Achieving robust average consensus over lossy wireless networks

Francesco Acciani, Paolo Frasca, Geert Heijenk, Anton Stoorvogel

Achieving robust average consensus over lossy wireless networks

Francesco Acciani, Paolo Frasca, Geert Heijenk and Anton Stoorvogel

***Abstract*—Average consensus over unreliable wireless networks can be impaired by losses. In this paper we study a novel method to compensate for the lost information, when packet collisions cause transmitter-based random failures. This compensation makes the network converge to the average of the initial states of the network, by modifying the weights of the links to accommodate for the topology changes due to packet losses. Additionally, a gain is used to increase the convergence speed, and an analysis of the stability of the network is performed, leading to a criterion to choose such gain to guarantee network stability. For the implementation of the compensation method, we propose a new distributed algorithm, which uses both synchronous and asynchronous mechanisms to achieve consensus and to deal with uncertainty in packet delivery. The theoretical results are then confirmed by simulations.**

I. Introduction

In the last decade, attention for multi agent control systems has grown, and the same has happened for distributed consensus algorithms. Consensus algorithms form the basis for a large number of distributed algorithms, like distributed hypothesis testing [1], distributed maximum likelihood estimation [2], and distributed Kalman filtering [3]. Distributed consensus algorithms have been studied under a wide variety of conditions, including networks with undirected or directed links, time varying topologies [4], and noisy channels [5], [6]. A significant example of an unreliable, time varying channel is the wireless medium, used for safety-critical applications such as vehicular networks (VANETs), which inspired the packet loss model used in this work. VANETs use the underlying IEEE 802.11p protocol in broadcast mode, where packets are sent from a node to every neighbour in its radio range. Here, packet loss is mostly caused by collisions, e.g. due to hidden terminals, affecting the reception of the message by all receivers. This motivates the broadcast approach and loss model used in this paper [7].

Most average consensus dynamics are not robust to packet losses [8]: if there are losses, consensus might not be reached, and if reached the agreed value might not be the average of the initial conditions. In many cases, an estimate of the induced distortion can be found following [9], [10]. Various strategies exist to deal with the information lost due to packet drops. Some methods are able to guarantee consensus, but not average consensus, e.g. by modifying the weights of the link between the nodes if a failure happens, like the biased compensation method proposed in [8]. Other methods preserve the average of the network, requiring retransmissions [11] or additional variables to be transmitted [12], [13]; or requiring the nodes to have a memory mechanism [14]; some of them work under restrictive failure models [15].

The aim of this paper is to devise a *compensation method* to preserve the convergence to the average in presence of packet losses. More specifically, we want each node to modify the weight of each link dynamically when a packet is not received from a neighbour: if the information from one neighbour is lost, the node will weigh differently the other links in its neighbourhood, so that the average of the network is not compromised. This weight-modifying mechanism is inspired by the one in [8], but is improved to converge to the average, and not to some other random value instead. Consequently, there are two levels of time-variance of the network: the network topology is varying in time, due to packet losses – i.e. a link might drop at any given time – and the numerical values of the weighted adjacency matrix are dynamically adjusted to preserve average consensus. After finding the update rule that achieves average consensus, we devise a distributed algorithm to implement the update rule. The key challenge of the distributed algorithm is to deal with the asynchronous nature of the packet exchange and with the uncertainty regarding delivery of packets to neighbours. The distributed algorithm provides the required network awareness to the nodes.

The contribution of this paper is twofold: (i) we propose a compensation mechanism to preserve the *average* in the consensus dynamics, where convergence speed can be improved by using a suitable gain, for which we provide bounds to secure stability of the network, and (ii) we describe and simulate a distributed algorithm to implement this compensation method, specifically to safely deliver the acknowledgementsand deal with the asynchronous nature of the communication.

The rest of the paper is organised as follows: Section II presents the model of the lossy consensus system; Section III describes the proposed compensation methods; Section IV proposes the distributed protocol and Section V presents the simulation results. In the final section we present some conclusions.

Preliminary and partial versions of some of our contribu-

F. Acciani is with Faculty of Electrical Engineering, Mathematics and Computer Science, University of Twente, 7500 AE Enschede, The Netherlands f.acciani@utwente.nl

P. Frasca is with Univ. Grenoble Alpes, CNRS, Inria, GIPSA-lab, F-38000 Grenoble, France paolo.frasca@gipsa-lab.grenoble-inp.fr

A. Stoorvogel is with Faculty of Electrical Engineering, Mathematics and Computer Science, University of Twente, 7500 AE Enschede, The Netherlands a.a.stoorvogel@utwente.nl,

G. Heijenk is with Faculty of Electrical Engineering, Mathematics and Computer Science, University of Twente, 7500 AE Enschede, The Netherlands geert.heijenk@utwente.nl

tions have appeared or are about to appear in the proceedings of IEEE conferences, namely the compensation method in [16] and the gain-acceleration in [17]. In this journal version, we not only present these results in revised and consistent form, with more details, but we also include a novel, further corrective step (named *asynchronous fallback*) which is necessary to preserve the average in the asynchronous implementation (Section IV.B and Section IV.C)

II. The consensus problem with packet loss

A network of n agents can be represented by a graph, where each *node* represents one agent, and each *edge* represents a link between two nodes. When the network is homogeneous, the hypothesis of bidirectional links arises naturally: this results in an undirected (symmetric) graph.

