University of Reading

Robust epidemic aggregation under churn

Conference or Workshop Item

Accepted Version

Ayiad, M. M. and Di Fatta, G. (2018) Robust epidemic aggregation under churn. In: The 10th International Conference on Internet and Distributed Computing System 2017 (IDCS 2017), 11 -13 December 2017, Mana Island Resort & Spa, Fiji, pp. 173-185. Available at http://centaur.reading.ac.uk/73501/

It is advisable to refer to the publisher's version if you intend to cite from the work. See Guidance on citing.

Published version at: https://doi.org/10.1007/978-3-319-97795-9_16

All outputs in CentAUR are protected by Intellectual Property Rights law, including copyright law. Copyright and IPR is retained by the creators or other copyright holders. Terms and conditions for use of this material are defined in the End User Agreement.

www.reading.ac.uk/centaur

CentAUR

Central Archive at the University of Reading

Reading's research outputs online

The 10th International Conference on Internet and Distributed Computing System 2017 (IDCS 2017), Mana Island, Fiji, Dec. 11-13, 2017.(in press)

Robust Epidemic Aggregation under Churn

Mosab M. Ayiad Giuseppe Di Fatta
Department of Computer Science, University of Reading
Whiteknights, Reading, Berkshire, RG6 6AY, UK *

November 1, 2017

Abstract

In large-scale distributed systems data aggregation is a fundamental task that provides a global synopsis over a distributed set of data values. Epidemic protocols are based on a randomised communication paradigm inspired by biological systems and have been proposed to provide decentralised, scalable and fault-tolerant solutions to the data aggregation problem. However, in epidemic aggregation, nodes failure and churn have a detrimental effect on the accuracy of the local estimates of the global aggregation target. In this paper, a novel approach, the Robust Epidemic Aggregation Protocol (*REAP*), is proposed to provide robustness in the presence of churn by detecting three distinct phases in the aggregation process. An analysis of the impact of each phase over the estimation accuracy is provided. In particular, a novel mechanism is introduced to improve the phase that is most critical for the protocol accuracy. *REAP* is validated by means of simulations and is shown to achieve convergence with a good level of accuracy for a reasonable range of node churn rates. **Keywords:** Distributed aggregation, Epidemic protocols, Gossip-based protocols, Mass recovery, Node churn

1 Introduction

In large-scale distributed systems, data aggregation is a fundamental task that is used to provide wide range of services, from the estimation of global system properties, such as resource capacity, average load and average uptime [1] to more complex network applications, such as failure detection [2], distributed data mining [3] and global attribute computation in Wireless Sensor Networks (*WSN*) [4]. Network protocols for aggregation typically compute a synopsis function such as (*sum*, *average*, *sample*, etc.) over a set of distributed data values with the purpose of providing global information about a network or a system. In large networks, centralised aggregation approaches limit the system scalability and can be subject to single points of failure and to bottlenecks [4]. Therefore, decentralised and fault-tolerance paradigms need to be utilised for an effective data aggregation solution. For instance, epidemic aggregation protocols are considered applicable and resilient to faults in large-scale distributed systems [5].

Epidemic (a.k.a. Gossip-based) protocols enable scalable, fault-tolerant, and fully decentralised solutions for various distributed problems, such as information dissemination and data aggregation [6]. In epidemic data aggregation, computation and communication are uniformly distributed among nodes using a randomised communication strategy. The random pairwise communication approach provides stochastic guarantees that nodes in the system ultimately converge to a common state in logarithmic time w.r.t. the system size [1]. The convergence to a target value is expected under static network conditions. In realistic conditions e.g. existence of node churn, the accuracy of local estimation value cannot be guaranteed and the result at convergence may significantly differ from the correct target value. This limitation is a consequence of the violation of the mass conservation invariant in a distributed system [7]. The mass conservation invariant refers to the ideal aggregation of the initial states of all nodes in the system. During the aggregation process, nodes exchange their states and distribute the mass among nodes. The convergence is achieved when the mass is distributed equally over all nodes. The failure and unreported departure of nodes determine a loss of the mass stored in these nodes, hence violating the conservation invariant and may lead to an estimation error of the target result that depends on the local state at the nodes departing the system.

This paper investigates the epidemic data aggregation process and identifies three distinct phases. It is shown that node churn at each phase have different effect on the estimation error. In particular, one of the three phases is critical for producing accurate results and is further investigated, resulting in a novel mechanism to address the mass conservation invariant problem. In this paper, the Robust Epidemic Aggregation Protocol (*REAP*) is proposed: *REAP* adopts a distributed failure detection and mass restoration mechanism. At each

*email: {m.m.ayiad@pgr., g.difatta@}reading.ac.uk node, *REAP* uses stochastic and heuristic methods to detect phases during the convergence process. The protocol implements a decentralised (Push-RePush) mechanism to detect failed and departed nodes and uses the timeout technique to trigger the mass restoration procedure. *REAP* is evaluated and validated by means of simulations. The experimental results show the ability of the protocol to produce a good accuracy with a moderate rate of churn. *REAP* achieves estimation error smaller than 1% in average at a churn rate less than or equal to 10%.

