Rapid and Round-free Multi-pair Asynchronous Push-Pull Aggregation

Hyun-Gul Roh, Claudia Lavinia Ignat

Rapid and Round-free Multi-pair Asynchronous Push-Pull Aggregation

Hyun-Gul Roh, Claudia-Lavinia Ignat

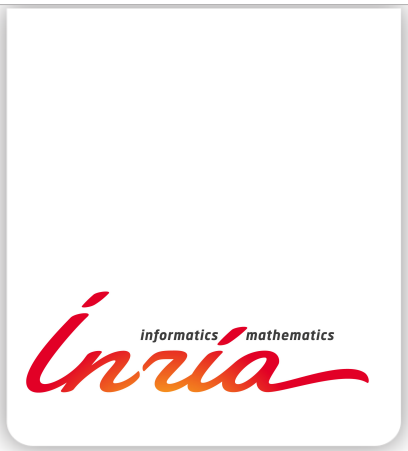
RESEARCH REPORT

N° 8044

August 2012

Project-Teams SCORE

ISSN 0249-6399 ISRN INRIA/RR--8044--FR+ENG



Rapid and Round-free Multi-pair Asynchronous Push-Pull Aggregation

Hyun-Gul Roh*, Claudia-Lavinia Ignat†

Project-Teams SCORE

Research Report n° 8044 — August 2012 — 30 pages

Abstract: As various distributed algorithms and services demand overall information on large scale networks, the protocols that aggregate data over networks are essential, and the quality of aggregations determines the quality of those distributed algorithms and services. Though a variety of aggregation protocols have been proposed, gossip-based iterative aggregations have outstanding advantages especially in accuracy, result distribution, topology-independence, and resilience to network churns. However, most of iterative aggregations, especially push-pull style aggregations, suffer from two synchronization constraints: synchronized rounds and synchronized communication. Namely, iterative protocols generally need prior configurations to synchronize rounds over all nodes, and messages should be exchanged in a synchronous way in order to ensure accurate estimates in push-pull or push-sum protocols. This paper proposes multi-pair asynchronous push-pull aggregation (MAPPA), which liberates the push-pull aggregations from the synchronization constraints, and pursues a way to accelerate the aggregation speed. MAPPA considerably reduces aggregation times, and shows an improvement in fault-tolerance. Thanks to topology independence, inherent from gossip mechanisms, and its rapidness, MAPPA is resilient to network churns, and thus suitable for dynamic networks.

Key-words: aggregation, gossip protocol, asynchronous computation, peer-to-peer overlay network

* hangulroh@gmail.com
† claudia.Ignat@loria.fr

RESEARCH CENTRE
NANCY – GRAND EST

615 rue du Jardin Botanique
CS20101
54603 Villers-lès-Nancy Cedex

Rapid and Round-free Multi-pair Asynchronous Push-Pull Aggregation

Résumé : Ce document montre comment utiliser le style RR.sty. Pour en savoir plus, consulter le fichier RR.dvi ou RR.pdf. Définissez toujours les commandes avant utilisation.

Mots-clés : calcul formel, base de formules, protocole, différentiation automatique, génération de code, modélisation, lien symbolique/numérique, matrice structurée, résolution de systèmes polynomiaux

Contents

1 Introduction

In large scale networks, such as peer-to-peer overlay networks, wireless sensor networks, and content delivery networks, various distributed services and algorithms demand to aggregate global statistic information. For example, cloud storage services should collect the information of disk capacity spread over their huge clouds; the number of active nodes need to be monitored by network management services; and load balancing algorithms will use current states of loads as inputs. Reflecting such strong demand, for the past decade, various aggregation protocols have been explosively proposed.

Aggregation protocols can be classified under two large groups: topology dependent and independent ones [7]. Topology dependent protocols, e.g., TAG [13], need prior efforts to constitute specific routing topologies such as trees, graphs, or rings. If such a hardship is endured, this class of aggregations are efficient in message overhead, but inflexible to network churns or fragile to single points of failures such as node crashes.

On the other hand, topology independent protocols are flexible in deployment and resilient to network changes. There are various kinds of topology independent aggregations exhibiting different characteristics. For example, Sample& Collide [14] and Hop-Sampling [10] rely on probabilistic sampling technique and probabilistic polling technique, respectively. Due to their probabilistic natures, these two aggregations are inaccurate even in fault-free environments [16].

Iterative aggregation protocols (also called averaging protocols by [7]), such as push-sum [9], push-pull [18][5], A1/A2 [15], and Flow-Updating [6], compute the aggregation by exchanging periodic messages. Especially, gossip-based iterative protocols are scalable and reliable, and constrain no specific network topologies due to the characteristics of gossiping, which randomly chooses messages recipients from neighbors. Generally, iterative protocols are of considerable merits in accuracy, result distribution and resilience of network churns [7][16].

This paper sheds new light on the push-pull style aggregations [18][5][15] from a practical point of view. We strive to find a way to liberate aggregations from *two synchronization constraints*: one is posed to most of the iterative aggregations, and the other is specific to the push-pull aggregations. Ultimately, we pursue a practical aggregation which is rapid and easily deployable.

The first constraint is that most of iterative protocols require all the nodes over large networks to synchronize their *rounds*, meaning that all nodes should coordinate the starting point of every round. A round is a unit of the iteration, in which estimates are exchanged and updated. Round-based protocols are conceptually simple, but synchronizing rounds in real-world networks cannot be achieved immediately and incurs non-negligible overhead. According to [1], it takes dozens of seconds to reduce the time difference between the first and the last starting points of each round within 100 ms. In other words, round-based protocols need to lay the ground for their bootstrapping, and the length of a round should be long enough to cope with such a time difference.

The second constraint is related to the synchronization of message passing. In push-pull aggregations, if messages are exchanged asynchronously, interferences among asynchronous updates cause damage to *system mass*, which is the sum of estimates located at all nodes. The *system mass conservation* is a key property for the correct aggregations in the iterative protocols [9], and many efforts [15][19][20][21] have been made in order to satisfy this property in the presence of the asynchrony. However, they still disallow certain kinds of asynchronous updates because interferences among such updates could result in damage to the system mass.

This paper proposes an adapted push-pull protocol named *multi-pair asynchronous push-pull aggregation* (MAPPA), which is free from these two synchronization constraints. We address a simple property, *pair mass conservation* that enables MAPPA to isolate asynchronous updates for avoiding interferences among them. By virtue of this permission for asynchrony, we also

liberate MAPPA from synchronized rounds. Consequently, MAPPA needs no prior setup for synchronized rounds, and is easy to be deployed in any large scale networks.

Furthermore, we take another step for rapid aggregations. By adopting multiple concurrent pushes rather than a single one, i.e., *multi-pair asynchrony*, we can accelerate convergences of estimates significantly, compared to other iterative protocols. Concomitantly but considerably, fault-tolerance is also enhanced in MAPPA. In our evaluations, we will show the behaviors of MAPPA under various conditions, and present comparisons with some other approaches. The results show that MAPPA takes a fraction of the convergence times needed in other approaches. Bringing out the best of the rapid convergence, we present an asynchronous overlapped restarting technique that enables all nodes to maintain convergent estimates, and prove that MAPPA is advantageous to dynamic networks.

This paper is organized as follows. Section 2 defines our system model and two synchronization constraints. Section 3 gives a survey of related work regarding iterative aggregations. We address the pair mass conservation in Section 4, and then Sections 5 and 6 propose MAPPA and the asynchronous overlapped restarting technique. In Section 7, MAPPA is evaluated with various experiments, and we conclude the paper in Section 8.

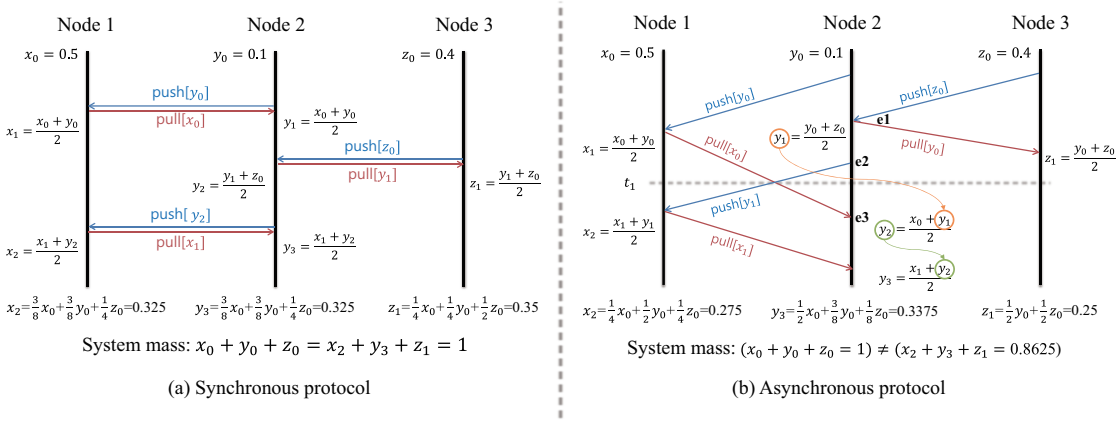


Figure 1: An example of synchronous and asynchronous aggregations. The estimates of nodes 1, 2, and 3 are x, y, and z. In (b) asynchronous protocol, system mass is not conserved.

2 Background

2.1 System model

This section presents a model of our gossip-based asynchronous aggregation protocol that will be used throughout the paper. Let $\mathcal{N} = \{1, 2, \cdots, n\}$ be a set of n nodes composing a network, and $\mathcal{G}_i \subset \mathcal{N}$ be a set of neighbors to which node i gossips messages. Each node $i \in \mathcal{N}$ has a value v_i that should be aggregated across the entire system, and maintains an estimate $\hat{v}_i(t)$. This paper defines an estimate $\hat{v}_i(t)$ as a function of time t because it is non-periodically updated whenever node i asynchronously receives pushes and pulls.

