary tree, no churn (left) and fast churn, 80% of the nodes online (right). The horizontal axis shows the level of the aggregation root node in the physical topology (0: root, 10: leaves).

5 Results

First, we demonstrate some weaknesses of the gossip approaches and GAP that are not so evident at first sight. This will shed light on which scenarios to avoid for these paradigms. Subsequently, we look at the two realistic topologies: static random k-out and the BA topology, and take a closer look at some interesting subtleties that are important in these cases, and that define which approaches are preferable.

5.1 The Achilles Heel of Gossip

Let us start with the Achilles heel of gossip protocols. In principle, gossip protocols for aggregation have been shown to work on any connected topology, that is, they are guaranteed to converge. However, if the communication topology is not a fully connected graph, but instead a *static* graph with relatively small average degree, then the convergence speed is well-known to depend on the mixing time of a certain random walk on this topology [23]. On the other hand, the convergence time of GAP depends on the diameter (that is, the maximal minimal path length) that bounds the maximal number of steps information needs to take to reach the root.

It is often the case that graphs with a low diameter also have a rapid mixing time, but trees are exceptions. For example, the rooted balanced binary tree has a diameter of $O(\log N)$ whereas it has a mixing time of $O(N)$ (see, for example, [24], Example 7.7).

For this reason, at least in the failure-free scenario, we expect gossip protocols to suffer, and this is indeed the case as Figure 1 (left) shows. GAP achieves full precision very quickly, whereas even LiMoSense does not reach convergence within 10,000 rounds when the node that is assigned the role of the root node is closer to the leaves in the physical topology, let alone restart that is inherently limited in the number of rounds until convergence. (We remind the reader not to confuse the root node of the aggregation with the root of the physical topology.) However, when we introduce churn, all algorithms suffer since the underlying topology is very fragile. Still, GAP performs best (see Figure 1 (right)).

The results also reveal another important point, namely it does matter a lot where the root node is placed within the underlying physical topology. It is much harder to break out of a region closer to the leaves for the diffusion process as it is from the root (recall that for gossip the aggregation root is initialized with a value of 1, while the remaining nodes have a value of 0).

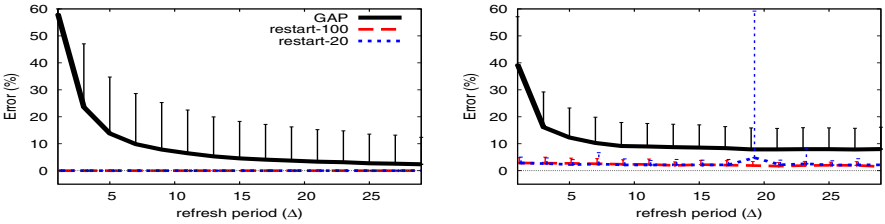

Fig. 2. NewsCast with $k = 30$, no churn (left) and fast churn, 80% of the nodes online (right). The horizontal axis shows the neighborhood refresh period of NewsCast in rounds (Δ).

5.2 The Achilles Heel of Spanning Trees and Bookkeeping Gossip Protocols

In many cases, gossip protocols assume a random set of neighbors in each round [1] that is given by a dynamic protocol for peer sampling [18]. This radically dynamic neighbor set is ideal for vanilla gossip, however, if bookkeeping is involved, it becomes a serious problem, since the tables will grow indefinitely until they reach the size of the whole network. This is not scalable, since all entries have an associated failure detector as well, which need to maintain a communication link with each node. For this reason, the members of the old neighbor set should be treated as failed nodes, to get scalability. This, however, completely destroys the ability of the protocol to converge if the aggregation task is global, like network size estimation. All in all, with dynamic peer sampling bookkeeping gossip cannot be applied at all with any hope of success.

For GAP, the changing neighbor set raises similar issues: growing tables (and eventually a spanning tree with a star topology) or the option of treating old neighbors as failed. In our implementation, we opt for the second approach, as the option of growing tables is clearly not scalable.