The term *churn* in this paper refers to nodes failure and unreported departure. Nodes fail and stop and do not restart. On another hand, an underlying reliable communication is assumed e.g. usage of *TCP* protocol, and hence communication failure and message loss are not examined. The churn of joining nodes is conveyed to future work due to limited space in the paper. A general approach to capture joining nodes is to implement a restart mechanism that repeats the aggregation operation with a fresh setting and encloses states of new nodes [1].

The paper is organised as follows. In Section 2, the analysis of the epidemic data aggregation process is explained. *REAP* is described in Section 3. Simulations and experimental results are discussed in Section 4. Some related work are listed in Section 5. Future work and conclusions are drawn in Section 6.

2 Epidemic Data Aggregation Process

Epidemic data aggregation is a global operation comprised of iterative distributed processes. Processes communicate in a randomised fashion and exchange their local information. Each process performs a designated algorithm (a.k.a *protocol*) on the local information to compute some global property or parameter. The output of the epidemic aggregation protocol at each node is a local estimation of the global function. Local estimations converge exponentially fast towards a target value. The distributed convergence corresponds to a reduction in the variance of the local approximation results. Eventually, all processes will converge to a target value with some tolerated error [1, 6, 7].

For the sake of this work, the Symmetric-Push-Sum Protocol (*SPSP*) [7] is used for the analysis of the aggregation process. *SPSP* is a proactive, decentralised and asynchronous aggregation algorithm that combines accuracy and stability of *Push-Sum* algorithms and efficiency and rapidity of *Push-Pull* algorithms. Each instance of *SPSP* periodically contacts a randomly chosen peer. For this purpose a simple Node Cache Protocol (*NCP*) [7] is used for sampling peers with an approximation of the uniform probability distribution and a k-regular random graph initialisation.

Let us consider a system of N nodes, each node i locally holds a numeric value x_i that represents a parameter or a local property in the node context. The aggregation is performed over a global set of local values, $f(x_i)$ $(0 < i \leq N)$. In *SPSP*, each node i maintains an aggregation pair $\langle v_i, w_i \rangle$, where v_i is the aggregation value and w_i is the aggregation weight. The aggregation value is initialised with the local value, $v_i = x_i$, and the initialisation of the weight is determined by the required aggregation function [7]. At each time step t (*cycle*), the node i halves the values of the local pair $\langle \frac{v_{i,t}}{2}, \frac{w_{i,t}}{2} \rangle$ and sends it (Push) to a random peer. When node i receives a Push message from node j, node i halves its local value and weight, sends the pair to j (Pull) and updates the local pair $\langle v_i + v_j, w_i + w_j \rangle$. The local aggregation estimate $e_{i,t}$ at the time t is computed by $e_{i,t} = \frac{v_{i,t}}{w_{i,t}}$. After sufficient number of cycles, each aggregation pair $\langle v_i, w_i \rangle$ is evenly distributed to all nodes in the system, and thus the variation among local estimates quickly decreases and estimates converge to the target value.

Without loss of generality, in the remainder of this work the problem of estimating the system size, corresponding to the aggregation function *count*, is chosen to study the aggregation process. In distributed systems, the correct estimation of the size is an essential task since it can be used for many other purposes e.g. assessing resource availability [8], parameter setting and network monitoring [4]. The protocol in [9] requires size estimation as an input parameter to achieve consensus.

In *SPSP*, the aggregation function *count* requires a particular initial setting of the aggregation pair $\langle v_i, w_i \rangle$: the value is set to a unit count, i.e. $v_{i,t_0} = 1$ $(0 < i \leq N)$, and the weight is set to $w_{\hat{\imath},t_0} = 1$ at a single *seed* node $\hat{\imath}$ whilst $w_{i,t_0} = 0$ $(0 < i \leq N, i \neq \hat{\imath})$, where t_0 is the system starting time and the seed node $\hat{\imath}$ is an arbitrary node, which can be selected randomly or with a leader election method. During the aggregation process, the pairs $\langle v_i, w_i \rangle$ are periodically exchanged and mixed, distributing and equalising the values over all nodes. With a correct convergence and after some time t_c $(t_c > t_0)$ the aggregation pair at each node i converges with a high probability [6] to $\langle 1, \frac{1}{N} \rangle$ and the local estimate e_i converges to a global target value n with a relative estimation error ϵ. On another hand and in a static network with ideal conditions, the total mass $\mathcal{M}$ is time invariant and is formulated as follows:

$$\mathcal{M}_v(\{v_i\}) = \sum_{i=1}^{N} v_i = N, \qquad \mathcal{M}_w(\{w_i\}) = \sum_{i=1}^{N} w_i = 1, \tag{1}$$

and the local estimate e_i in the epidemic aggregation process converges as follows:

$$e_i \to n = \frac{\frac{\mathcal{M}_v}{N}}{\frac{\mathcal{M}_w}{N}} = \frac{\sum_{i=1}^{N} v_i}{\sum_{i=1}^{N} w_i} = N. \tag{2}$$