This paper focuses on the average aggregation because it is a basis of various aggregations such as sum, count, product, or variance [5]. For a set of the values of all the nodes $\mathbf{v}=\{v_1, v_2, \cdots, v_n\}$, the goal of the average aggregation is for each node i to obtain an estimate $\hat{v}_i$ that is equal to the average $\bar{\mathbf{v}} = \frac{1}{n}\sum_{\forall k \in \mathcal{N}} v_k$. Therefore, every estimate should converge to the average as follows:

$$\hat{v}_i \approx \frac{1}{n} \sum_{\forall k \in \mathcal{N}} \hat{v}_k \text{ for } \forall i \in \mathcal{N} \tag{1}$$

According to this formula, the aggregation is determined by the sum of estimates $\sum_{\forall k \in \mathcal{N}} \hat{v}_k$, called the system mass. Kempe et al. [9] addressed *the system mass conservation* as a property for the correct aggregation. In Section 4, we propose a more elaborative definition of the system mass conservation with a solution to preserving the property in the presence of asynchrony.

This paper focuses on the push-pull style aggregation in which updates carry out pairwise. Namely, each node i pushes its local estimate to a random neighbor in $\mathcal{G}_i$, and then a certain value is pulled from the neighbor. We denote a push and a pull message by push$[p_s]$ and pull$[p_l]$, where p_s and p_l represent the delivered values of the push and pull, respectively.

2.2 Two synchronization constraints

In this section, we identify two synchronization constraints of the push-pull aggregation.

The first constraint is related to the communication of pushes and pulls. Following the push-pull protocol of Jelasity et al. [18][5] that takes arithmetic means fairly on both sides, Fig. 1 shows an example of synchronous and asynchronous communications. Specifically, the push-pull aggregations can be distinguished into synchronous and asynchronous ones as follows:

- **Synchronous protocol**: Any push synchronizes with its corresponding pull message. In other words, as shown in Fig. 1-(a), after a node sends push$[y_0]$, the node should wait for pull$[x_0]$ that responds to push$[y_0]$ *without doing anything*.
- **Asynchronous protocol**: While a node is waiting for a pull responding to a sent push, it can deal with some events of pairwise updates. Between an update of push-pull, three types of events can happen as illustrated in Fig. 1-(b):

  e1) receiving a push from any node and responding to it with a pull,

  e2) sending a new push to any node,

  e3) receiving a pull of any previous sent push.

  Note that **e2** and **e3** are reciprocal each other. For example, between the update of push$[y_0]$ − **e3**(pull$[x_0]$), push$[y_1]$ corresponds to **e2** while pull$[x_0]$ is **e3** between the update of **e2**(push$[y_1]$) − pull$[x_1]$.

If the push-pull aggregation of Jelasity et al. [5] is operated with the asynchronous communication, the system mass is damaged due to the interferences among pairwise updates. For example, at node 2 of Fig. 1-(b), **e1** and **e3** make the values of y that are used in the updates of the both-side of pull$[x_0]$ and pull$[x_1]$ be different, thereby interfering with each other. As such interferences cause errors in calculating arithmetic means on both-sides of a pull, the asynchronous aggregation cannot conserve the system mass as in Fig. 1-(b). In Section 4, we introduce a simple and practical property to liberate this synchronization constraint.

Another synchronization constraint is related to the *round* (also known as cycle), which is a time unit most of iterative aggregations [9][5][19][21][20][6] are adopting. Rounds have to be synchronized so as to proceed in a lock-step mode; in other words, the same round r should have physically identical starting and ending times at all nodes. As stated in Introduction, however, the round synchronization in real-world networks cannot be achieved immediately, and cannot avoid errors [1]. The round-based push-pull aggregations are encouraged to complete a pairwise update in a round, and thus a push and a pull generated at round r should arrive to another node in the same round r. This constraint is hard to meet because of arbitrary propagation delays. The event types **e2** and **e3** defined above can be associated with the synchronized round:

e2) sending multiple pushes to multiple neighbors in a round,

e3) receiving an out-of-round push or pull.

In Section 3, we will examine how to deal with these events in existing iterative protocols.

In our proposed protocol, each node cyclically sends one or more pushes in parallel regardless of the completions of the previous pairwise updates. Besides, a push and a pull generated at a node are accepted by another node regardless of the receiver's state, meaning pairwise updates of each node proceeds independently without synchronization. To distinguish such characteristics with those of the round, this paper uses the term *cycle*.

3 Related Work

The aggregation protocols can be classified according to two perspectives: the network topologies and computation methods on which the algorithms reply [16][7]. Different classes of aggregation protocols impose different restrictions and trade-offs when other distributed applications adopt them. Since our proposed protocol is a gossip-based push-pull aggregation that is irrespective of specific network topologies and based on iterative computations, this section focuses on the similar category of the aggregation algorithms.

According to the comparative works of [16] and [7], push-sum [9], push-pull [18][5][15], and Flow-Updating [6] protocols belong to the category of iterative aggregations. They have advantages in accuracy, topology-independence, result distribution, and fault-tolerance, over other approaches, such as TAG [13], Sample&Collide [14] and Hop-Sampling [10]. However, as iterative aggregations need periodic message passing among nodes, the synchronization issues addressed in Section 2.2 are important. We, therefore, look into the existing iterative aggregations in terms of the synchronization issues.

Kempe et al. [9] proposed a push-sum aggregation protocol, where nodes send only pushes. However, this protocol requires an acknowledgment for every push in order to cope with faults such as message loss. Using the values delivered with many pushes, nodes update local estimates at the end of every round. Although pushes can transfer with arbitrary delays, synchronized rounds are encouraged so that any push should arrive to another node in the same round as at the sending node [20].

As stated in Section 2.2, Jelasity et al. [18][5] presented the synchronous push-pull aggregation protocol. Rao et al. [20] applied asynchrony to the protocol of Jelasity et al. and the push-sum protocol. They permit *loosely synchronized rounds* that allow different nodes not exactly same time to begin any round r. Nevertheless, to some extent, the rounds should be synchronized so that messages generated at round r should be delivered to other nodes at the same round r; otherwise, out-of-round messages are ignored. Therefore, the two adapted protocols let the round length be long enough, and permit only the event type **e1**, but not **e2** and **e3** (see Section 2.2). As a result, interferences among updates are suppressed, but cannot be eliminated all, thereby admitting errors in estimates. In their comparison between the push-pull and push-sum protocols, the former is faster than the latter, but accuracy is impaired more seriously in the former.

Rao et al. presented two additional push-pull protocols [19] and [21] for asynchrony. Though the authors claim that rounds are asynchronous, these protocols also require the loosely synchronized rounds likewise as in their previous protocol [20]. Instead, if a node receives any push and pull of a different round, it computes some recovery shares in order to resolve the interferences caused by asynchrony. The protocols are classified into optimistic [19] and pessimistic [21] ones according to when nodes update estimates. In the optimistic one [19], nodes update estimates just after receiving pushes or pulls, but the pessimistic one [21] updates the estimates at the completion of every round. The optimistic protocol can handle a number of **e1** events pushed in the same round, while the pessimistic one permits only a single **e1** in a round and abandons subsequent ones. Both of the protocols permit **e3**, but not **e2**, which means that a node can send only one push in a round.

Rao et al. encouraged the length of the loosely synchronized rounds to be as long as the maximum round trip time (RTT)[1]. For secure aggregation, the long round lengths were chosen in the papers [20][19][21], i.e., about from 0.8 to 1.7 *sec*, and thus there is a limitation to accelerate convergence speeds.

[1](RTT) According to the recent study [11], 0.2, 0.5, and 1 *sec* are taken for about 50%, 90%, and 95% of messages to complete a round trip, respectively. The maximum RTT is far longer than 1 *sec*.

Mehyar et al. proposed A1 and A2 algorithms in the distributed averaging domain, which is in principle equivalent to the distributed aggregation [15]; A1 is a push-pull type while A2 is a push-only one.

In A1 protocol, a node that is in the middle of a pairwise update blocks every **e1** event and sends back a negative acknowledgment. As no overlapping updates are allowed, all the events **e1**, **e2**, and **e3** could not happen in A1 protocol. Unlike the protocol of Jelasity et al. [5] doing the homogeneous updates, a pair of nodes in A1 protocol updates their estimates heterogeneously; after one of the pair updates its estimate by using both of the estimates, the other node compensates the counterpart update. In fact, this pairwise update is similar to our proposed protocol's. However, they missed the significant aspect of such heterogeneous updates, which will be explained in Section 4, and thereby A1 pursues the synchronous communication by the blocking mechanism, though it needs no synchronized rounds.

A2 protocol indirectly computes estimates by using additional symmetric values between neighbors. Authors claim that maintaining a symmetric value for every neighbor enables to conserve the system mass in case of node crashes, but symmetric links need to be predefined among nodes. Hence, it is difficult to take advantage of the gossip mechanisms that make protocols topology-independent, scalable, and reliable. In A2, each node not only periodically broadcasts STATE messages containing its estimate to all its neighbors, but also send a REPLY message in order to update symmetric values. Though it is unclear how to treat **e2** and **e3** events in the paper, a node can send a REPLY again to the same neighbor after having received the TCP ACK of the previous REPLY. A2 is asynchronous because it requires no blocking constraints and no synchronized rounds like our proposed protocol. In Section 7.2, we compare our protocol with A2, and give a further analysis with Experiment 2.

Recently, Jesus et al. [6] introduced Flow-Updating to enhance fault-tolerance. Flow Updating is progressive as it is resilient to message loss. Similarly to A2, Flow Updating uses symmetric values called flows. If a crashed node is detected by failure detectors in all its neighbors, the protocol has a way to recover from the damage of the system mass. However, implementing such a practical and instantaneous failure detector is not a simple issue [12]; we will discuss this issue in Sections 6 and 7.

Flow-Updating requires synchronized rounds, in which flows and estimates should be symmetrically pushed to every pair of neighbors (push-only). Though the length of rounds in Flow-Updating can be taken as a half of RTT, round synchronization also imposes some restrictions on the deployment in real-world overlay networks. In addition, Flow-Updating must maintain symmetric links among neighbors, via which node i has to send its flow and estimate to node j if node j sends its ones in the same round. Therefore, before starting Flow-Updating, nodes in overlay networks should set up the symmetric links. This means that Flow-Updating needs a preliminary configuration and has a difficulty in being used in dynamic topology networks. In Section 7.2, Experiment 6 will illustrate such difficulty.

In Table 1, we summarize the constraints of the existing iterative aggregation protocols. Notice that there is no iterative protocol that is not only fully topology-independent, i.e., gossip-based, but also round-free and completely asynchronous. In Sections 4 and 5, we propose a gossip-based round-free push-pull aggregation protocol supporting asynchronous communication.