Figure 2 shows simulation results with the NewsCast dynamic topology. Clearly, for fast refreshing periods the only feasible protocol is restarted gossip. Yet in the case of slower refreshing (when the topology becomes relatively stable on the short run) GAP is competitive. LiMoSense is the least favorable option in this scenario.

5.3 The k-Out Topology

We examined the k-out topology for different values of k. Without churn, all the protocols can achieve an error that is practically 0% for $k \geq 2$, except restart-20 that achieves 25% error for $k = 2$. Clearly, an epoch length of 20 is not sufficient for such a low value of k. Note that the lower the value of k is the greater the mixing time becomes.

Figure 3 contains our experiments involving churn. Our first observation is that GAP and restart are rather insensitive to the speed of churn, whereas LiMoSense is very sensitive. In the slow churn scenario its results are dramatically better. This is because fast churn is highly disruptive for this algorithm due to the constant attempts to repair the state of the system when a neighbor leaves.

At the same time, in slow churn all the algorithms become rather unstable when the offline session lengths are long (that is, when α is small). This is because—although the network is relatively stable—in such scenarios the aggregation root can become disconnected and can remain so for a relatively long time, which temporarily causes extremely large errors.

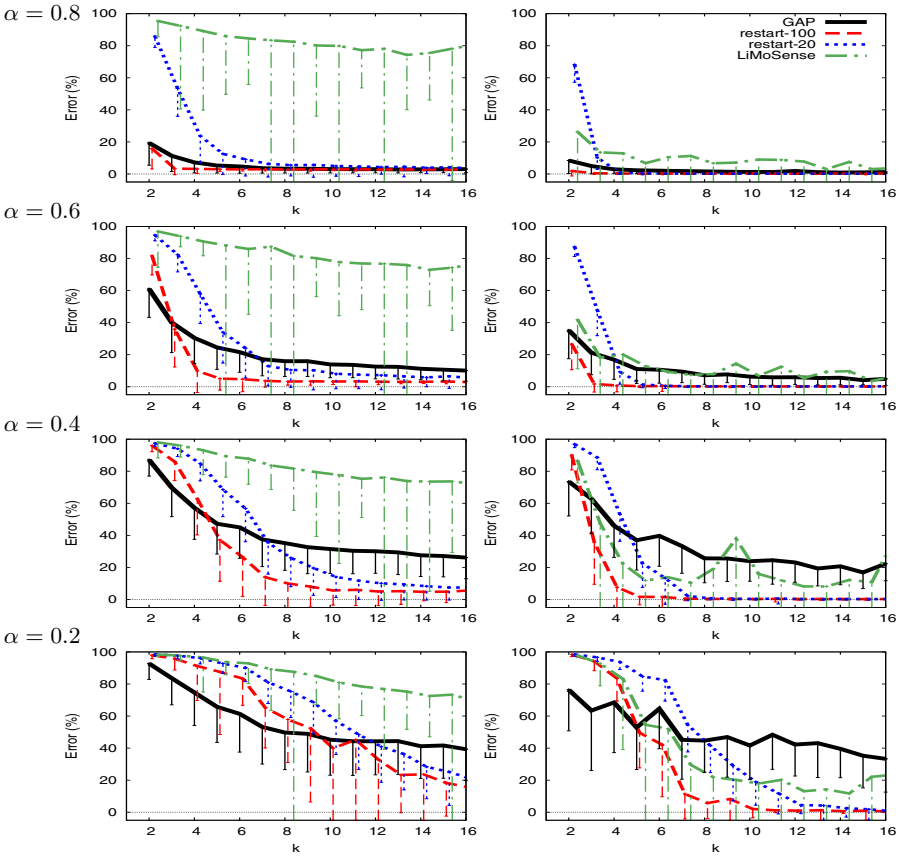

Fig. 3. Results with k-out topology, fast churn (left column) and slow churn (right column). The horizontal axis shows values of k. The proportion of online nodes is given by α.