As shown in the formula (2), $n = N$ when the mass conservation invariant holds and the local estimate e_i converges to the correct value N for $t \geq t_c$, also the estimation error ϵ_i converges to a very small value. However, the estimation error can be tolerated to some extent [6]. For some applications, a quick estimation might be more appropriate than a more accurate one which would take longer to compute. In the rest of the paper the symbol ε refers to the error threshold that is tolerated and determined by the application and the symbol ϵ represents the relative estimation error in local estimates during the aggregation process. The estimation error ϵ_i at each node i at any time $t \geq t_0$ is defined as:

$$\epsilon_i(t) = \frac{|e_i(t) - N|}{N}. \tag{3}$$

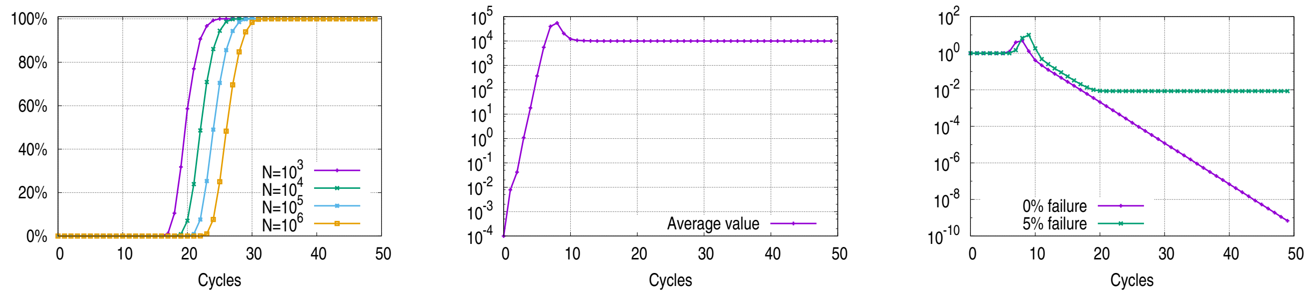
A typical aggregation process in *SPSP* is shown in Figure 1. The percentage of nodes converged to the true system size is illustrated in Figure 1.(a) for various system sizes. Figure 1.(b) shows the average of local estimates in the system converging to the correct value N and Figure 1.(c) illustrates the average estimation error in ideal and dynamic network conditions.

In dynamic networks, nodes can suddenly fail or leave the system, and hence there are two issues that need to be addressed. First, the target value (i.e. system size) is a time variant function $n(t)$. Although the protocol could be repeated periodically to follow the changes in the system size, a single global target value must be defined in each aggregation operation in order to achieve convergence. Secondly, the mass conservation invariant does not hold and node churn affect the accuracy of the estimates and may even prevent the convergence. In the next section, we show that partitioning the aggregation process at each node into phases helps to address these issues.

2.1 Phases of the Aggregation Process

Size estimation (global function *count*) is achieved with an aggregation process starting at t_0 and initiated at the seed node $\hat{\imath}$, which holds the weight $w_{\hat{\imath},t_0} = 1$. The seed node initiates an exponential diffusion process of the non-null weight. The computation at a generic node with the weight $w_{i,t_0} = 0$ are ineffective to the aggregation process until the node joins the diffusion propagation of the non-null weight ($w_i > 0$) either by a Push or a Pull message. The convergence is achieved at time t_c when each node i holds a similar fraction of the initial weight $w_{i,t_c} = \frac{w_{\hat{\imath},t_0}}{N} = \frac{1}{N}$. At a time $t \geq t_c$, although the exchange of the aggregation pair among nodes and the computation in each node continue, the local estimate at every node holds to same approximation result. This is the consequence of the equal distribution of the system mass among nodes. In general, during the aggregation process a node i can be in one of the following three phases:

1. Initial, if $w_i = 0$,
2. Propagation, if $w_i > 0$ and local convergence is not yet achieved, and
3. Convergence, if local convergence is detected.



(a) Percentage of nodes converged to correct size at various sizes, $\varepsilon \leq 1\%$.

(b) Average of local estimates, $N = 10^4$.

(c) Average of local estimation error, $N = 10^4$.

Figure 1: Size estimation in *SPSP*, $k = 30$

To analyse the impact of node churn on the accuracy of local estimates during each phase, let us consider that node i is in the INITIAL phase. At each time $t > t_0$, the local estimate e_i is not available due to the null weight value. Message exchanges with other nodes in the same phase do not contribute nor modify the estimate values, either local or remote. A message exchange with a node in the PROPAGATION phase determines a local phase transition. Although, the failure of a node i in the INITIAL phase does alter the system mass $\mathcal{M}_v$, it has no impact over the convergence process and over the accuracy of the approximation results because the more critical system mass $\mathcal{M}_w$ is preserved. As a consequence, the unexpected departure of a node i in the INITIAL phase only effects the target value $n(t)$ from N to $N-1$, as if the node i was never part of the system. There is no need to address churn in the INITIAL phase, apart from redefining the target value: the target value we then refer to is the number of nodes which have entered the PROPAGATION phase $N_p(t) = N - f_{init}(t)$ where f_{init} is number of nodes in the INITIAL phase which have departed the system.