Type	Protocols	Topology dependency	Allowed events	Rounds
push -only	push-sum [9]	gossip	none	sync.
	Rao's push-sum[20]	gossip	e1	loosely sync.
	A2 [15]	symmetric	async.	round-free
	FU [6]	symmetric	e1, e2	sync.
push -pull	Jelasity [5]	gossip	none	sync.
	Rao's push-pull[20]	gossip	e1	loosely sync.
	Rao '10 [19]	gossip	e1, e3	loosely sync.
	Rao '11 [21]	gossip	e1, e3	loosely sync.
	A1 [15]	gossip	none	round-free

Table 1: Constraints of iterative aggregation protocols

4 Pair Mass Conservation

Let us consider an asynchronous update precisely; in Fig. 2, between two nodes i and j, there are three timings t_1, t_2 and t_3; (t_1) when node i sends a push with a parameter p_s; (t_2) when node j pre-updates $\hat{v}_j$ and sends back a pull with a parameter p_l; and (t_3) when node i post-updates $\hat{v}_i$. Regarding the estimates at these timings, we can present a simple property of a pairwise update.

Definition 1. [*Pair mass conservation*] *For $\hat{v}_i(t)$ and $\hat{v}_j(t)$ that are the estimates of nodes i and j, the pair mass conservation holds on the pairwise update iff:*

$$\lim_{t \to t_3^-} \hat{v}_i(t) + \lim_{t \to t_2^-} \hat{v}_j(t) = \hat{v}_i(t_3) + \hat{v}_j(t_2). \tag{2}$$

In the above one-sided limits of estimate functions, t_3^- and t_2^- represent the times immediately before the updates. Hence, this definition means that a pairwise update, i.e., the pre-update and post-update occurring on the both sides of a pull, should conserve the mass of their estimates. In our proposed protocol, the pair mass conservation is ensured by delivering a pull with the following parameter p_l:

$$p_l \leftarrow \hat{v}_j(t_2) - \lim_{t \to t_2^-} \hat{v}_j(t)$$

After that, node i subtracts p_l from $\hat{v}_i$. Thus, regardless of the pre-update, the post-update can be defined as follows:

$$\hat{v}_i(t_3) = \lim_{t \to t_3^-} \hat{v}_i(t) - p_l = \lim_{t \to t_3^-} \hat{v}_i(t) + \lim_{t \to t_2^-} \hat{v}_j(t) - \hat{v}_j(t_2)$$

Note that $\hat{v}_j$ is updated at t_2 but until t_3 $\hat{v}_i$ is not, which results in the change of the amount of the system mass between t_2 and t_3. In the asynchronous communication, such a change of the system mass is inevitable, and the system mass would not be conserved transiently. For instance, in Fig. 1-(b), at t_1 before interferences happen, the system mass is not preserved as $x_1 + y_1 + z_1 = 0.3 + 0.25 + 0.25 = 0.8$. We consider that the rest of the mass 0.2 is *migrating*. This paper, therefore, introduces $v_{on}(t)$ as a global online value being in transit, which is also included in the system mass as follows:

Definition 2. [*System mass*] *System mass is defined as*

$$\mathfrak{M}(t) = \sum_{\forall k \in \mathcal{N}} \hat{v}_k(t) + v_{on}(t). \tag{3}$$

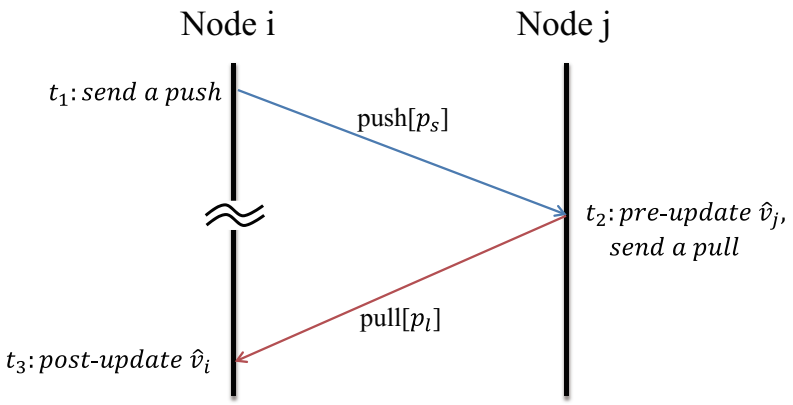


Figure 2: Asynchronous pairwise push-pull update.

After a pre-update, the parameter of a pull, p_l is added to v_{on}, and conversely p_l is subtracted from v_{on} after a post-update. Then, we can present a definition of the system mass conservation for asynchronous protocols.

Definition 3. [*System mass conservation] For any times* t_1 *and* t_2 $(t_1 \neq t_2)$, the system mass conservation *is ensured iff:* $\mathfrak{M}(t_1) = \mathfrak{M}(t_2)$.

The system mass conservation can be guaranteed if every pairwise update satisfies the pair mass conservation. Intuitively, this also can be understood by the idea that pulls induce system mass to *migrate* among nodes via v_{on}. Remind that pull$[x_0]$ in Fig. 1-(b) conveys *estimate* x_0 that will be used in computing the arithmetic mean, but the post-update of **e3** cannot exactly calculate the mass change in its corresponding pre-update due to asynchrony. Contrarily, in our approach pulls deliver mass changes caused by pre-updates, instead of estimates, and post-updates can exactly compensate the change. In other words, each of our pull and post-update based on the pair mass conservation involves only two nodes in a mass migration. Even if multiple updates are asynchronously mingled, each migration is isolated and never interferes with the others. We can prove the following theorem by tracing migrations of the mass.

Theorem 1. *If the pair mass conservation holds for every asynchronous update, the system mass conservation is guaranteed.*

Proof. For any two times t_1 and t_2 $(t_1 < t_2)$, assume $\mathfrak{M}(t_1) = \sum_{\forall k \in \mathcal{N}} \hat{v}_k(t_1) + v_{on}(t_1)$ and $\mathfrak{M}(t_2) = \sum_{\forall k \in \mathcal{N}} \hat{v}_k(t_2) + v_{on}(t_2)$. Given any pull$[p_l]$ between node i and j, let t_s be the time when node i invokes the pull$[p_l]$, and t_e be the time when node j receives the pull$[p_l]$. Then, the following four cases can affect $\mathfrak{M}(t_1)$ and $\mathfrak{M}(t_2)$:

- $t_s < t_1 \leq t_e \leq t_2$: p_l moves from v_{on} to $\hat{v}_j$ at t_e.
- $t_s < t_1 < t_2 < t_e$: No mass moves.
- $t_1 \leq t_s < t_e \leq t_2$: p_l moves from $\hat{v}_i$ to v_{on} at t_s, and after then moves from v_{on} to $\hat{v}_j$ at t_e.
- $t_1 \leq t_s < t_2 < t_e$: p_l moves from $\hat{v}_i$ to v_{on} at t_s.

In any case, this pull does not change the system mass. Hence the system mass conservation is guaranteed. □

In fact, the updates of A1 protocol also satisfied the pair mass conservation, but Mehyar et al. [15] apparently never recognized the property in terms of the asynchronous communication; hence, they provided even a blocking mechanism in order to prevent A1 from actual asynchronous communication (see Section 3). None of previous works [20][19][7][21] recognized the property of A1. In this regard, it is worthy emphasizing the significance of recognizing the pair mass conservation, which is a basis of our proposed protocol making the best use of the asynchrony in the next Section.

5 Multi-pair asynchronous push-pull aggregation

The goal of the average aggregations can be written as

$$\hat{v}_i(t) \approx \frac{\mathfrak{M}(t)}{n} \text{ for } \forall i \in \mathcal{N} \text{ and } t > 0$$

In other words, the estimates of all the nodes should reach the convergence on the average. In our protocol, pulls convey the changes of estimates, and $v_{on}(t)$ indicates the mass of ongoing pulls; hence, $v_{on}(t)$ should approach to zero in the stage of the convergence. To this end, the pre-update needs to be designed so that v_{on} converge to zero. Obviously, taking an arithmetic mean of a pair of the estimates in the pre-updates can make v_{on} converge to zero if estimates are getting equal.

Algorithm 1 Multi-pair asynchronous push-pull aggregation protocol

```
1  Let i and v̂_i be the index and the estimate of this node;
2  Let G_i be the set of neighbors;
3  Let m be the value of multi-pair asynchrony (m < |G_i|);
4
5  PeriodicPushing( ){
6      agg.v̂ ← v̂_i;
7      agg.mode ← PUSH;
8      Let M be a set of m nodes randomly selected from G_i;
9      foreach(node j ∈ M) SEND agg to node j;
10 }
11
12 EventReceiving(agg from node j){
13     if(agg.mode = PUSH){                    // pre-update
14         diff ← (agg.v̂ − v̂_i)/2m;
15         v̂_i ← v̂_i + diff;
16         agg.v̂ ← diff;
17         agg.mode ← PULL;
18         SEND agg to node j;
19     } else if(agg.mode = PULL){             // post-update
20         v̂_i ← v̂_i - agg.v̂;
21     }
22 }
```

To boost the benefits of the asynchrony, we propose *multi-pair asynchronous push-pull aggregation* (MAPPA), which allows every node to send multiple pushes concurrently to different neighbors. MAPPA introduces m as the value of the multi-pair asynchrony. In Algorithm 1, PeriodicPushing method, which is called by each node every cycle, sends m pushes concurrently. Each pre-update takes an arithmetic mean of a pair of only $1/m$ of estimates, and adds it to the rest $(m\text{-}1)/m$ of the estimate. Thus, the pre-update that node j receives $p_s = \hat{v}_i(t_1)$ at t_2 in Fig.