The consensus problem can be described as follows: each agent of the network can update itself, relying only on the messages received from its neighbourhood, and the consensus problem is solved if all the nodes in the network agree on the same value. Assume that each agent i is endowed with a scalar state x_i, and can communicate with a subset of agents, namely its neighbourhood $\mathcal{N}_i$. In the following, we indicate the set of all agents as $\mathcal{N}$. Each agent can update itself, computing a new value of its state, relying on the states of its neighbourhood, according to an *update rule* $g(\cdot)$:

$$x_i(k+1) = g_i\left(\{x_j(k) | j \in \mathcal{N}_i\}\right)$$

where k denotes an integer in a discrete-time framework. The neighbourhood $\mathcal{N}_i$ indicates all the nodes which state is accessible to i, so $i \in \mathcal{N}_i$.

As mentioned before, the consensus problem is solved if the network converges –asymptotically– to the same value ξ, so if:

$$\lim_{k\to\infty} x_i(k) = \xi \quad \forall i \in \mathcal{N}. \tag{1}$$

Clearly, the consensus problem is influenced by two factors: the neighbourhood of each node, and the update rule. When the network's topology is time invariant, i.e. the links are reliable and fixed, it is possible to choose the following update rule:

$$x_i(k+1) = \sum_{j=1}^{n} \overline{w}_{ij} x_j(k), \tag{2}$$

which can be interpreted as each node trying to drive its state to the (weighted) average of its neighbourhood. This update rule can be written in a more compact form, assuming $x(k) = [x_1(k)\, x_2(k) \dots x_n(k)]^T$, as

$$x(k+1) = \overline{W} x(k), \tag{3}$$

where $\overline{w}_{ij} \neq 0 \Longrightarrow i$ and j are connected. We assume that $\overline{W}$ has positive diagonal elements.

The $\overline{W}$ matrix is the *weighted adjacency* matrix of the network, associated with the graph representing the topology of the network, where the nodes are the vertices of the graph. If the graph associated with $\overline{W}$ is connected, i.e. a path (sequence of links) exists between each pair of nodes, then a sufficient condition for the network to converge in the sense of (1) is the row-stochasticity of $\overline{W}$. In the rest of the paper we assume that the graph associated with the network – in its *nominal* conditions, i.e. without failures – is connected , that the initial links' weights are chosen such that the $\overline{W}$ matrix is stochastic, i.e. $\overline{W}\mathbb{1} = \mathbb{1}$ where $\mathbb{1}$ denotes a vector of ones, and its diagonal elements are positive. Moreover, recalling the bidirectional nature of the wireless communication, the $\overline{W}$ matrix is assumed to be symmetric, and hence doubly stochastic.

The double stochasticity condition assures *average* consensus: the network will converge – in absence of packet losses – to the average of the initial conditions of the nodes. When the matrix $\overline{W}$ is only row-stochastic, the convergence of the network is still guaranteed, but the asymptotic consensus value is not the average of the initial conditions, but it is some *weighted* average of the nodes' initial conditions instead.

When the links are not reliable, due to failures in the communication between nodes, the dynamics changes to a time-dependent one:

$$x(k+1) = W(k)x(k) \tag{4}$$

because when the communication between two generic nodes i and j drops at some instant $\hat{k}$, then $w_{ij}(\hat{k}) = 0$. This implies that the matrix $W(\hat{k})$ is not stochastic anymore, and thus consensus can be compromised. There are two possible failure outcomes for the average consensus problem: when the topology changes randomly the condition (1) might not be verified, i.e. the network does not converge at all, or the network might converge, but to a value that is not the average of the initial conditions. However, it is possible to modify the matrix $W(k)$ when a communication failure arises: weights of links in the neighbourhood that did not experience a failure during the k^{th} iteration can be changed to compensate for the communication drop and to preserve the double-stochasticity nature of the matrix $W(k)$.

We consider the specific case when a packet *collision* causes a communication loss. This scenario arises when two nodes try to communicate at the same time, causing interference in the wireless medium, thus making the communication impossible in the neighbourhood where the collision happens. It should be noticed that communications in the IEEE 802.11p standard –which will be used for vehicle-to-vehicle communication– are used by different applications in the same channel, and even communication in other channels might lead to collisions [18], thus an approach based solely on scheduling or priorities is not sufficient to avoid communication losses. Moreover, we assume that communicating nodes are relatively close, compared to the transmission range of the communication, as it would be the case in vehicular network. As a result, these nodes will experience often good communication quality, and phenomena such as collisions will be mostly experienced by all the nodes involved in the consensus process in a neighbourhood. The nature of this phenomenon of loss of communication in a neighbourhood suggests a node-based failure model, instead of a link-based one: if a collision happens, the message broadcast from one node to all its neighbours is lost, as all the communication in the area is impossible. While the dynamics (4) assumes synchronous

communication, the actual implementation is asynchronous, as we explain in Section IV: the nodes communicate one after the other, so a sending failure does not affect the incoming messages – as it would be expected – because the messages are not simultaneous.