Likewise, churn in the CONVERGENCE phase have no impact on the accuracy of the estimations w.r.t. the redefined target value. This is due to the equal distribution of the initial mass $\mathcal{M}_v$ and $\mathcal{M}_w$ among existing nodes. The exchange of aggregation pair and the computation of the local estimate at each node in the CONVERGENCE phase always gives the approximation result which the system has converged to. This can be proven as follows, where for simplicity we assume ideal network conditions and that all nodes have entered the PROPAGATION phase ($N_p = N$). Considering the formula (2), the target value is $N = \frac{\sum_{i=1}^{N} v_{i,t_0}}{\sum_{i=1}^{N} w_{i,t_0}}$, and hence,

$$\sum_{i=1}^{N} v_{i,t_0} = N \sum_{i=1}^{N} w_{i,t_0}, \tag{4}$$

and the local estimate of each node i in the CONVERGENCE phase is a good approximation of the target value: $e_{i,t} = N \implies N = \frac{v_{i,t}}{w_{i,t}}$, hence the following relation holds:

$$v_{i,t} = N w_{i,t}. \tag{5}$$

Assume a node j in the CONVERGENCE phase has failed, the system mass is deprived of the pair $\langle v_j, w_j \rangle$. In this case, the target can be calculated as $n(t) = \frac{\sum_i v_{i,t_0} - v_{j,t}}{\sum_i w_{i,t_0} - w_{j,t}}$, and by using the formulas (4) and (5) we have:

$$n(t) = \frac{\sum_{i=1}^{N} v_{i,t_0} - v_{j,t}}{\sum_{i=1}^{N} w_{i,t_0} - w_{j,t}} = \frac{N \sum_{i=1}^{N} w_{i,t_0} - N w_{j,t}}{\sum_{i=1}^{N} w_{i,t_0} - w_{j,t}} = N. \tag{6}$$

This analysis shows that if a node departs the system after it has converged, the global target of the aggregation process does not change and the aggregation process remains stable. Any node still in the system maintains an asymptotic convergence to the same target value, which corresponds to the number of nodes which have entered the PROPAGATION phase (N_p). As a consequence, there is no need to address churn in the CONVERGENCE phase.

It is now clear that the PROPAGATION phase is the critical phase w.r.t. the convergence and the accuracy of the aggregation process. Unexpected departure of nodes in the PROPAGATION phase need to be addressed explicitly. Any node i in the PROPAGATION phase has weight $w_i > 0$ that is a portion of the total weight mass $\mathcal{M}_w$. Also, $w_i \neq \frac{\mathcal{M}_w}{N_p}$ as node i has not yet reached convergence. Nodes in this phase hold critical information needed for a correct convergence to the target value. At t_0, the seed node $\hat{\imath}$ is the first node in the PROPAGATION phase and the most critical node. For $t > t_0$ other nodes transit from the INITIAL phase to the PROPAGATION phase in the exponential diffusion process. Although the probability of losing weight mass is very small at t_0, it exponentially increases with the diffusion process. At the same time the portion of the weight mass held at nodes in the PROPAGATION phase decreases over time. Thus, the impact of losing some portions of the weight mass on the approximation results also decreases.

The loss of mass due to node churn causes an estimation error in the approximation results during the aggregation process as shown in the analysis. In order to provide robustness to the accuracy of the convergence in the presence of churn, the local state $\langle v, w \rangle$ needs to be protected while the node is in the PROPAGATION phase. One way to achieve that is by means of a failure detection to enable mass restoration from its local state. The next section describes the protocol *REAP* that adopts a specific mechanism to achieve this goal.

3 The Robust Epidemic Aggregation Protocol (REAP)

REAP adopts non-blocking and asynchronous communication scheme. The protocol implements a symmetric PUSH-PULL strategy similar to *SPSP* [7] and utilises a (PUSH-REPUSH) model that is exclusively used for the churn detection during the PROPAGATION phase. The protocol detects PUSH messages from nodes in the PROPAGATION phase and maintains temporary replicas of the aggregation pair $\langle v, w \rangle$ in these messages. *REAP* uses the temporary replicas in the mass restoration procedure that is triggered when a given timeout value is expired and a failed or departed node is detected. The protocol is illustrated in Algorithm 1