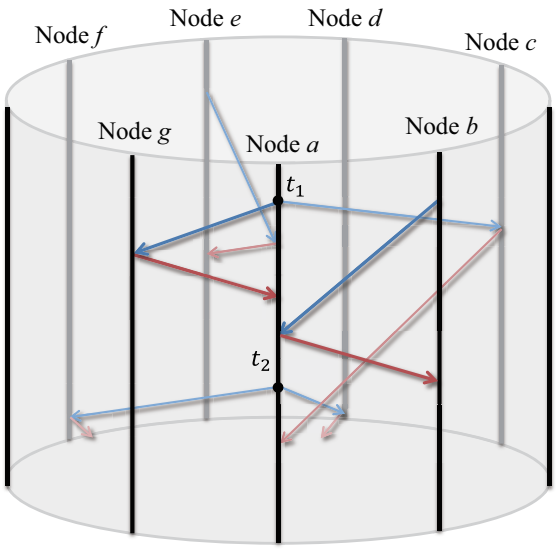


Figure 3: Asynchronous communication of MAPPA for $m = 2$. Only the pushes and pulls associated with node a are given.

2 is computed as follow.

$$\begin{aligned}\hat{v}_j(t_2) &= \frac{(m-1)}{m} \lim_{t \to t_2^-} \hat{v}_j(t) + \frac{1}{2}\left(\frac{\lim_{t \to t_2^-} \hat{v}_j(t)}{m} + \frac{\hat{v}_i(t_1)}{m}\right) \\ &= \lim_{t \to t_2^-} \hat{v}_j(t) + \frac{1}{2m}\left(\hat{v}_i(t_1) - \lim_{t \to t_2^-} \hat{v}_j(t)\right) \end{aligned} \tag{4}$$

In formula (4), the right term corresponds to the change in the updated estimate, which becomes the parameter of the responding pull. In Algorithm 1, the pre-update is presented in lines 14–18 inside EventReceiving method that is invoked upon arrival of messages. Since the post-update of line 20 satisfies the pair mass conservation, the system mass is conserved in MAPPA despite the multi-pair asynchrony.

In Fig. 3, the asynchronous communication of MAPPA is illustrated for m=2, where we present the pushes and pulls related to only node a for simplicity's sake. At t_1, node a sends two pushes at once to neighbors c and g (**e2**), and asynchronously receives their responding pulls. Node a receives two pushes from nodes b and e, and responds to them with pulls (**e1**). At t_2 when a cycle passes, node a can send m=2 pushes again regardless of cycles of the other nodes (round-free), even if some pulls replying to the previously sent pushes, e.g., the pull from node c, have not yet arrived (**e3**). In result, MAPPA is round-free and permits all the asynchronous event types.

Although each node chooses different m, estimates can converge to the correct value. Nevertheless, if a node pushes its estimates in parallel to m neighbors by PeriodicPushing method, it is natural that each of the m neighbors uses the same m in EventReceiving method because it is considered that m parallel arithmetic means are taken by dividing the estimate into m pieces. If each node uses different m, it should send every push with its m. Therefore, we fix the values of m at all the nodes for convenience sake and for simplicity in the analysis.

In principle, the average aggregation comes close to the average by distributing extremely segmentalized *shares* (pieces) of the initial values $\mathbf{v}=\{v_1, v_2, \cdots, v_n\}$ to all the nodes. MAPPA accelerates the convergence by encouraging an estimate to be more extensively distributed to other nodes and to be mixed with numerous shares. In Fig. 3, at the second cycle t_2, the estimate of node a would be reconstituted with the shares of the values of nodes b, e, and g, but not yet c.

Just for analysis purpose, let us assume that cycles are synchronized and every pairwise update is completed at the end of each cycle. Initially, an estimate of each node has a single full share of its own initial value. By m pairwise updates in a cycle, an estimate is mixed with the shares of m estimates belonging to the neighbor set $\mathcal{G}$. On the other hand, as every node sends m pushes, a node asymptotically receives m pushes from other nodes on the assumption that each node has uniformly random neighbors. As a result, at the end of a cycle, the estimate of a node includes the shares of $2m+1$ estimates of the previous cycle. Let $S_m(c)$ be the function that returns the number of shares of initial values which an estimate includes at cycle c, permitting duplicates of initial values. Based on these inferences, at each cycle, $S_m(c)$ can be derived as a geometric sequence:

$$\begin{aligned} S_m(0) &= 1 \\ S_m(1) &= (2m+1)S_m(0) = (2m+1) \\ S_m(2) &= (2m+1)S_m(1) = (2m+1)^2 \\ &\cdots \\ S_m(c) &= (2m+1)S_m(c-1) = (2m+1)^c \end{aligned} \tag{5}$$

According to this formula, the number of shares grows exponentially with growth rate m. Thus, a larger value of the multi-pair asynchrony m leads to a more rapid convergence. For example, regarding m=1, 4, and 13, $S_1(c) = 3^c$, $S_4(c) = 9^c = 3^{2c}$, and $S_{13}(c) = 27^c = 3^{3c}$ which means MAPPAs with m=4 and 13 could take half cycles and one-third cycles of m=1, respectively, to include the same number of shares.

Indeed, shares are more rapidly mixed than the indication of the formula because updates are asynchronously (immediately) reflected on estimates. In Fig. 3, when node a receives a push from node b, the estimate of node a has already included the shares of nodes e and g; hence, the pre-update of the push is delivered with those shares. Though such nondeterministic cases are difficult to be modeled by a formula, Experiment 4 in Section 7.2 shows that MAPPA with m=4 takes far less cycles than the expectation of formula (5).

The initial values in $\mathbf{v} = \{v_1, v_2, \cdots, v_n\}$ can be selected as shares, allowing duplicates. The probability for a certain value in $\mathbf{v}$ to be chosen depends on the characteristics of network topologies. If the underlying topology is a purely random graph, built according to the Erdős-Rényi (ER) model [2], any initial values could be selected with a similar probability; thus, estimates can rapidly converge. Meanwhile, if the topology is a locally clustered graph such as Watts-Strogatz graph with $\beta = 0$ [22], geographically dispersed initial values are difficult to be mixed into an estimate, which ends in a slow convergence (see Experiment 5).

Meanwhile, under the asynchrony, some pathological problems impeding convergences are also observable as shown in Fig. 4. First, look at the two pairwise updates between nodes 1 and 2 (see the chunky arrows). If two pairwise updates cross, both of the effects are offset each other; in the post-updates, nodes 1 and 2 recovers as much values as the ones changed by the corresponding pre-updates. Besides, if the initial deviation of estimates is large and if the cycle of sending pushes is much shorter than the period of a pairwise update, a respectable amount of the system mass might hover in transit. If $\mathfrak{M}(0)$=1 in Fig. 4, $v_{on}(t_1)$=$\frac{31}{32}$ at t_1 due to the ongoing pulls. As subsequent updates cannot reflect the whole system mass, estimates may

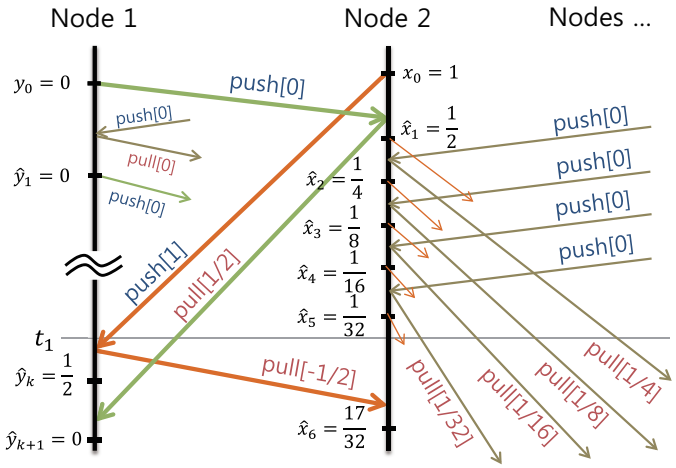


Figure 4: Pathological problems of asynchronous pairwise updates. Assume that except node 2, all the other nodes have initial value 0.

oscillate without convergence (see Experiment 3). To make things worse, if some of those pulls accompanying a large portion of system mass are lost, the system mass is seriously damaged, which means the aggregation is fault-intolerant.

Consequently, various factors, such as multi-pair asynchrony values m, cycle lengths, and network topologies, should be considered for the convergence. Since analytic analysis has a limitation on the consideration of all of those factors, we will present various evaluation results in Section 7.

6 Asynchronous overlapped restarting technique

Aggregations can be used for *continuous monitoring* that unceasingly provides effective estimates. In dynamic systems, however, network churns incurred by crashes and inflows of nodes lead to changes in the system mass. The mass added by a newly joined node is spontaneously reflected to the aggregation. Though the sign-off technique can deal with voluntary departures of nodes [21], a sudden crash of a node results in damage of the system mass, which cannot be recovered in the push-pull aggregation by itself. To recover from the damaged mass of dynamic networks, Jelasity et al. proposed the restarting technique [5].

Although A2 protocol [15] and Flow Updating [7] can correct damage of crashes by maintaining symmetric values (see Section 3), such a correction is only possible when every node can detect failures of its neighbors. However, it also takes a certain amount of time to detect failures securely, as stated in the paper [12], after then additional time is needed for estimates to re-converge to the corrected one. If the convergence time from restarting is shorter than the sum of the failure detection and re-convergence times, the restarting technique might be effective rather than A2 and Flow Updating.

In the restarting technique of Jelasity et al., all nodes have to synchronize their restarts based on *epoch*, which is a fixed number of rounds. As no synchronized rounds are used, MAPPA restarts aggregations asynchronously based on *hops*. An instance of an aggregation records hops which increase whenever sending a push. If the hop of a local instance is different from the one of any received push or pull, the larger one becomes the hops of the local instance and the responding pull.

Instead of restarting a new instance in series, we let two instances of aggregations to overlap their lifetimes so that a new instance should warm up for a certain period. Note that all of the iterative aggregations go through a disequilibrium in which nodes have unstable and untrustworthy estimates. To avoid such a disequilibrium, we let each node to prepare two instances in an aggregation called adult and child, and only the value of adult is used as an estimate. Initially, both of adult and child begin the same pairwise updates. If a child reaches to a predefined number of hops, the child *is promoted* to adult, and a new instance is generated so that a child should restart with its initial value.

An aggregation containing an adult and a child has a generation number. If a node promotes its child, it augments the generation number of its aggregation. If a node receives a push or a pull with an aggregation of a higher generation number, it tunes its aggregation to the received one by promoting its current child and restarting a new child. In the opposite case that a node receives a push of a lower generation number, the node likewise tunes the aggregation of the responding pull to its own aggregation. If a node receives a pull of a lower generation number, its aggregation is updated by considering the generation number of the pull.