Motivated by this application, we assume that each node is in a failure state or not with a given probability: the communication loss is modelled by a *failure vector* $f(k)$ where $f_i(k) = 1$ if the communication from node i to its neighbours is successful during the k^{th} consensus iteration, 0 otherwise. To keep the analysis simple, the failure probability $\mathbb{P}[f_i(k) = 0] = p$ is assumed to be the same for each node of the network, and the failures are assumed to be independent of each other. While this hypothesis might seem restrictive, as a failure from a sending node in a synchronous scenario implies the loss of incoming communication from its neighbours, it should be noticed how this variable captures only the failures of outgoing packets, according to the asynchronous protocol implementation that we will suggest in Section IV. Alongside the spatial independence of failure events, the failures are also time independent, as a previous failure event does not influence any subsequent transmission. However, the probability independence does not influence the compensation method, but is used to analyse the dynamics of the system.

It is now possible to devise an update rule that preserves average consensus in the network, which is done in the next section.

III. The average preserving compensation methods

Using the model presented in the previous section, it is possible to modify the update rule followed by each node, to compensate for a link failure, due to packet collision. When a transmission fails, the receiving node can use its state to perform the update, instead of the neighbour's one which did not transmit successfully. Moreover, when a node learns that its transmission was not successful, it will not update itself. This compensation method leads to the following update rule:

$$\begin{aligned} x_i(k+1) =& f_i \left(\sum_{j=1}^{n} f_j \overline{w}_{ij} x_j(k) + \sum_{j=1}^{n} (1-f_j) \overline{w}_{ij} x_i(k) \right) \\ &+ (1-f_i) x_i(k) \end{aligned} \tag{5}$$

where $f_j(k)$ models the loss of communication that a neighbour is experiencing, it is equal to zero when the node j fails its communication, and $f_i(k)$ models the behaviour of the node, which will go in a standby state when its last communication was not successful.

After some manipulation, equation (5) reads:

$$x_i(k+1) = x_i(k) + \sum_{j \in \mathcal{N}_i} f_i(k) f_j(k) \overline{w}_{ij} \left(x_j(k) - x_i(k) \right),$$

and it is now possible to give more details about the consensus dynamics (4) in presence of losses, for which the matrix $W(k)$ is given by:

$$W(k) = I + F(k) \overline{W} F(k) - \operatorname{diag}(F(k) \overline{W} F(k) \mathbb{1}) \tag{6}$$

and $\operatorname{diag}(v)$ is the diagonal matrix whose entries are the elements of the vector v, while $F(k) = \operatorname{diag}(f(k))$.

The following criterion for consensus is a special case of the one proved in [8]. In order to state it, let us denote the expected value of a random variable X as $\mathbb{E}[X]$, and the directed graph associated with an adjacency matrix M as $\mathcal{G}_M$: we associate to the matrix M the digraph $\mathcal{G}_M$ with a set of vertices $\{1, \dots, n\}$ in which there is an edge from i to j whenever $M_{ij} \neq 0$.

Lemma III.1. *Assume that $A(k)$ is a sequence of i.i.d stochastic matrices, such that for all i we have $A(k)_{i,i} > 0$. If $\mathcal{G}_{\mathbb{E}[A(k)]}$ is connected then $A(k)$ achieves consensus almost surely.*

By this result, the problem of verifying the convergence to consensus property of the system is transformed into checking the structure of the expected value of the matrix $W(k)$.

Thanks to the previous lemma, the following result holds true:

Proposition III.2. *Assume $\overline{W}$ in (3) is a doubly stochastic weighted adjacency matrix, $W(k)_{i,i} > 0$, and $\mathcal{G}_{\overline{W}}$ is connected. The expectation of $W(k)$ is*

$$\mathbb{E}[W(k)] = (1-p)^2 \overline{W} + p(2-p) I$$

and $W(k)$ achieves consensus almost surely.

Proof. From (6) we have:

$$\begin{aligned} \mathbb{E}\left[W(k)\right] &= I + \mathbb{E}\left[F(k)\overline{W}F(k) - \operatorname{diag}(F(k)\overline{W}F(k)\mathbb{1})\right] \\ &= I + \mathbb{E}\left[F(k)\overline{W}F(k)\right] - \operatorname{diag}(\mathbb{E}\left[F(k)\overline{W}F(k)\mathbb{1}\right]) \end{aligned}$$

remembering that:

$$\left[F(k)\overline{W}F(k)\right]_{ij} = \begin{cases} f_i(k) f_j(k) \overline{W} ij & \text{if } i \neq j, \\ f_i(k) \overline{W} ij & \text{if } i = j. \end{cases}$$

Therefore, it is possible to evaluate the expected value:

$$\mathbb{E}\left[F(k)\overline{W}F(k)\right]_{ij} = \begin{cases} (1-p)^2 \overline{w}_{ij} & \text{if } i \neq j \\ (1-p)\overline{w}_{ij} & \text{if } i = j \end{cases}$$

The expectation of $W(k)$ then becomes:

$$\mathbb{E}\left[W(k)\right] = (1-p)^2 \overline{W} + p(2-p) I$$

Clearly $\mathcal{G}_{\mathbb{E}[W(k)]} = \mathcal{G}_{\overline{W}}$, which we assumed to be connected, and thus, by Lemma III.1, the update rule (6) achieves consensus almost surely. □

The state $x(k+1)$ can be expressed as:

$$x(k+1) = W(k)x(k) = \prod_{\xi=1}^{k} W(\xi) x(0) = \Delta(k) x(0),$$

and it can easily be checked that:

$$\mathbb{1}^T \Delta(k) = \mathbb{1}^T W(k) \Delta(k-1) = \mathbb{1}^T \Delta(k-1) = \mathbb{1}^T. \tag{7}$$

Similarly $\Delta(k)\mathbb{1} = \mathbb{1}$ and its elements are nonnegative since $W(k)$ has nonnegative elements, hence $\Delta(k)$ is doubly stochastic. Therefore the consensus value reached asymptotically by the network is the average of the nodes' initial conditions.