Algorithm 1: Robust Epidemic Aggregation Protocol

```
Require: $\mathcal{T}$, maximum timeout value; $\varepsilon$, error threshold; $\Upsilon$, cycles threshold;
Initialisation: each node $i$ has:
    $v = 1$, $w = 0$, except one seed node has $w = 1$; critical PUSH flag $\rho = false$; convergence flag
    $\hat{\rho} = false$;
    a buffer for PUSH entries $\mathcal{P} = \{\mathbf{a}_p = \langle \beta, t, v, w \rangle, ...\}$, where $\beta$ is peer's id, $t$ is current time, and v,w
    are aggregation pair;
    a recovery cache $\mathcal{R} = \{\mathbf{a}_r = \langle \alpha, \tau, v, w \rangle, ...\}$, where $\alpha$ is sender id, $\tau$ is REPUSH receiving timeout, and
    v,w are replica pair;
    and has a fixed length queue $\mathcal{Q} = \{\}$;

At each cycle $c$ at node $i$:
$\rho \longleftarrow w > 0$ and $\neg\hat{\rho}$                                   // Detect PROPAGATION phase
$j \longleftarrow getRandomPeer()$                                          // Perform PUSH
$v = \frac{v}{2}$; $w = \frac{w}{2}$
$send(j, v, w, \rho, reply = true)$
if $\rho$ then $\mathcal{P} \longleftarrow \mathcal{P} \cup \{j, c, v, w\}$                     // Insert critical PUSH in $\mathcal{P}$
foreach $\mathbf{a}_p \in \mathcal{P}$ do                                            // Perform REPUSH
    if $\mathbf{a}_p.t < c$ then
        $send(\mathbf{a}_p.\beta, \mathbf{a}_p.v, \mathbf{a}_p.w, \rho = true, reply = true)$
        $\mathcal{P} \longleftarrow \mathcal{P} - \{\mathbf{a}_p\}$
foreach $\mathbf{a}_r \in \mathcal{R}$ do                   // Perform churn detection and mass restoration
    $\mathbf{a}_r.\tau \longleftarrow \mathbf{a}_r.\tau - 1$
    if $\mathbf{a}_r.\tau == 0$ then                              // Timeout value expired, restore mass
        $v = v + \mathbf{a}_r.v$; $w = w + \mathbf{a}_r.w$
        $\mathcal{R} \longleftarrow \mathcal{R} - \{\mathbf{a}_r\}$

At event 'receive message $m$ from $j$' at node $i$:
if $m.reply$ then                                                 // Distinguish a PUSH
    if $\exists \mathbf{a}_r \in \mathcal{R}$ where $\mathbf{a}_r.\alpha == j$ then                     // Distinguish a REPUSH
        $\mathcal{R} \longleftarrow \mathcal{R} - \{\mathbf{a}_r\}$
        go to line 18
    else
        $v = \frac{v}{2}$; $w = \frac{w}{2}$                                             // Perfrom PULL
        $send(j, v, w, \rho = false, reply = false)$
$\mathcal{Q}.enqueue(\frac{v}{w}, \frac{m.v}{m.w})$                                         // Enqueue estimates
$v = v + m.v$; $w = w + m.w$                                     // Update aggregation pair
if $m.\rho$ then $\mathcal{R} \longleftarrow \mathcal{R} \cup \{\mathbf{a}_r = \langle j, \mathcal{T}, v, w \rangle\}$                 // Replicate a critical pair
$\hat{\rho} \longleftarrow \widehat{C_v}(\mathcal{Q}) \leq \varepsilon$ for $\Upsilon$ cycles                               // Detect convergence
```

The detection of the PROPAGATION phase in *REAP* is performed as follows. A node i enters the PROPAGATION phase when receives a non-null weight ($w > 0$) and exits the phase when node i detects the convergence locally. The local detection of convergence is accomplished using a heuristic method [10] based on the computation of the moving average. Each node i in *REAP* maintains a fixed length queue $\mathcal{Q}$ to store the local estimate e_i and received estimate e_j from node j. Then after, node i computes the *Coefficient of Variance*, $\widehat{C_v} = \frac{s}{\bar{x}}$ for current elements in $\mathcal{Q}$ where s is the standard deviation and $\bar{x}$ is the arithmetic mean. The convergence is detected at node i when $\widehat{C_v}$ falls below a given error threshold ε for a number of consecutive cycles Υ. The error threshold ε controls the desired level of approximation accuracy and Υ is used to prevent precociously convergence detection during the convergence process.

In the PROPAGATION phase, each node i contacts a random node j twice: one time to perform a PUSH and a second time to perform a REPUSH. The two contact operations occur at two consecutive cycles, e.g. a node i sends the PUSH message to node j at time t and sends the REPUSH message at time $t + 1$. The PUSH and REPUSH messages carry the same information and hence *REAP* distinguishes the type of message by the order of reception. This manner enables the protocol to cope with asynchronous communication in which messages encounter different delays and a REPUSH message may be delivered before the corresponding PUSH message. Typically, the REPUSH message is used by the receiver node to detects the liveness of the sender node. A node i detects the failure or departure of node j when the REPUSH message from node j is not received within a predefined period $\mathcal{T}$ where $\mathcal{T}$ is the maximum timeout value.

REAP maintains a buffer of PUSH entries $\mathcal{P}$ to manage the PUSH-REPUSH operations. Each cycle the protocol inserts an entry $\mathbf{a}_p$ for the fresh PUSH information in $\mathcal{P}$. Each entry $\mathbf{a}_p$ holds the random peer's Id β, the contact time t, and the local values of the aggregation pair ($\mathbf{a}_p = \langle \beta, t, v, w \rangle$). In the subsequent cycle, the protocol performs a REPUSH operation using the information in the previous PUSH entry $\mathbf{a}_p$. After the REPUSH operation, the entry $\mathbf{a}_p$ is removed from $\mathcal{P}$ keeping the buffer size minimum.

Nodes in the PROPAGATION phase assign a flag to each PUSH message indicating that the message has a critical information for the aggregation process. On the reception of a critical PUSH message, the receiver node i updates local aggregation pair and then replicates the local pair and stores the replica in a local cache $\mathcal{R}$ using the entry ($\mathbf{a}_r = \langle \alpha, \tau, v, w \rangle$) where α is sender's Id, τ is REPUSH receiving timeout and v, w are the replica's pair. In the subsequent cycle, node i receives a REPUSH message and deletes the corresponding entry in $\mathcal{R}$. In case the REPUSH message is not received within $\mathcal{T}$ cycles, the protocol assumes that the corresponding node has failed and the replicated pair in the entry $\mathbf{a}_r$ is used in the mass restoration process. The restored mass corrects the local aggregation pair values. The correction amount is then distributed among nodes through regular contact operations. The distribution of the restored mass produces a very small delay in the convergence speed but leads the convergence process to a lower estimation error of the target value.