In this way, MAPPA can asynchronously restart new instances in the overlapping manner. This can ensure all the nodes monitor convergent estimates most of the times. In Experiment 7, we will show the effects of this asynchronous overlapped restarting technique and compare it with the behaviors of Flow-Updating protocol.

7 Evaluation

7.1 Experimental setup

We implemented MAPPA with the event-driven simulator, Peersim [17]. Every node sends pushes on the same length of cycle, but the cycles of nodes are not synchronized. We let the propagation latency of any push or pull to be distributed uniformly between 50 and 350 ms. Considering that about 90% of messages complete a round trip within 500 ms [11], this unidirectional latency is reasonable and long enough.

In our evaluation, the count aggregation, which estimates the network size n, is carried out because it is the most representative and extreme case of the average aggregation [5]. It begins over the network which has only one node with value 1 and the other $n-1$ nodes with value 0. As the aggregation proceeds, each estimate approaches to the average $1/n$; thus, we can obtain the network size by taking the reciprocal.

We have MAPPA run generally on Erdős-Rényi (ER) random graphs [2] as underlying networks, but also evaluate Watts-Strogatz (WS) model networks [22] in Experiment 5. Most of our experiments are performed with n=1000 nodes, and every node has twenty gossip neighbors, i.e., $|\mathcal{G}|$=20. Following the gossip principle, each node sends pushes to only m neighbors in $\mathcal{G}$. The size of $\mathcal{G}$ rarely affects the performance of MAPPA, which can be identified in Experiment 2. Instead, a large number of neighbors reinforces reliability against node crashes. According to the paper [8], the network where each node has $\log n$ neighbors can avoid a partitioning, even if half nodes suddenly die. To emulate dynamic topologies in Experiment 5 and to eliminate crashed neighbors in Experiment 7, we use *Simple Newscast* [4].

We have conducted each experiment 30 times with different random seeds, and took averages of the estimates of the 30 runs for each individual node. This section presents two sorts of time-series graphs: estimates distribution graph and coefficient of variation of the root-mean-square deviation, abbreviated to CV(RMSD), graph. The former illustrates how estimates diverge and converge along time by marking spots of the reciprocals of estimates of all nodes. Meanwhile, CV(RMSD) is a normalized RMSD to a correct mean, i.e., RMSD/$\bar{\mathbf{v}}$. Unlike the standard deviation relying on the mean of given populations, RMSD is a deviation from a given mean $\bar{\mathbf{v}}$, and is computed by the following formula.

$$RMSD(t) = \sqrt{\frac{\sum_{i=1}^{n}(\hat{v}_i(t) - \bar{\mathbf{v}})^2}{n}} \tag{6}$$

CV(RMSD) is a good measure to compare convergence speeds; the faster the values of CV(RMSD) approach to zero, the more rapid its convergence speed is.

This section presents seven experiments with the following six parameters on the count aggregations.

- n: the number of nodes in the network.
- c: the cycle length.
- m: the multi-pair asynchrony value.
- e: the ratio of message loss in delivery.
- tc: the length of the cycle at which the network topology changes by *Simple Newscast* protocol. If $tc = \infty$, network topology is static.
- h: the number of hops to restart a child instance (see Section 6).

The parameters fixed in each experiment are specified in the caption of the graph of each result.

7.2 Experiment results

Experiment 1: Pair vs. Fair asynchronous push-pull aggregations [Fig. 5]

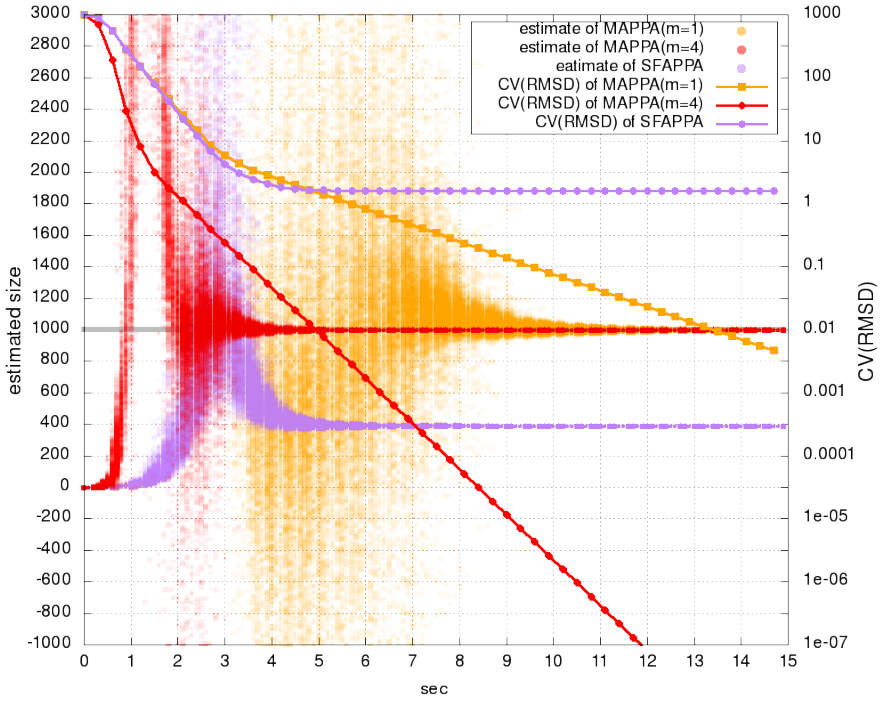


Figure 5: [Exp. 1: n=1000, c=400ms, e=0.0, tc=∞, h=∞] Estimates distribution graphs (left y-axis) and CV(RMSD) graphs (right y-axis) of MAPPA and single fair asynchronous push-pull aggregation (SFAPPA).

Let us call the asynchronous push-pull aggregation following Jelasity et al. as "single *fair* asynchronous push-pull aggregation (SFAPPA)" because each node sends a single push and both of nodes fairly update their estimates by arithmetic means. As interferences among asynchronous updates hinder SFAPPA from satisfying the system mass conservation, all the estimates converge to incorrect values, as shown in estimates distribution graphs of Fig. 5. As a result, high values of CV(RMSD) is lasting in the convergence state because $(\hat{v}_i(t) - \bar{\mathbf{v}})^2$ in formula (6) does not approach to zero.

In meantime, our proposed protocol MAPPA always converges to the correct network size thanks to the pair mass conservation. Notice that in the convergence state, due to the post-update wherein differences are subtracted, estimates can be negative; hence, as estimates oscillate between wider range, MAPPA with m=1 somewhat slowly converges than SFAPPA. Surprisingly, convergence speeds are significantly improved if a node concurrently sends multiple pushes. If m=4, it takes only less than 5 *sec* to reach the CV(RMSD) to 0.01, while m=1 takes more than 13 *sec*.

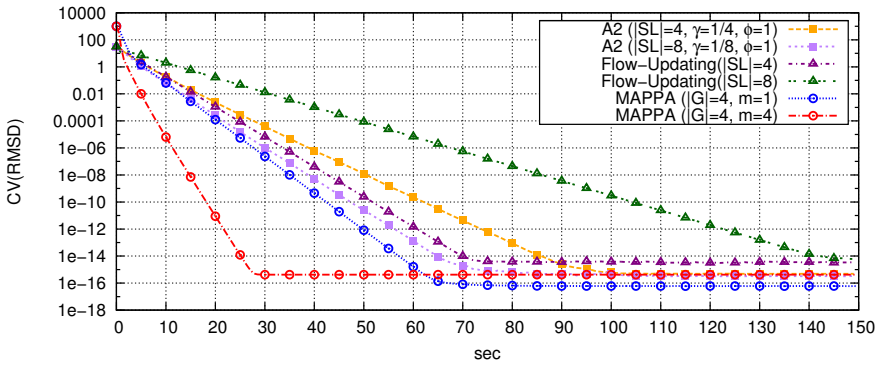


Figure 6: [Exp. 2: n=1000, c=400ms, e=0.0, tc=∞, h=∞] CV(RMSD) graphs of MAPPA, A2, and Flow-Updating.

Experiment 2: Comparison with other aggregation protocols [Fig. 6]

We compare MAPPA with some other aggregation protocols. As a comparison group, A2 and Flow-Updating are chosen because each of them is, respectively, the only round-free asynchronous protocol and one of the up-to-date protocols. Since these two protocols exchange messages only through predefined symmetric links, we predefined a set of symmetric links $\mathcal{SL}$ for each node (see Section 3). We run the two protocols over the networks of $|\mathcal{SL}|$=4 and $|\mathcal{SL}|$=8 utilizing all the links. On the other hand, MAPPA works on asymmetric networks, wherein each node has a set of neighbors $\mathcal{G}$. We constraint $|\mathcal{G}|$=4 for the sake of fairness, and each node pushes to only m=1 or m=4 neighbors. It is worthy noting that the performance of $|\mathcal{G}|$=4 is similar to that of $|\mathcal{G}|$=20, i.e., the default size of $\mathcal{G}$ in other experiments.

According to the paper [15], A2 requires two parameters γ and ϕ for updating estimates and symmetric values. The authors claim they should satisfy the conditions $0 < \gamma < \frac{1}{|\mathcal{SL}|+1}$ and $0 < \phi < \frac{1}{2}$ in order to achieve the convergence, but no exact values were recommended in the paper. Moreover, it is unclear whether each node pushes REPLY messages to multiple links or randomly chosen one. After implementing A2 with event-driven Peersim, we have tried to tune the protocol, and found out two facts not given in the paper [15]. First, concurrent pushing to multiple neighbors reduces convergence time also in A2. Second, even if the conditions of the two parameters hold, estimates frequently diverged. Instead, we found that if $\gamma = 1$ and $\phi = \frac{1}{|\mathcal{SL}|}$, A2 can always achieve more rapid and stable convergence. In fact, these conditions make A2 be similar to MAPPA.

In A2 protocol, we make REPLY messages toward each neighbor be propagated like a relay race; that is, each node sends a new REPLY immediately after receiving an ACK of the previous REPLY. Estimates are accompanied with REPLY and ACK and updated every 100 ms. In Flow-Updating, we let the round length be 500 ms so that messages of the maximum 350 ms delay should be delivered even if rounds are slightly out of step.