A. The α-average preserving compensation method (αAP)

The idea behind the average preserving compensation method is straightforward: when there is a topology change in the network due to packet loss, the nodes will perform a compensation, modifying the links' weights and resulting in a change in the *weighted* adjacency matrix, such that the convergence properties of the network are preserved. This method, however, is rather conservative: using the update rule (5) a node that experienced a failure will not update, preventing the node to introduce errors in the network, but when the probability of packet loss is high, several nodes might be in an idle-status, slowing down the network convergence. While those nodes are idle, however, the nodes that are updating might speed up the process, e.g. they could update *twice*, using the information received during the last successful round, hopefully going closer to the average. Using this empiric method leads to a weighted adjacency matrix as follows:

$$W(k) = I + 2F(k)\overline{W}F(k) - 2\,\mathrm{diag}(F(k)\overline{W}F(k)\mathbb{1})$$

Simulations show that this approach actually increases the speed of convergence of the network, for high probability of packet loss, but would lead to instability if the probability of packet loss is low. Generalising, it is possible to speed up the convergence process, using a gain α, defining the update rule as follows:

$$\begin{aligned} x_i(k+1) =& \alpha f_i \left(\sum_{j=1}^{n} f_j \overline{w}_{ij} x_j(k) + \sum_{j=1}^{n} (1-f_j)\overline{w}_{ij} x_i(k) \right) \\ &+ (1-\alpha f_i) x_i(k) \end{aligned} \tag{8}$$

which can be described, in a vectorial form, as:

$$W_\alpha(k) = I + \alpha F(k)\overline{W}F(k) - \alpha\,\mathrm{diag}(F(k)\overline{W}F(k)\mathbb{1}) \tag{9}$$

with $\alpha > 1$.

Modifying the compensation algorithm using the gain α leads to an increase in performance but the new $W_\alpha(k)$ matrix is not necessarily stochastic anymore: some of its entries might be negative. However, even if the matrix $W_\alpha(k)$ is not stochastic for every k anymore, the system converges for some values of α, faster than when $\alpha = 1$.

It is possible to investigate the relation between the choice of the gain α and the convergence of the network.

We will show that the condition $\mathbb{E}[W_\alpha^T W_\alpha] \geq 0$ componentwise leads to convergence in mean square sense; for this we need to further investigate the error dynamics, defined as:

$$y(k) = x(k) - \frac{1}{n}\mathbb{1}\mathbb{1}^T x(0), \tag{10}$$

where n denotes the number of agents in the network, i.e. the number of vertices in the graph. The convergence in mean square sense is then:

$$\lim_{k\to\infty} \mathbb{E}[y^T(k)y(k)] = 0. \tag{11}$$

From the update rule of $x(k)$ follows that, remembering that $W_\alpha(k)\mathbb{1} = \mathbb{1}$:

$$\begin{aligned} y(k+1) &= x(k+1) - \frac{1}{n}\mathbb{1}\mathbb{1}^T x(0) \\ &= W_\alpha(k)x(k) - \frac{1}{n}W_\alpha(k)\mathbb{1}\mathbb{1}^T x(0) \\ &= W_\alpha(k)\left(x(k) - \frac{1}{n}\mathbb{1}\mathbb{1}^T x(0)\right) \\ &= W_\alpha(k)y(k). \end{aligned}$$

It is possible to study the convergence of the expected value of $y(k+1)^T y(k+1)$, for which the following theorem holds true:

Theorem III.3. *A system described by*

$$x(k+1) = W_\alpha(k)x(k),$$

where

$$W_\alpha(k) = I + \alpha F(k)\overline{W}F(k) - \alpha\,\mathrm{diag}(F(k)\overline{W}F(k)\mathbb{1}),$$

converges in mean square sense, according to (11), if:

$$\alpha < \frac{2\overline{w}_{ij}}{\Xi_{ij}} \tag{12}$$

for each $i, j \neq i$, for which $\Xi_{ij} > 0$, where:

$$\Xi_{ij} = 2p\overline{w}_{ij}^2 + 2(1-p)\overline{w}_{ij} - (1-p)[\overline{W}^T\overline{W}]_{ij}. \tag{13}$$

Proof. The expected value of the squared error is:

$$\mathbb{E}[y(k+1)^T y(k+1)|y(k)] = y(k)^T \mathbb{E}[W_\alpha(k)^T W_\alpha(k)]y(k),$$

and:

$$\begin{aligned} \mathbb{1}^T y(k) &= \mathbb{1}^T W_\alpha(k-1)y(k-1) = \mathbb{1}^T y(k-1) \\ &= \cdots = \mathbb{1}^T y(0) = \mathbb{1}^T x(0) - \frac{1}{n}\mathbb{1}^T\mathbb{1}\mathbb{1}^T x(0) = 0 \end{aligned}$$

where we used the doubly stochasticity property of $W_\alpha(k)$ to show $y(k) \perp \mathbb{1}$.