As shown in Algorithm 1, at each cycle, the protocol detects the PROPAGATION phase and sets the PUSH flag in line 4. In lines 5-7 the protocol performs a PUSH operation. The protocol inserts fresh PUSH entry in $\mathcal{P}$ in line 8 and performs the REPUSH operation in lines 9-12. The detection of churn and the mass restoration procedure are performed in lines 13-17. The protocol distinguishes the PUSH and REPUSH messages in line 19 and line 20. The PULL message is formulated in lines 24 and 25. The protocol inserts local and received estimates in the queue $\mathcal{Q}$ in line 26. A critical aggregation pair is replicated and an entry $\mathbf{a}_r$ is stored in $\mathcal{R}$ cache in line 28. The local convergence is detected in line 29.

4 Simulations and Experimental Results

The protocol *REAP* is simulated using PEERSIM [11], a Java-based discrete-event P2P simulation tool. PEERSIM is flexible, configurable and scalable. The simulation model is fully Event-based and uses the event-driven engine in PEERSIM. Two events are used in the simulation: (*i*) The RUN EVENT is scheduled to trigger at every cycle and stops after a predefined number of cycles. At this event, a node performs contact operations, detects churn and performs the mass restoration. (*ii*) The MESSAGE RECEIVE EVENT occurs when a node receives a message. At this event, the incoming message is processed, the local aggregation pair is updated and the detection of convergence is performed.

Two protocols are examined in the simulations: *SPSP* and *REAP*. The performance and the accuracy of the protocols are monitored and recorded at each cycle using a dedicated external observer. The communication latency is adjusted for all messages to be delivered in the same global cycle. This adjustment is necessary to avoid transition of portions of the total mass $\mathcal{M}$ across cycles during the simulations. The adjustment is applied only for the sake of the simulations and can be relaxed in a realistic context.In each simulation run, a different random seed is applied to enforce randomisation and each experiment is repeated for tens of times to validate the setting and the results. The protocols are initialised by a peak data distribution whereas $v_i = 1$, $0 < i \leq N$; $w_0 = 1$ at seed node 0 and $w_i = 0$, $1 < i \leq N$. Protocols' parameters are carefully adjusted to certain values to meet practical applications requirements [9]: the timeout value is set to $\mathcal{T} = 3$ cycles, the tolerated error threshold is set to $\varepsilon = 1\%$, the minimum number of consecutive cycles is set to $\Upsilon = 5$ cycles, the length of the $\mathcal{Q}$ is set to 10 elements. On another hand, the protocol *NCP* is used as a peer sampling service and is configured to maintain a random k-regular overlay with $k = 30$.

The performance of *REAP* is examined in a static network conditions. The Figure 2.(a) shows the smooth transition of nodes among phases and the correct computation of the entry and exit points of the PROPAGATION phase. In Figure 2.(b) the local estimate value of a randomly chosen node is illustrated for *SPSP* and *REAP* showing the performance similarity when no churn is encountered. The Figure 2.(c) shows the communication overhead of both protocols whereas *REAP* produces only one extra message for a temporary period during the PROPAGATION phase.

To evaluate the protocol *REAP*, we examined the protocol against *SPSP* at different churn rates: $\{$0%, 1%, 5%, 10%, 20%$\}$. A number of random nodes are selected at each cycle from cycles $[0, 30[$ and removed from the system by an external actor during the aggregation process. In each case, the number of failing nodes is increased and distributed over the 30 cycles to ensure random selection of nodes from each phase of the aggregation process. The simulation experiment of each case is repeated for ten times and aggregation pair in each alive node is observed. During the simulations, the local estimation error is computed at each cycle by the external observer as $\epsilon_i = \frac{|e_i - N_p|}{N_p}$. The experiments results are averaged and overall results are illustrated in Figure 3 and Figure 4.

The Figure 3 shows the total mass $\mathcal{M}_v$ and $\mathcal{M}_w$ during the aggregation process. Trends of lines in the figure show the correction in mass magnitude as a result of the mass restoration mechanism in *REAP*. The figure also shows that amount of correction in the comparison to *SPSP*. The improvement of accuracy in *REAP* is shown in the Figure 4. The figure illustrates the trend of estimation error in each case of the simulated churn. In the first case no churn is encountered and estimation error tends to a very small value indicating the correct convergence to the target value. In the rates less than 10%, the estimation error rises to 1%, whilst the error exceeds 1% for higher rates. Generally, estimation error in *REAP* is lower than the error in *SPSP*. In the presence of churn, local estimates in *SPSP* converge to approximation result different from the target value as shown in Figures 4 and 3.