Fig. 6 shows that MAPPA is faster, despite m=1, than the other protocols. Like m of MAPPA, a larger $|\mathcal{SL}|$ of A2 leads to a more rapid convergence. Interestingly, Flow-Updating is much faster with $|\mathcal{SL}|$=4 than with $|\mathcal{SL}|$=8; namely, parallel updating rather impedes convergences. For rapid aggregations, Flow-Updating would need to constitute network topologies of few links, but of short node-to-node distances (see Experiment 5). However, since overlay networks need enough links to avoid a partitioning against network churns, Flow-Updating

could encounter a tug-of-war between reliability and performance. In any case, even if message overhead is considered and the round synchronization time is not included, MAPPA with m=4 outperforms all the other aggregation protocols.

Experiment 3: The effects of the cycle length [Fig. 7]

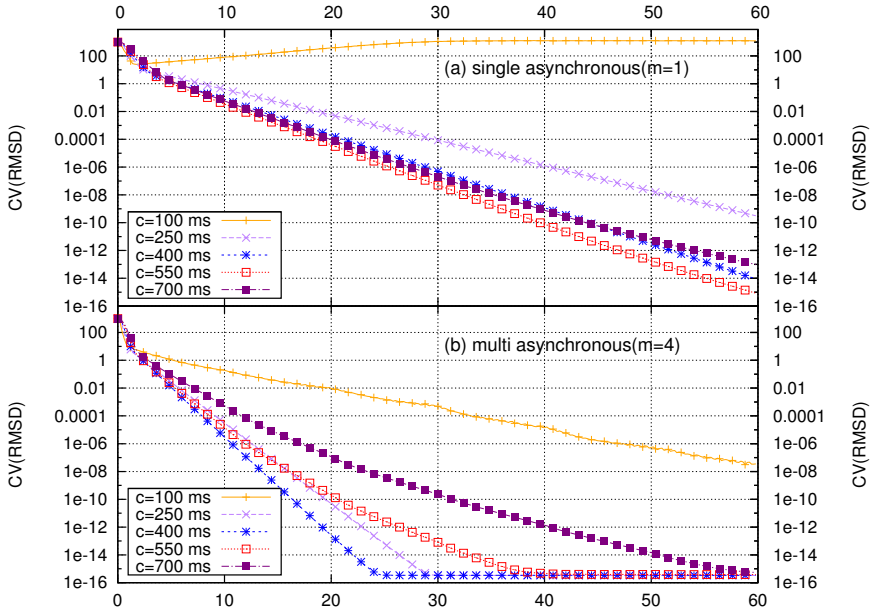


Figure 7: [Exp. 3: n=1000, e=0.0, tc=10 *sec*, h=∞] CV(RMSD) graphs with respect to cycle lengths c.

The result of Fig. 7 indicates that cycle lengths are a sensitive factor affecting convergence speeds. As aforementioned in Section 5, if cycles are too short, aggregations heavily go through the pathological problems such as excessive wafting of the system mass to v_{on}; hence, convergences are delayed, or estimates rather diverge as illustrated in the graph of c=100 ms in Fig. 7-(a) where m=1. If m=4, as segmentalized estimates are distributed to more nodes, negative effects are slightly suppressed; thus, even under c=100 ms, MAPPA with m=4 converges. In any case, the convergence speeds increase to some degree of cycle lengths, but there exist certain cycle lengths when convergence speeds go into reverse; regarding m=1, the cycle length of the fastest convergence speed is c=550 ms, and then longer cycle lengths make convergences slower.

Meanwhile, we observed that the convergence with 550 ms cycle insignificantly is faster than that of 400 ms cycle over a static topology (tc=∞), but over the dynamic topology (tc=10 *sec*) the convergence is faster with 400 ms cycle than with 550 ms one as shown in Fig. 7-(b). According to additional experiments, against the propagation delay of 50$\sim$350 ms, MAPPA with c=400 ms is generally fastest regardless of multi-pair asynchrony ($m > 1$) and network topologies. As a result, we choose 400 ms as a default cycle length. The effects of dynamic topologies will be precisely discussed in Experiment 5.

Experiment 4: The effects of multi-pair asynchrony and network size [Fig. 8]

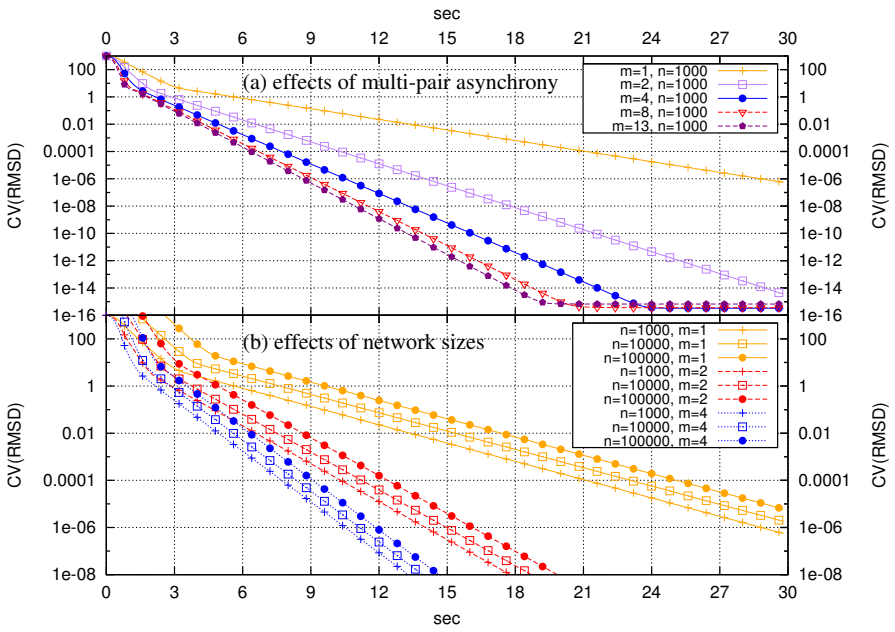


Figure 8: [Exp. 3: c=400 ms, e=0.0, tc=∞, h=∞] CV(RMSD) graphs with respect to different levels of multi-pair asynchronism and different network size.

Clearly, Fig. 8-(a) proves that multi-pair asynchrony can improve convergence speeds. As discussed in Section 4, the convergences are accelerated *more than* the prediction of formula (5): $S_m(c) = (2m+1)^c$. For example, the rise of m from 1 to only 2 almost halves the cycles required to reach the same convergence states. However, according to the formula, we can infer that increasing m logarithmically reduces c required to have the same number of shares. In other words, as m is getting larger, the amount of the reduction in convergence times declines sharply. Therefore, m should be chosen considering the trade-off between performance and message overhead that is proportional to m; so, we use m=4 as default in our evaluation.

Network size has a weaker correlation with convergence speeds than the multi-pair asynchrony as shown in Fig. 8-(b). Though a larger network converges a little slowly, the inclinations of CV(RMSD) graphs are invariable with respect to n. Considering the results of Fig. 8-(b), it is highly expected that the graphs are horizontally shifted to the right proportionally to $\log n$.

Experiment 5: The effects of the network topologies[Fig.9]

To enhance scalability and reliability in overlay networks, gossip mechanisms encourage topologies to be altered dynamically [3][4]. In this experiment, we make each node dynamically change its neighbors every tc second by the *Simple Newscast* protocol [4]. Fig. 9-(a) shows that although too frequent changes of topologies retard the latter half convergence, the delays of the primary stage are negligible. As stated in Experiment 3, the cycle length c influences convergence in dynamic topologies. Short cycle lengths make convergence speeds almost identical, but with long cycle lengths the latter half of convergences, at which enough accuracy is already secured, is slightly slacker.

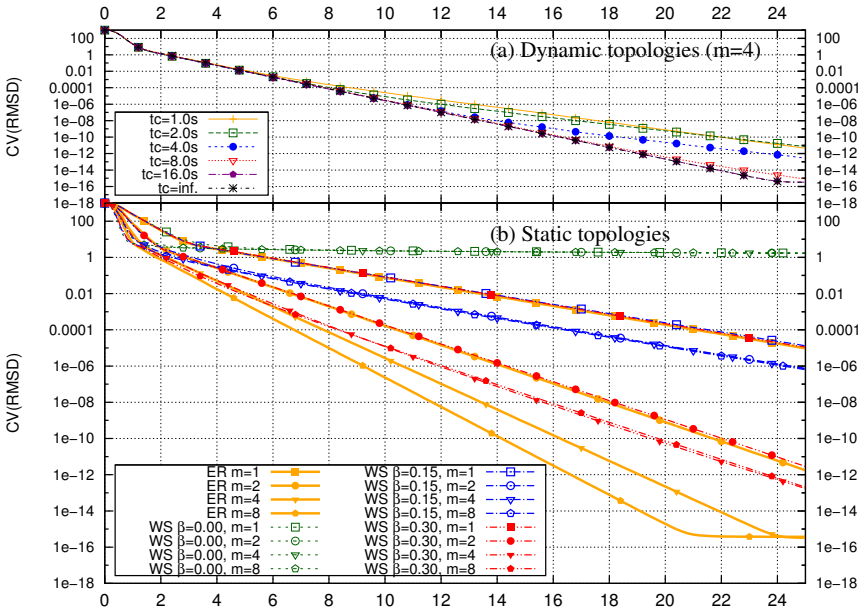


Figure 9: [Exp. 4: n=1000, c=400 ms, m=4, e=0.0, $h = \infty$] CV(RMSD) graphs with respect to network topologies.

Characteristics of network topologies are crucial to the convergence time. Our evaluations have been conducted over the topologies following Erdős-Rényi (ER) model [2]. ER model produces purely random graphs with a short average node-to-node distance and a small clustering coefficient. Watts and Strogatz addressed *small world properties* that many real-world networks have a small average node-to-node distance, but a higher clustering coefficient than the graphs of ER model [22]. They presented Watts-Strogatz (WS) model, in which nodes are wired in the shape of a ring lattice so that random graphs should have a high clustering coefficient. The model also introduced a parameter β, which is a probability to rewire any pair of randomly selected nodes. If β=0, no nodes are rewired, and thus random graphs have a long average node-to-node distance. If β=1, all nodes are randomly rewired, thereby producing pure random graphs similar to those of ER model.