For a symmetric matrix $A \in \mathcal{R}^{n\times n}$, with eigenvalues $\lambda_n \leq \lambda_{n-1} \leq \cdots \leq \lambda_1$, the following holds true:

$$z^T A z \leq \lambda_2 z^T z$$

if $z \perp e_1$, where e_1 is the eigenvector associated to the λ_1 eigenvalue. Since $y(k) \perp \mathbb{1}$ and $\mathbb{E}[W_\alpha^T W_\alpha]$ is symmetric, it follows that:

$$\mathbb{E}[y(k+1)^T y(k+1)|y(k)] \leq \lambda_2\left(\mathbb{E}[W_\alpha^T W_\alpha]\right) y^T(k)y(k).$$

The time dependence is dropped, because the matrices $W_\alpha(k)$ are mutually independent, according to the failure independence property described in the previous section. Recursively, we get:

$$\mathbb{E}[y(k+1)^T y(k+1)] \leq \lambda_2^{(k+1)}\left(\mathbb{E}[W_\alpha^T W_\alpha]\right)\mathbb{E}[y(0)^T y(0)].$$

Then a sufficient condition for

$$\lim_{k\to\infty} \mathbb{E}[y^T(k)y(k)] = 0$$

is that $\mathbb{E}[W_\alpha^T W_\alpha]$ is stochastic and that the graph associated with $\mathbb{E}[W_\alpha^T W_\alpha]$ is connected: this ensures that the absolute value of all the eigenvalues of $\mathbb{E}[W_\alpha^T W_\alpha]$ is smaller than one except for $\lambda_1 = 1$.

The matrix $\mathbb{E}[W_\alpha^T W_\alpha]$ can be explicitly computed:

$$\mathbb{E}[W_\alpha^T W_\alpha] = \alpha^2(1-p)^3\overline{W}^T\overline{W} -$$

$$2\alpha^2 p(1-p)^2 \overline{W} \odot \overline{W} +$$
$$2\alpha(1-p)^2 \overline{W} - 2\alpha^2(1-p)^3 \overline{W} +$$
$$(\alpha^2(1-p)^3 - 2\alpha(1-p)^2 + 1)I +$$
$$2\alpha^2 p(1-p)^2 \operatorname{diag}(\overline{W}^T \overline{W}),$$

where $\odot$ denotes the Hadamard – i.e. element-wise – product.

By construction, $\mathbb{E}[W_\alpha^T W_\alpha]\mathbb{1} = \mathbb{1}$. Therefore, the matrix is stochastic if $\mathbb{E}[W_\alpha^T W_\alpha]_{ij} \geq 0$ for all $i, j \in \mathcal{N}$

It is easily verified that the diagonal elements of $\mathbb{E}[W_\alpha^T W_\alpha]$ are positive using that W_α is symmetric. It is sufficient to find α such that the elements outside the diagonal of $\mathbb{E}[W_\alpha^T W_\alpha]$ are positive whenever the corresponding element of $\overline{W}$ is positive and otherwise at least nonnegative.

This would guarantee that $\mathbb{E}[W_\alpha^T W_\alpha]$ is stochastic and the associated graph is connected whenever the graph associated with $\overline{W}$ is connected, which is assumed true.

To show this property for the off-diagonal elements, we note that it is possible to evaluate $\mathbb{E}[W_\alpha^T W_\alpha]$ element-wise:

$$\begin{aligned}\mathbb{E}[W_\alpha^T W_\alpha]_{ij} &= \alpha^2(1-p)^3[\overline{W}^T \overline{W}]_{ij} - 2\alpha^2 p(1-p)^2 \overline{w}_{ij}^2 + \\ &\quad 2\alpha(1-p)^2 \overline{w}_{ij} - 2\alpha^2(1-p)^3 \overline{w}_{ij} \\ &= (1-p)^2 \alpha \left(-\alpha \Xi_{ij} + 2\overline{w}_{ij}\right) \; \forall i \neq j.\end{aligned}$$

where Ξ_{ij} is defined in (13). The condition for the network to converge is then

$$-\alpha \Xi_{ij} + 2\overline{w}_{ij} \geq 0 \tag{14}$$

whenever $\overline{w}_{ij} = 0$ and

$$-\alpha \Xi_{ij} + 2\overline{w}_{ij} > 0 \tag{15}$$

otherwise. It is easily verified that $\overline{w}_{ij} = 0$ implies that $\Xi_{ij} \leq 0$ and hence (14) is satisfied. If $\Xi_{ij} \leq 0$ and $\overline{w}_{ij} > 0$, then (15) is satisfied. If $\Xi_{ij} > 0$ then (15) becomes (12) which completes the proof. □

It should be noticed that this condition is not restrictive since $\alpha = 1$ always satisfies (12). After all, if $\Xi_{ij} > 0$ then $\overline{w}_{ij} > 0$ and hence (12) with $\alpha = 1$ is equivalent to:

$$2\overline{w}_{ij} - \Xi_{ij} > 0$$

which, using the definition of Ξ_{ij}, is equivalent to:

$$2p\overline{w}_{ij}(1 - \overline{w}_{ij}) + (1-p)[\overline{W}^T \overline{W}]_{ij} > 0$$

which is clearly satisfied since $1 > \overline{w}_{ij} > 0$ and $[W^T W]_{ij} \geq 0$.