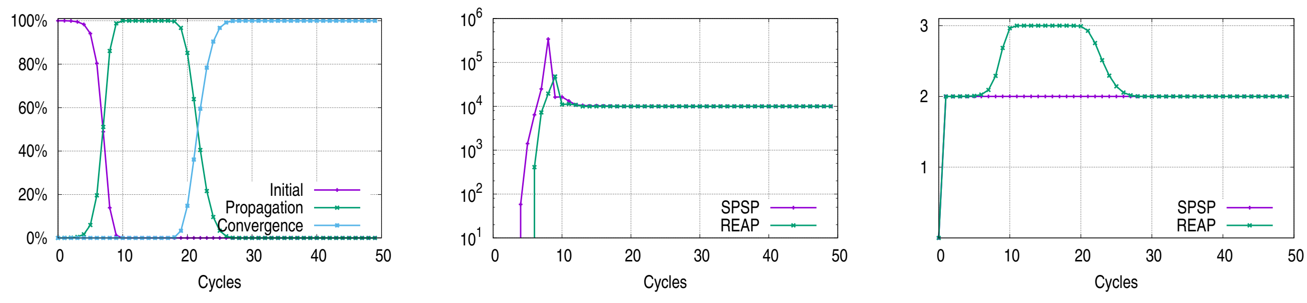
The deficiency in the accuracy in *REAP* at higher rates of churn ($\geq 20\%$) is due to the increase in the *cascaded failure* event. Let us consider two nodes i and j, node i sends a Push message to node j and then fails. The node j requires $\mathcal{T}$ cycles before detecting the failure of node i. During this period node j fails too. In this case, the system mass deprives of aggregation pairs in node j and node i. Despite this limitation which can be a potential future research, the protocol *REAP* has enabled robust epidemic data aggregation for applications which can tolerate 1% of estimation error.

5 Related work

The work in [10] proposed a number of heuristic methods to locally detect convergence in epidemic protocols, e.g. the calculation of standard deviation and root mean squared error over a number of buffered estimates are used as a criterion in the detection formulas. The work in [12] proposes a pessimistic approach and introduces a recovery mechanism which conserves the system mass in the presence of asynchrony and churn. This approach applies asynchrony to the work of [1] enabling loosely synchronised rounds. However, the round length is set long enough for messages to be delivered within the same round, i.e. out-of-round messages are ignored. The protocol computes some recovery shares in order to resolve the interference caused by asynchrony or churn. Authors in [13] studied the *Push-Pull* protocols from a practical point of view. The work proposes a gossip-based multi-pair asynchronous *Push-Pull* protocol and introduces the pair mass conservation as a mechanism to protect the mass conservation invariant. A gossip-based aggregation technique is proposed and evaluated in [14] namely *Flow Updating*. It has shown that *Flow Updating* can operate on faulty dynamic networks and that the technique is adaptable to churn, input value changes, and message loss without requiring periodic restarting. However, the technique is based on symmetric values exchange and hence a strong correlation among neighbour nodes is a core requirement. Also, the correct detection of nodes failure takes a certain amount of time in addition to a noticeable delay that is needed for local estimates to re-converge to a correct value.

6 Conclusions

In epidemic data aggregation, nodes failure and churn negatively affect the accuracy of the approximation results. In this paper, the aggregation process is analysed and three different phases are identified. The occurrence of node churn during each phase has a different impact on the accuracy of the approximation results. The second phase, namely the Propagation phase, is identified as the most critical phase for the aggregation accuracy. The paper introduces a novel Robust Epidemic Aggregation Protocol (*REAP*) with an innovative Push-RePush communication technique to detect churn and restore the lost mass particularly for nodes in the Propagation phase. *REAP* is validated by means of simulations and the experimental results have shown the ability of the protocol to attain robustness of the accuracy in the presence of churn. The average estimation error in *REAP*

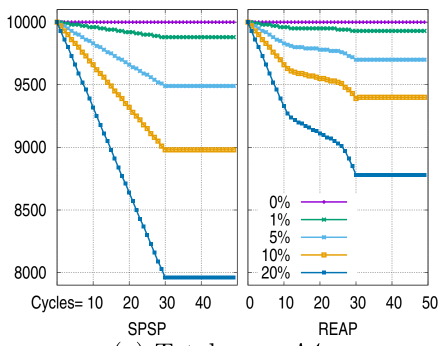


(a) Percentage of nodes in each phase.

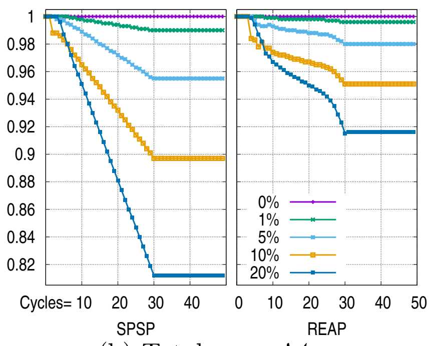
(b) Local estimate of a single node.

(c) Average number of sent messages.

Figure 2: Size estimation in *REAP*, $N = 10^4$, $\varepsilon = 1\%$, $\Upsilon = 5$, $k = 30$



(a) Total mass $\mathcal{M}_v$.



(b) Total mass $\mathcal{M}_w$.

Figure 3: Size estimation at various churn rates, $N = 10^4$, $\varepsilon = 1\%$, $\Upsilon = 5$, $k = 30$