In Fig. 9-(b), aggregations are performed over ER and WS random graphs. With respect to β of the WS model, convergences take on significantly different aspects. If β=0, estimates hardly converge despite a large m. Though aggregations succeed in convergence for $\beta > 0$, there are limits, correlating with β, in reducing convergence times. If β=0.15 and β=0.3, the convergence speeds are no longer improving for $m \geq 2$ and $m \geq 3$, respectively. Therefore, we can presume that networks of WS graphs have maximum feasible convergence speeds with respect to β, and MAPPA can make the best use of the networks. However, before the speeds are saturated in the WS graphs, i.e., $m < 2$ and $m < 3$ for β=0.15 and β=0.3, respectively, for equal m, the aggregations over the WS graphs almost coincides with the aggregation over the ER graph regardless of β; e.g., for m=1, the aggregation over the ER graph almost coincides with those over the WS graphs of β=0.15 and β=0.30.

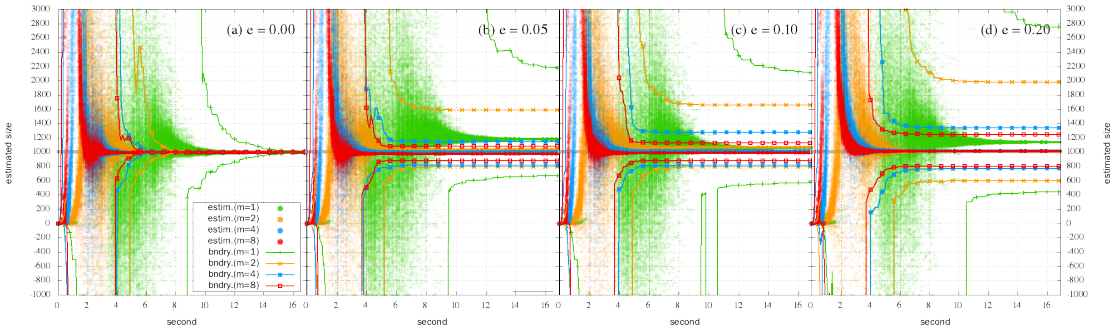


Figure 10: [Exp. 5: n=1000, c=400ms, tc=∞, h=∞] Estimates distribution graphs with respect to different message loss and multi-pair asynchrony.

Experiment 6: The effects of message loss and multi-pair asynchrony [Fig. 10]

MAPPA makes a significant improvement in fault-tolerance. To show the effects of message loss, we drop 5%, 10%, and 20% of messages against different levels of multi-pair asynchrony. In Fig. 10, estimates of 1000 nodes, which are obtained by averaging 30 runs per node, are distributed for different fault ratios. Besides, a boundary of each distribution is also marked, which indicates the maximum and minimum of estimates any node has along time without averaging 30 runs. Regardless of fault ratios, estimates always converge to a certain value, but a higher fault ratio leads to a larger deviation of convergence values from the correct average. Nevertheless, as we can identify from the boundaries, a larger m remarkably reduces the deviation. Such an improvement in the fault tolerance is originated from the fact that the parameters of pulls are getting smaller as m grows, thereby mitigating the damage of system mass against message loss.

Meanwhile, note that the converged estimates averaging 30 runs nearly approach to the correct value. Jelasity et al. enhanced accuracy in the presence of faults by averaging estimates of multiple instances [5]. Our results prove that the technique is still effective in MAPPA.

Experiment 7: Continuous monitoring of a dynamic network [Fig. 11]

We compare MAPPA and Flow-Updating (FU) with a scenario of continuous monitoring over a dynamic network. In this scenario, there are initially 1000 nodes, and we impose three types of network churns for 200 seconds as below:

- Sudden churn: At 25 *sec*, 400 random nodes suddenly crash. After then new 400 nodes join the network at 50 *sec*.
- Gradual churn: From 75 *sec* to 80 *sec*, 80 random nodes gradually crash every second. After then new 80 nodes join gradually every second from 105 *sec* to 110 *sec*.
- Mixed churn: At 135 *sec*, the crashes of 400 random nodes coincide with the inflows of new 200 nodes. After then 200 random nodes crash, and simultaneously new 400 nodes join at 160 *sec*.

We run MAPPA and Flow-Updating under different conditions reflecting their characteristics. MAPPA restarts a child instance whenever its hop count encounters h=35 and h=50 (see Section

6). Meanwhile, we let each node of Flow-Updating be connected along maximum 4 and 8 symmetric links, i.e., $|\mathcal{SL}|$=4 and $|\mathcal{SL}|$=8, in advance, and the links are fully utilized to exchange messages every 500 ms round. In the case of the inflows, new nodes are connected so that $|\mathcal{SL}|$ of each node should not exceed the maximum symmetric links, though those connections are impossible without global topology information.

Fig. 11 shows the results. In graph (a), we compare CV(RMSD) of MAPPA and Flow-Updating for the message loss ratio e=0.0. Generally, MAPPA has more accurate estimates over the dynamic networks. The other graphs present the estimates distributions; graphs (b)-(e) for e=0.0 and graphs (f)-(i) for e=0.1. In graphs (b), (c), (f) and (g), MAPPA allows the majority of nodes to have nearly convergent estimates thanks to the asynchronous overlapped restarting technique, while estimates of Flow-Updating are divergent most of times as shown in graphs (d), (e), (h) and (i).

In detail, MAPPA can instantly compute the initial network size. If MAPPAs with h=35 and h=50 meet the crashes of the sudden and gradual churns, they reflect them after about 10 and 15 seconds, respectively. Those relatively long time-lags are originated from the warming-up time of the overlapped restarting technique. Contrarily, the large inflows of the sudden and gradual churns are caught up with by MAPPA in an instance. In the case of the mixed churn, the inflow hidden by a large crash immediately increases the estimates. Since the system mass of 400 nodes, each of which estimate is about 0.001, is vanished, the remained system mass, about 0.6, is distributed to the new 200 nodes, and thus the estimates soars around 800/0.6≈1333. After restarting, the system mass is reset to 1. Likewise, at the end of the mixed churn, the similar effects occur. In MAPPA, hop counts affect the responsiveness against the crashes, but take no effects on inflows.

Flow-Updating, at the beginning, takes longer times to obtain effective estimates than MAPPA; if $|\mathcal{SL}|$=8, nodes can hardly have convergent estimates up until the sudden churn. Like MAPPA, Flow-Updating by itself cannot reflect any crash. In these experiments, a node regards its neighbor as crashed one if it receives no message subsequently 8 times, i.e., 4 seconds. We took this specified number of times because failures were falsely detected in the presence of 10% of message loss if the number of times is less than 8. When wholesale nodes randomly die at the beginning of the sudden and gradual churns, some nodes were isolated over the network of $|\mathcal{SL}|$=4, i.e., network partitioning. As a result, the estimates fail to converge during the sudden and gradual churns for $|\mathcal{SL}|$=4. In the network of $|\mathcal{SL}|$=8, partitioning can be avoided, but the overall aggregation is getting slower. Regarding the inflows of the sudden and gradual churns, convergence of Flow-Updating is relatively faster than at its initial stage, but still slower than MAPPA. Interestingly, the gradual inflows make the estimates more divergent than the sudden inflow, and thus more time is spent on re-converging. In the case of the mixed churn, convergence of the end is faster than that of the beginning.

We also conducted the same experiments with 10% message loss (e=0.1). MAPPA has aggregation errors in convergent estimates, but the errors seem not to seriously distort the tendency of changes of the network size. In Flow-Updating, message loss retards the convergence, and thus effective estimates are barely obtained by the end of the churns. Nevertheless, if networks are unchanged for a long time, Flow-Updating can converge to the correct network sizes despite message loss.

To sum up, MAPPA outperforms Flow-Updating in convergence speeds, even if the restarting technique is used. The overlapped restarting technique rather enables all nodes to maintain convergent estimates. We expect that this property is useful because algorithms operating on distributed nodes can use almost convergent aggregated data as inputs by MAPPA. Meanwhile, in this experiment, we identify that network churns might demand on Flow-Updating additional configurations to repair symmetric links; thus it is not completely topology independent. Addi-

tionally, the efforts to synchronize rounds would be also required to begin Flow-Updating over large networks. Nevertheless, Flow-Updating is an innovative approach because it can compute correct estimates in spite of message loss. In conclusion, MAPPA is in better position over dynamic networks while Flow-Updating is better in faulty environments over static networks.

7.3 Discussion on message overhead

In our evaluation, the message overhead of MAPPA, i.e., the number of messages for a certain period of time, is comparable to that of other iterative aggregation protocols A2 and Flow-Updating, because they also send periodic messages in parallel to multiple neighbors. However, from the cost-effective point of view MAPPA is superior to the other approaches, as shown in Experiments 2 and 7.

In MAPPA, the number of messages along time t follows the formula: $NM(t) = 2mnt/c$. Apparently, as the level of the multi-asynchrony increases, message overhead with respect to a specific time rises. For instance, if m=4, the overhead is 4 times higher than that with m=1 after the same time t. However, the message overhead required to achieve a certain level of accuracy is acceptable in spite of $m > 1$. To illustrate, let us consider the time to reach the accuracy CV(RMSD)(t) = 0.01 in Fig. 8-(a). For each of m=1, 2, and 4, it takes about t=13.5 *sec*, 6.2 *sec*, and 4.5 *sec*, respectively. According to the above formula, MAPPAs with m=2 and m=4 require 91% and 133% of messages used in MAPPA of m=1 reaching CV(RMSD)(t)=0.01. Therefore, if the aggregation is just one-off event that computes a predefined accuracy level of estimates, MAPPA with an appropriate m incurs affordable overhead, but provides considerable benefit not only in the convergence time but also in the fault-tolerance, as addressed in Experiments 4 and 6.

8 Conclusion

This paper proposed the gossip-based multi-pair asynchronous push-pull aggregation protocol (MAPPA) that allows asynchronous communication of push-pull style updates and requires no synchronized rounds. We introduced the pair mass conservation as a property that ensures the system mass conservation by isolating migrations of the system mass incurred by asynchronous updates. Based on this property, we addressed the multi-pair asynchrony, which enables MAPPA significantly to reduce convergence times and to enhance fault-tolerance to message loss. Since MAPPA is round-free and follows the principles of gossip mechanisms, it is completely topology independent and unconstrainedly deployable in large scale networks. In addition, we also suggested the asynchronous overlapped restarting technique that can provide mostly convergent estimates in dynamic networks. We have showed these achievements with a number of experiments and comparisons with other iterative protocols. The comparisons show that MAPPA holds a dominant position in the practicality and performance, compared to other iterative aggregation protocols.