As for GAP, we can observe an interesting case that is consistent with our findings over the binary tree topology: when k is very small, GAP has a slight advantage due to not depending on the mixing time of the topology. Note that for a very small k the random k-out topology behaves locally like a tree as there is a rather small probability for finding short circles, which slows gossip protocols down.

For large values of k gossip protocols can take advantage of the very good mixing properties and can beat GAP, esp. with an epoch length of 100. GAP also profits from an increasing k (and therefore a decreasing diameter, and more options to repair the tree) but not as much as gossip protocols.

5.4 The Barabasi-Albert Topology

We generated one BA topology with $k = 2$ as previously described. In this fixed topology we placed the aggregation root at nodes with different degrees. Our results involving churn are shown in Figure 4.

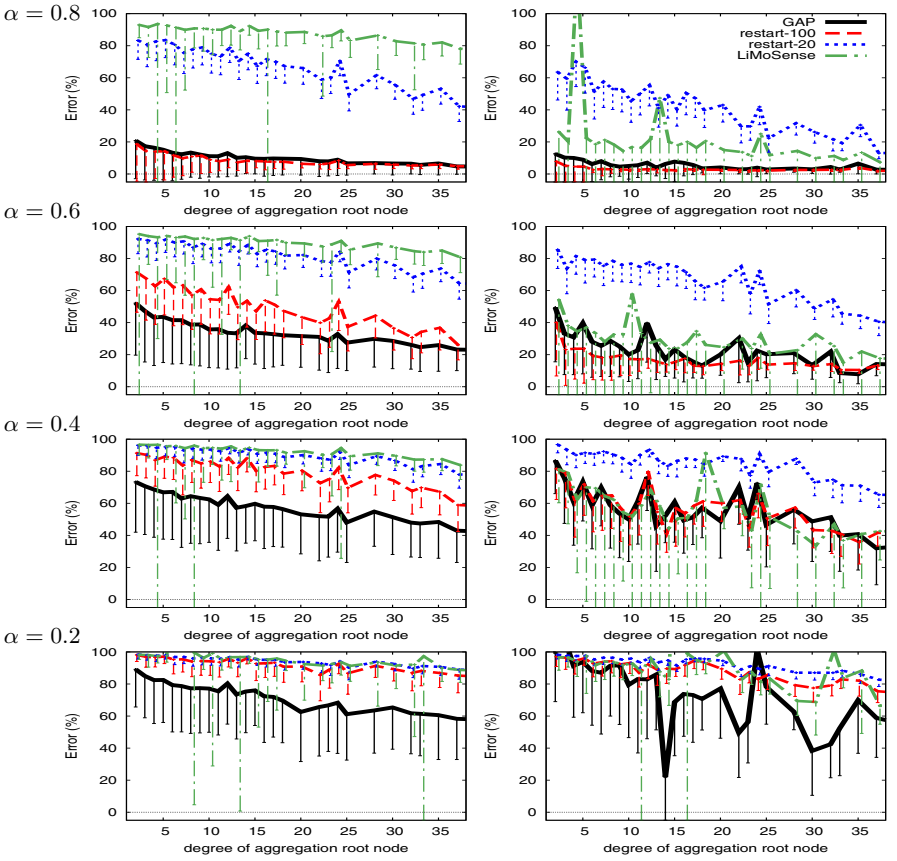

Fig. 4. Results with the BA topology, fast churn (left column) and slow churn (right column). The horizontal axis shows the degree of the node where the aggregation root is placed. The proportion of online nodes is given by α.

We can observe a strong dependence of the precision of the aggregation result on the position of the aggregation root in the underlying BA topology. Central nodes with a large degree achieve a significantly better approximation. This is also true for the gossip protocols, which here also rely on a fixed "root" node (see Section 3.3).

As in the k-out topology, without churn (not shown), all the protocols can achieve an error that is practically 0%, except restart-20 that achieves an error of 35% to 5% depending on the centrality of the root. Restart-20 performs poorly throughout the experiments, clearly indicating that an epoch length of 20 is not sufficient. At the same time, restart-100 is among the best options in most scenarios.