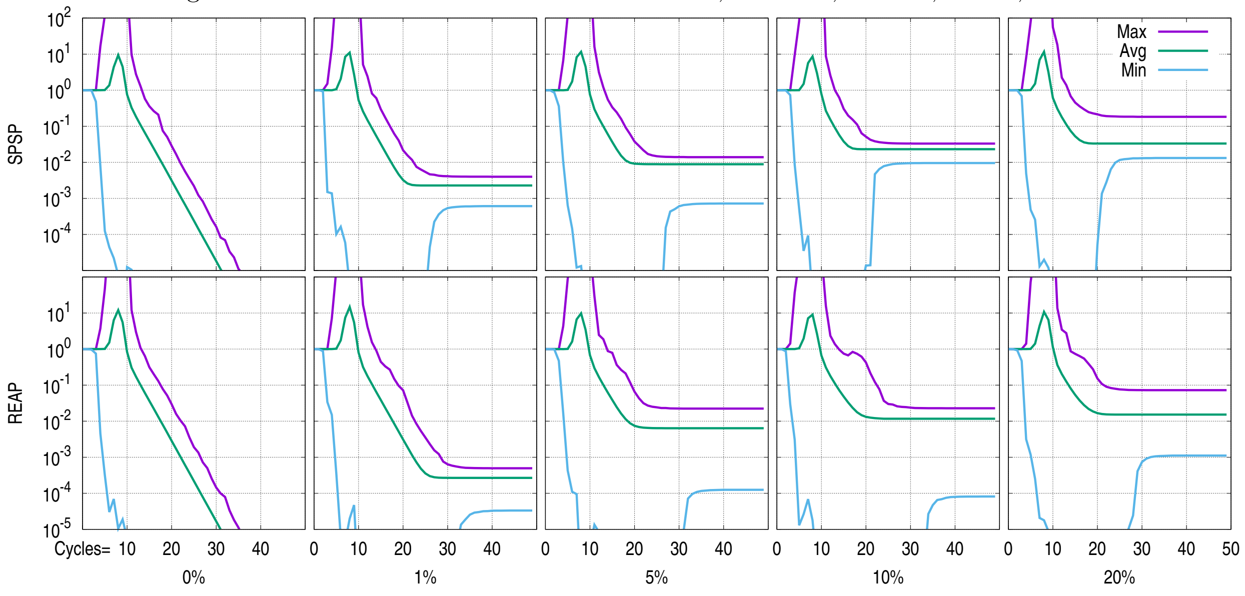


Figure 4: Estimations error (ϵ) at various churn rates, $N = 10^4$, $\varepsilon = 1\%$, $\Upsilon = 5$, $k = 30$

is less than 1% at churn rate up to 10%. This error falls within the tolerated threshold in epidemic applications. The future work includes studying the cascaded failure problem in *REAP* and may also address the aggregation process in a fully asynchronous system and lossy network scenarios.

7 Acknowledgements

The author Mosab M. Ayiad is supported for his PhD project by the Islamic Development Bank-Merit Scholarship Programme for High Technology (*MSP*), 2015-(600029531).

References

[1] M. Jelasity, A. Montresor, and O. Babaoglu. "Gossip-based Aggregation in Large Dynamic Networks". In: *ACM Transactions on Computer Systems* (2005).

[2] A. Katti, G. D. Fatta, T. Naughton, and C. Engelmann. "Epidemic failure detection and consensus for extreme parallelism". In: *The International Journal of High Performance Computing Applications* (2017).

[3] G. D. Fatta, F. Blasa, S. Cafiero, and G. Fortino. "Fault tolerant decentralised K-Means clustering for asynchronous large-scale networks". In: *Journal of Parallel and Distributed Computing* (2012).

[4] L. Chitnis, A. Dobra, and S. Ranka. "Aggregation Methods for Large-scale Sensor Networks". In: *ACM Transactions on Sensor Networks* (2008).

[5] R. Makhloufi, G. Bonnet, G. Doyen, and D. Gaïti. "Decentralized Aggregation Protocols in Peer-to-Peer Networks: A Survey". In: *Proceedings of IEEE International Workshop on Modelling Autonomic Communications Environments.* Springer, 2009.

[6] D. Kempe, A. Dobra, and J. Gehrke. "Gossip-based computation of aggregate information". In: *Foundations of Computer Science, Proceedings. 44th Annual IEEE Symposium on.* IEEE, 2003.

[7] F. Blasa, S. Cafiero, G. Fortino, and G. D. Fatta. “Symmetric Push-Sum Protocol for decentralised aggregation”. In: *Proceedings of International Conference on Advances in P2P Systems*. IARIA, 2011.

[8] D. Kostoulas, D. Psaltoulis, I. Gupta, K. Birman, and A. Demers. “Decentralized Schemes for Size Estimation in Large and Dynamic Groups”. In: *Fourth IEEE International Symposium on Network Computing and Applications*. IEEE, 2005.

[9] M. Ayiad, A. Katti, and G. Di Fatta. “Agreement in Epidemic Information Dissemination”. In: *Internet and Distributed Computing Systems: International Conference (IDCS), Proceedings*. Springer, 2016.

[10] P. Poonpakdee, N. Orhon, and G. Di Fatta. “Convergence Detection in Epidemic Aggregation”. In: *Parallel Processing Workshops*. Springer, 2014.

[11] A. Montresor and M. Jelasity. “PeerSim: A scalable P2P simulator”. In: *Peer-to-Peer Computing*. IEEE, 2009.

[12] I. Rao, A. Harwood, and S. Karunasekera. “Gossip-based asynchronous and robust aggregation protocol - A pessimistic approach”. In: *Consumer Communications and Networking Conference*. IEEE, 2011.

[13] H.-G. Roh and C. L. Ignat. *Rapid and Round-free Multi-pair Asynchronous Push-Pull Aggregation*. Tech. rep. INRIA, 2012.

[14] P. Jesus, C. Baquero, and P. S. Almeida. “Flow updating: Fault-tolerant aggregation for dynamic networks”. In: *Journal of Parallel and Distributed Computing* (2015).