References

[1] O. Babaoglu, T. Binci, M. Jelasity, and A. Montresor, "Firefly-inspired heartbeat synchronization in overlay networks," in *Proceeddings of IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO '07)*, July 2007, pp. 77 –86.

[2] P. Erdős and A. Rényi, "On the evolution of random graphs," in *Publication of the mathematical institute of the hungarian academy of sciences*, 1960, pp. 17–61.

[3] P. T. Eugster, R. Guerraoui, S. B. Handurukande, P. Kouznetsov, and A.-M. Kermarrec, "Lightweight probabilistic broadcast," *ACM Transactions on Computer System*, vol. 21, no. 4, pp. 341–374, Nov. 2003.

[4] M. Jelasity, R. Guerraoui, A.-M. Kermarrec, and M. van Steen, "The peer sampling service: experimental evaluation of unstructured gossip-based implementations," in *Proceedings of ACM/IFIP/USENIX international conference on Middleware*. New York, NY, USA: Springer-Verlag New York, Inc., 2004, pp. 79–98.

[5] M. Jelasity, A. Montresor, and O. Babaoglu, "Gossip-based aggregation in large dynamic networks," *ACM Transaction on Computer System*, vol. 23, pp. 219–252, Aug. 2005.

[6] P. Jesus, C. Baquero, and P. S. Almeida, "Fault-tolerant aggregation for dynamic networks," in *Proceedings of the 29th IEEE Symposium on Reliable Distributed Systems (SRDS)*, ser. SRDS '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 37–43.

[7] ——, "A survey of distributed data aggregation algorithms," University of Minho, Braga, Portugal, Tech. Rep., Sept. 2011.

[8] M. F. Kaashoek and D. R. Karger, "Koorde: A simple degree-optimal distributed hash table," in *Proceedings of International Workshop on Peer-to-peer Systems (IPTPS)*, 2003, pp. 98–107.

[9] D. Kempe, A. Dobra, and J. Gehrke, "Gossip-based computation of aggregate information," in *Proceedings of IEEE Symposium on Foundations of Computer Science (FOCS)*, Oct. 2003, pp. 482 – 491.

[10] D. Kostoulas, D. Psaltoulis, I. Gupta, K. Birman, and A. Demers, "Decentralized schemes for size estimation in large and dynamic groups," in *Proceedings of the IEEE International Symposium on Network Computing and Applications (ISNCA)*, July 2005, pp. 41 –48.

[11] D. Lee, K. Cho, G. Iannaccone, and S. Moon, "Has internet delay gotten better or worse?" in *Proceedings of the International Conference on Future Internet Technologies (CFI)*. New York, NY, USA: ACM, 2010, pp. 51–54.

[12] Z. Li, L. Yuan, P. Mohapatra, and C.-N. Chuah, "On the analysis of overlay failure detection and recovery," *Computer Networks*, vol. 51, no. 13, pp. 3828 – 3843, 2007.

[13] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, "Tag: a tiny aggregation service for ad-hoc sensor networks," *SIGOPS Operating System Review*, vol. 36, no. SI, pp. 131–146, Dec. 2002.

[14] L. Massoulié, E. L. Merrer, A.-M. Kermarrec, and A. Ganesh, "Peer counting and sampling in overlay networks: random walk methods," in *Proceedings of the ACM symposium on Principles of distributed computing (PODC)*, New York, NY, USA, 2006, pp. 123–132.

[15] M. Mehyar, D. Spanos, J. Pongsajapan, S. H. Low, and R. M. Murray, "Asynchronous distributed averaging on communication networks," *IEEE/ACM Transactions on Networking*, vol. 15, pp. 512–520, June 2007.

[16] E. L. Merrer, A.-M. Kermarrec, and L. Massoulie, "Peer to peer size estimation in large and dynamic networks: A comparative study," in *Proceedings of IEEE International Symposium on of High Performance Distributed Computing (HPDC)*, 0-0 2006, pp. 7 –17.

[17] A. Montresor and M. Jelasity, "Peersim: A scalable p2p simulator," in *Proceedings of IEEE International Conference on Peer-to-Peer Computing (P2P)*, sept. 2009, pp. 99 –100.

[18] A. Montresor, M. Jelasity, and O. Babaoglu, "Robust aggregation protocols for large-scale overlay networks," in *Proceedings of the International Conference on Dependable Systems and Networks (ICDSN)*. Washington, DC, USA: IEEE Computer Society, 2004, pp. 19–.

[19] I. Rao, A. Harwood, and S. Karunasekera, "A gossip-based asynchronous aggregation protocol for p2p systems," in *Proceedings of 35th IEEE Conference on Local Computer Networks (LCN)*, Oct. 2010, pp. 248 –251.

[20] ——, "Impacts of asynchrony on epidemic-style aggregation protocols," in *Proceedings of 16th IEEE International Conference on Parallel and Distributed Systems (ICPADS), 2010*, Dec. 2010, pp. 601 –608.

[21] ——, "Gossip-based asynchronous and robust aggregation protocol - a pessimistic approach," in *Proceedings of IEEE Consumer Communications and Networking Conference (CCNC)*, jan. 2011, pp. 543 –548.

[22] D. J. Watts and S. Strogatz, "Collective dynamics of 'small-world' networks." *Nature*, vol. 393, no. 6684, pp. 440–442, Jun. 1998.

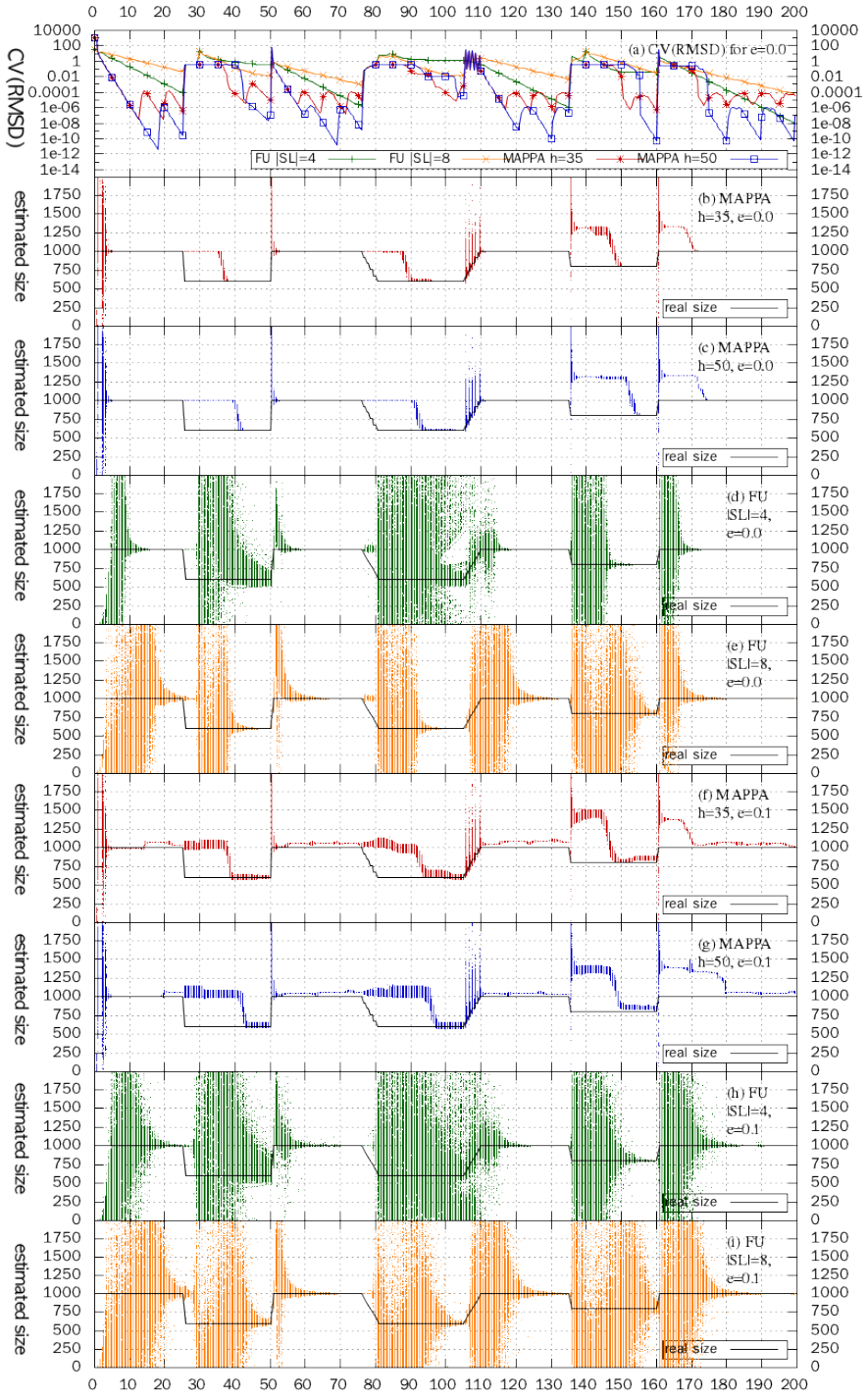
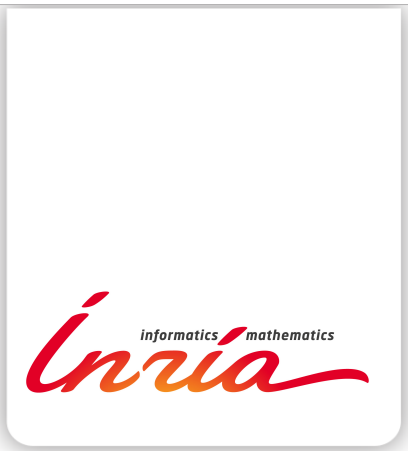


Figure 11: [Exp. 6: c=400 ms, m=4, tc=5 sec] MAPPA and Flow-Updating in dynamic networks



RESEARCH CENTRE
NANCY – GRAND EST

615 rue du Jardin Botanique
CS20101
54603 Villers-lès-Nancy Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399