The bound for α allows for an *a priori* tuning of the parameter, to ensure convergence when a stochastic modelling of the network is known. It is possible to pick a *safe* α_s defined as the biggest α that satisfies all the inequalities (15), i.e.:

$$\alpha_s = \min_{ij \text{ s.t.} \Xi_{ij} > 0} \left(\frac{2\overline{w}_{ij}}{\Xi_{ij}} \right).$$

Any $\alpha \leq \alpha_s$ will secure convergence of the network to the average of the initial conditions of the nodes: this can be derived from the double stochasticity of the $\mathbb{E}[W_\alpha^T W_\alpha]$ matrix. However, the aforementioned condition is only sufficient: it might be possible to pick bigger values for α which might make the network converge faster, but if α exceeds certain – unknown – threshold, the network might instead reach instability, Intuitively, the gain should be one when the probability of packet loss is equal to 0, and it should increase when the probability of packet loss increases. Moreover, it might depend on the size of the network: small networks will experience a lower amount of packet losses, simply because the number of links in the network is lower, thus a smaller gain should be employed, because the number of idled nodes will be lower. From those intuitive assumptions, it is possible to choose α empirically as:

$$\alpha_h = \frac{1}{(1-p) + \frac{p}{n}}. \tag{16}$$

Simulations suggest that this heuristic value makes the network converge and that – in some cases – convergence is faster than using α_s, even though convergence is not guaranteed by our theoretical results.

IV. Distributed asynchronous algorithms

The compensation methods presented in the previous section must be translated into an algorithm, which describes the operations that each node must perform in order to achieve the dynamics compensation described by (5) and (9) which will be called the AP and the α-AP methods respectively.

The two methods aim to keep the weight-adjacency matrix doubly stochastic, modifying the weight of the links between the nodes. In order to achieve such a weight distribution dynamically, each node needs to know if its last communication was successful or not before updating its value. This network awareness can be obtained using explicit acknowledgements, i.e. dedicated messages sent by the receiver to acknowledge a successful transmission to the original sender. Such a solution, however, might not be realistic in ad-hoc networks: even if an acknowledgement message is shorter, its probability of collision is lower but can not be simply neglected. To provide feedback from the network to each node we propose an implicit acknowledgement, i.e. the acknowledgement information is carried by the same message that carries the state. However, these *informative* messages, carrying the states, are not reliable: this implies that some acknowledgements might be lost, impairing the compensation methods effectiveness, as some node would compensate whereas it should not. We propose a local correction method, in *addition* to the AP and α-AP compensation methods, to be performed asynchronously by each node, to overcome the acknowledgements losses. It should be noticed that the AP (α-AP) compensation modifies the dynamics of the network, while the asynchronous correction step is a technical exploit to be able to practically implement the acknowledgements.

A. The main algorithm

The compensation method can be summarised by the following procedure: each node starts in an *initialisation* phase, followed shortly by its first *sending* phase. During the sending phase, each node asynchronously broadcasts a message to its neighbours, sending its state and the acknowledgment to

each neighbour. After the sending phase, the node moves to a *receiving* phase, during which the node waits for incoming packets, until a timer triggers. All the packets received after the timer triggers, are lost. Lastly, each node computes its new state, using the information gathered during the previous phase, during the so called *computing* phase.

We detail the behaviour for the i^{th} node, which has n_i neighbours, indicated by the set $\mathcal{N}_i$. A superscript indicates the local variables kept in its memory by the node, as $v^{(i)}$. Each node has a memory, where it stores: 1) its state, $x_i^{(i)}(k)$; 2) for each neighbour $j \in \mathcal{N}_i$ the iteration number of the latest transmission from j received by i, denoted as $t_j^{(i)}$; 3) the last iteration number during which it broadcast a packet, $t_i^{(i)}$. Moreover we assume that the communication between two nodes in the same neighbourhood happens without errors at least once every $\hat{k} < \infty$ rounds, that each node has perfect information of its neighbourhood in its *nominal*, i.e. without failures, condition, and we neglect computation and sending times. The phases of the algorithm can be detailed as follow:

- *Initialisation Phase*:
  Before the first communication round, each node i switches on. Moreover, each node has its state initialised to some initial value $x_i^{(i)}(0)$, assumed random in the following of the paper.
  When the consensus routine begins each node moves to its first sending phase, at some random time t_s^i, and we assume that $t_s^j \neq t_s^i$ for all $\forall j \in \mathcal{N}_i, j \neq i$: we assume that two nodes will not move to the first transmission phase exactly at the same time in the same neighbourhood. If two nodes try to communicate at the exact same time, due to the homogeneous nature of the nodes, their communication will always end up in a collision, according to out theoretical model. The aforementioned hypothesis can be relaxed, e.g. it is possible to each node to add a small random delay before every transmission, to prevent the case where the communication between two nodes always ends up in a collision.
- *Sending Phase*: During the k^{th} consensus iteration, the i^{th} node sends its message, according to the following steps:
  1) it creates its message, containing the following:
     - $x_i^{(i)}(k)$;
     - $t_i^{(i)} = k$;
     - for each neighbour $j \in \mathcal{N}_i$ $t_j^{(i)}$, which have been stored during the previous receiving phase – they are initialised to 0 during the first sending phase –
  2) it broadcasts the aforementioned packet to every neighbour;
  3) it moves to the receiving phase.
- *Receiving Phase*: In the receiving phase each node waits for incoming packets. This phase is characterised by a timer: the node waits for incoming packets until a timer triggers — after a certain T_{out} — and after it all remaining packets are lost. The parameter T_{out} needs to be chosen taking into account the average message latency, and the density of the network.
  For each neighbour $j \in \mathcal{N}_i$, the node i expects $[x_j^{(j)}, t_j^{(j)}, t_i^{(j)}]$. The round index $t_i^{(j)}$ is used as acknowledgement: if the node receives all $t_i^{(j)} < t_i^{(i)}$, then there will not be any update for the node, as it means that its last communication was unsuccessful. When the receiving phase ends, the node moves to the computing phase.
- *Computing Phase*: After the receiving phase, the node can update its state, by compensating for the missing packets, according to the following:
  $$x_i^{(i)}(k+1) = W_{\alpha i}(k) x^{(i)}(k),$$
  where $x^{(i)}(k)$ is the collection of the received states during the k^{th} iteration, i.e. $x^{(i)}(k) = [x_1^{(i)}(k) \dots x_n^{(i)}(k)]$, and $W_{\alpha i}(k)$ is the i^{th} row of the matrix describing the compensation method. This algorithm can be used to implement both the AP and α-AP methods, using the value of $\alpha = 1$ leads to the simple average preserving compensation method.

A pseudocode describing the algorithm can be found in Algorithm 1.

As mentioned before, the proposed compensation methods use feedback from the network, more specifically acknowledgements from other nodes, which are written in the *informative* messages. Such messages are unreliable, therefore acknowledgements can be missed. If this is the case, consensus is still achieved asymptotically, but the average is not preserved anymore. To cope with this problem, it is necessary to employ an extra compensation step, the *Asynchronous Fallback Correction*. Moreover, another source of wrong acknowledgement is the nodes' transmission order. If the nodes in a neighbourhood do not experience any fail, the order of sending messages during one round does not change, and clearly the last node to transmit will not receive any acknowledgement, even if its message is correctly delivered: it will receive the acknowledgement for round k only during round $k+1$. This problem is also solved by the asynchronous fallback correction step, which enables the nodes to compensate for delayed acknowledgements.

B. The asynchronous fallback correction step

When a node sends its message successfully, but all the neighbours fail in acknowledging the successful communication, then the sender node performs its compensation — i.e. does not update — while the neighbours use the sender's state to update themselves. When this happens, the average of the network is compromised.

However, if a node would be able to understand if it experienced a *virtual failure*, i.e. it did not update because the acknowledgments were missing, whereas its message was received successfully, then it could compensate for it, updating its value outside the aforementioned scheduling.

To explain the mechanism, consider the restricted case in which there is only one virtual failure event in the network, so that the sending node will not update because the neighbours failed to acknowledge its previous message. Let us assume that the nodes use a cumulative acknowledgement strategy: they

Algorithm 1 Compensation algorithm - α-*AP*.

procedure INITIALIZATION
  for each node i **do**
    $k = 1$
    $x = x(0)$
    goto: Sending
  end for
end procedure
procedure SENDING
  for each node i **do**
    $t_i^{(i)} \leftarrow k$
    $f_I^{(i)} = 0$
    Broadcast $[x_i^{(i)}(k), t_i^{(i)}, t_j^{(i)}]$
    goto: Receiving
  end for
end procedure
procedure RECEIVING
  for each node i **do**
    $r_p = 0$;
    $x_j^{(i)} = 0 \, \forall j$
    $\hat{f}_j^{(i)} = 0 \, \forall j$
    while $t < T_{out} \wedge r_p < |\mathcal{N}_i|$ **do**
      $t = t + 1$
      upon event incoming packet $[x_j^{(j)}, t_j^{(j)}, t_i^{(j)}]$ **do**
        $t_j^{(i)} \leftarrow t_j^{(j)}$
        $x_j^{(i)} \leftarrow x_j^{(j)}$
        $\hat{f}_j^{(i)} = 1$
        $r_p = r_p + 1$
        if $t_i^{(j)} = t_j^{(j)}$ **then**
          $f_i = 1$
        end if
    end while
    goto: Computing
  end for
end procedure
procedure COMPUTING
  for each node i **do**
    $x_i^{(i)} = x_i^{(i)} + \alpha f_i \sum_{j=1}^{n} \hat{f}_j^{(i)} \overline{w}_{ij} \left(x_j^{(i)} - x_i^{(i)} \right)$
    $k = k + 1$
    goto: Sending
  end for
end procedure

keep in memory not only the last successful communication round number received by neighbours, but all the successful rounds. If this is the case, when a node receives a packet, it can check if it experienced a virtual failure: if all the successful rounds are stored and transmitted, a node can see if it receives an acknowledgement from the past, i.e. it receives a $\hat{t}_i^{(j)}$ in which it performed no update, compromising the average of the network. When this happens, the node can simply compensate according to the following:

$$x_i^{(i)^+} \leftarrow x_i^{(i)} + \hat{x}_i - x_i^{(i)}(\hat{t}_i^{(j)})$$

Algorithm 2 Asynchronous fallback correction algorithm.