Clearly, in these scenarios GAP provides the most stable performance. As with the k-out topology, gossip protocols continue to be sensitive to the speed of churn: with LiMoSense it is more so, but the restart variants also show sensitivity, with restart-100 being the most robust.

As in the k-out topology, in slow churn we observe a very large variance for small α and for a low degree aggregation root. The reason is the same: the aggregation root can get disconnected.

Table 1. Summary of conclusions

	sensitivity to changing		delay due to	
	membership	**topology**	**convergence**	**epoch length**
spanning tree	moderate	high	diameter	none
bookkeeping gossip	high	high	mixing time	none
restarted gossip	moderate	none	mixing time	epoch length

6 Discussion and Conclusions

In this paper, we compared three different paradigms for global distributed aggregation: approaches based on a spanning tree, restarted gossip, and bookkeeping gossip. We argued that network size estimation is an appropriate problem for the purposes of this comparison. We stressed the role of different topologies, and shed light on the weak and strong points of the approaches.

Table 1 summarizes some of the conclusions we arrived at in the evaluation section. In our experiments the effective network size was constant, so the effect of the delay due to the epoch length remained hidden. However, the epoch length must be chosen such that it lies in the range of the mixing time so as to allow for proper convergence. This means that restarted gossip will double the delay of bookkeeping gossip in the worst case, while it is not sensitive to a dynamic topology (due to not relying on failure detectors and neighborhood tables) and it is less sensitive to churn for the same reason.

As for the spanning tree, the convergence time of gossip (that depends on the mixing time) is typically at least an order of magnitude larger than the diameter in most topologies, even in the random k-out topology (which has a low mixing time), let alone more practical topologies. Our experiments clearly support this insight. This means that a spanning tree is much faster than the other methods, and its advantages mainly result from this property, along with the ability to self-repair equally quickly, when the topology is not too dynamic.

Overall, when selecting the right protocol, one needs to consider the structure of the topology and the patterns of dynamism in the membership (churn) and the topology itself. If the topology is relatively stable, a spanning tree approach is preferable even in high churn, while for dynamic topologies a restarted gossip protocol with the right epoch length is more suitable. We could identify no scenarios where bookkeeping gossip is clearly preferable, when truly global aggregation is needed. If local sampling approximates the global aggregate well, we face a very different problem that requires a different approach for analysis. Nevertheless, results on global problems always represent a lower bound on performance. The ultimate solution is most likely a combination of gossip and tree approaches in an adaptive way, based on the automated detection of topology and dynamism properties; an interesting venue for future research.

References

1. Jelasity, M., Montresor, A., Babaoglu, O.: Gossip-based aggregation in large dynamic networks. ACM Transactions on Computer Systems 23(3), 219–252 (2005)
2. Eyal, I., Keidar, I., Rom, R.: Limosense – live monitoring in dynamic sensor networks. In: Erlebach, T., Nikoletseas, S., Orponen, P. (eds.) ALGOSENSORS 2011. LNCS, vol. 7111, pp. 72–85. Springer, Heidelberg (2012)

3. Jesus, P., Baquero, C., Almeida, P.S.: Fault-tolerant aggregation by flow updating. In: Senivongse, T., Oliveira, R. (eds.) DAIS 2009. LNCS, vol. 5523, pp. 73–86. Springer, Heidelberg (2009)
4. Mehyar, M., Spanos, D., Pongsajapan, J., Low, S.H., Murray, R.M.: Asynchronous distributed averaging on communication networks. IEEE/ACM Trans. Netw. 15(3), 512–520 (2007)
5. Wuhib, F., Dam, M., Stadler, R., Clemm, A.: Robust monitoring of network-wide aggregates through gossiping. In: Proc. 10th IFIP/IEEE Intl. Symp. on Integrated Management (IM 2007), pp. 21–25 (May 2007)
6. Madden, S., Franklin, M.J., Hellerstein, J.M., Hong, W.: TAG: A tiny aggregation service for ad-hoc sensor networks. In: Proc. 5th Symp. on Operating Systems Design and Implementation (OSDI 2002), pp. 131–146 (2002)
7. Gupta, I., van Renesse, R., Birman, K.P.: Scalable fault-tolerant aggregation in large process groups. In: Proc. Intl. Conf. on Dependable Systems and Networks (DSN 2001). IEEE Computer Society Press (2001)
8. Birman, K.P., van Renesse, R., Vogels, W.: Scalable data fusion using astrolabe. In: Proc. Fifth Intl. Conf. on Information Fusion (FUSION 2002), vol. 2, pp. 1434–1441 (2002)
9. Dam, M., Stadler, R.: A generic protocol for network state aggregation. In: Proc. Radiovetenskap och Kommunikation, RVK 2005 (2005)
10. Prieto, A.G., Stadler, R.: A-gap: An adaptive protocol for continuous network monitoring with accuracy objectives. IEEE Trans. on Netw. and Serv. Manag. 4(1), 2–12 (2007)
11. Krishnamurthy, S., Ardelius, J., Aurell, E., Dam, M., Stadler, R., Wuhib, F.Z.: Brief announcement: the accuracy of tree-based counting in dynamic networks. In: ACM Symp. on Principles of Distr. Comp. (PODC), pp. 291–292. ACM (2010)
12. Jain, N., Mahajan, P., Kit, D., Yalagandula, P., Dahlin, M., Zhang, Y.: Network imprecision: A new consistency metric for scalable monitoring. In: Proc. 8th USENIX Conf. on Operating Systems Design and Implementation (OSDI 2008), pp. 87–102. USENIX Association (2008)
13. Le Merrer, E., Kermarrec, A.M., Massoulie, L.: Peer to peer size estimation in large and dynamic networks: A comparative study. In: Proc. 15th IEEE Intl. Symp. on High Performance Distr. Comp. (HPDC 2006), pp. 7–17 (2006)
14. Chitnis, L., Dobra, A., Ranka, S.: Aggregation methods for large-scale sensor networks. ACM Trans. Sen. Netw. 4(2), 9:1–9:36 (2008)
15. Dolev, S., Israeli, A., Moran, S.: Self-stabilization of dynamic systems assuming only read/write atomicity. Distributed Computing 7(1), 3–16 (1993)
16. Kempe, D., Dobra, A., Gehrke, J.: Gossip-based computation of aggregate information. In: Proc. 44th Annual IEEE Symp. on Foundations of Computer Science (FOCS 2003), pp. 482–491. IEEE Computer Society (2003)
17. Montresor, A., Jelasity, M.: Peersim: A scalable P2P simulator. In: Proc. 9th IEEE Intl. Conf. on P2P Comp. (P2P 2009), pp. 99–100. IEEE (September 2009), extended abstract
18. Jelasity, M., Voulgaris, S., Guerraoui, R., Kermarrec, A.M., van Steen, M.: Gossip-based peer sampling. ACM Transactions on Computer Systems 25(3), 8 (2007)
19. Roverso, R., Dowling, J., Jelasity, M.: Through the wormhole: Low cost, fresh peer sampling for the internet. In: Proc. 13th IEEE Intl. Conf. on P2P Comp. (P2P 2013). IEEE (2013)
20. Albert, R., Barabási, A.L.: Statistical mechanics of complex networks. Reviews of Modern Physics 74(1), 47–97 (2002)
21. Roozenburg, J.: Secure decentralized swarm discovery in Tribler. Master's thesis, Parallel and Distributed Systems Group, Delft University of Technology (2006)
22. Stutzbach, D., Rejaie, R.: Understanding churn in peer-to-peer networks. In: Proc. 6th ACM SIGCOMM Conf. on Internet Measurement (IMC 2006), pp. 189–202. ACM (2006)
23. Boyd, S., Ghosh, A., Prabhakar, B., Shah, D.: Randomized gossip algorithms. IEEE Transactions on Information Theory 52(6), 2508–2530 (2006)
24. Levin, D.A., Peres, Y., Wilmer, E.L.: Markov Chains and Mixing Times. AMS (2008)