procedure ASYNCHRONOUS FALLBACK COMPENSATION
  for all $t_i^{(j)}$ **do**
    if $f_i^{(i)}(t_i^{(j)}) = 0$ **then**
      $x_e = \sum_{j=1}^{n} f_j^{(i)}(t_i^{(j)}) \overline{w}_{ij} \left(x_j^{(i)}(t_i^{(j)}) - x_i^{(i)}(t_i^{(j)} \right)$
      $\hat{x}_i = x_i^{(i)}(t_i^{(j)}) + \alpha x_e$
      $x_i^{(i)}(k) = x_i^{(i)}(k) + \hat{x}_i - x_i^{(i)}(t_i^{(j)})$
    end if
  end for
end procedure

where $x_i^{(i)}(\hat{t}_i^{(j)})$ is the state at time $\hat{t}_i^{(j)}$, when the node should have performed the update, but it did not, and $\hat{x}_i$ is the state that it should have computed at that time, if it would have received a proper acknowledgement. In the above formulas, the round k is not included in the notation: this indicates that this compensation can happen at any time a virtual failure is recognised.

To compute $\hat{x}_i$, the node needs to keep track of the received messages — i.e. the states — from the network. When a node receives an out-of-order acknowledgement, it updates itself asynchronously, according to the following:

$$\hat{x}_i = x_i^{(i)}(\hat{t}_i^{(j)}) + \alpha \sum_{j=1}^{n} \hat{f}_j^{(i)}(\hat{t}_i^{(j)}) \overline{w}_{ij} \left(x_j^{(i)}(\hat{t}_i^{(j)}) - x_i^{(i)}(\hat{t}_i^{(j)}) \right).$$

When a node compensates for an acknowledgement received out of order, it can free the memory for that round: the only states that he keeps in memory are the ones relative to rounds when he did not receive an acknowledgement. Moreover, we assumed that the communication between any pair of nodes in a neighbourhood happens successfully at least once every $\hat{k} < \infty$ rounds: it is possible to free the memory up to a round index $\tilde{k}$ if at least one message from each node in the neighbourhood has been received after $\tilde{k}$. The amount of memory required by each node is then finite, as every state kept in memory can be deleted at some point. In Figure 1 it is possible to appreciate the difference when the asynchronous fallback correction step is employed. Until a virtual failure happens, the average of the network remains the correct one whether or not the fallback correction is employed, deviating from the blue line only when the nodes update themselves. However, without the correction step the average of the network – red line – deviates from the correct average at some point, while when the correction step is employed – green line – the average of the network remains the correct one. A pseudo-code, describing the asynchronous fallback compensation routine can be found in Algorithm 2

This additional correction mechanism, to overcome the acknowledgements losses, requires each node to keep track of the neighbours previous successful transmissions, and their previous states. This implies that each node needs some memory to keep track of the previous states of the system.

C. Buffer design

It is possible to characterise the amount of memory required by each node. Let us assume that in a given consensus round the talking order in each neighbourhood is random: a reasonable assumption is that the node i has a probability $\frac{1}{d_i}$, where d_i is the number of neighbours in its neighbourhood $\mathcal{N}_i$, to be the first one to send its message in its neighbourhood, and that the probability to be the second, third and so on up to the d^{th} is the same. Clearly, a node will not receive any acknowledgement of successful communication during round k when it is the last one sending a message in the k^{th} round, moreover a node will not receive any acknowledgement of successful communication if all the other nodes, communicating after it, will fail. The probability of a *virtual failure* is then:

$$\begin{aligned} p_v &= \frac{1}{d_i}(1-p) + \frac{1}{d_i}p(1-p) + \cdots + \frac{1}{d_i}p^{d_i-1}(1-p) \\ &= \frac{1-p^{d_i}}{d_i}, \end{aligned}$$

where the first term represents the case when the node i is the last one communicating during a round, the second term is the probability of having only one node scheduled to transmit a message after node i, failing, and the last term is the probability for node i to be the first one to communicate during a round, and that all the neighbours fail in the communication.

Define an *overflow* event as the event that arises when there are m subsequent virtual failures. We are interested in the probability that an overflow happens in n consensus iterations. Let us indicate the probability that an overflow – i.e. m subsequent virtual failures – happens in n consensus iterations as $y_n^{(m)}$. In the following, the superscript m will be omitted for readability, but clearly the probability to experience an overflow depends on the number of subsequent failures that will accumulate, i.e. m. We recall a simple but instructive result:

Lemma IV.1. *Consider a sequence of Bernoulli trials of length n, where the probability of* failure *is denoted as p_v. Then the probability that exists a sequence of m failures in n trials, denoted as y_n, satisfies the following recursion:*

$$\begin{aligned} y_{n+1} &= y_n + (1-y_{n-m})(1-p)p^m \\ y_0 &= y_1 = \dots y_{m-1} = 0 \\ y_m &= p^m. \end{aligned} \tag{17}$$

Proof. Before computing the probability y_n, we notice how the probability of obtaining a sequence of m failures in a sequence of n trials is trivially 0 if $m < n$, and it is equal to p^m if $m = n$. We now compute the probability that a sequence of m subsequent failures happens in $n+1$ iterations. If such overflow happens in $n+1$ iterations, it can happen in two mutually exclusive ways: an overflow happened in the first n iterations, or it did not happen in the first $n-m$ iteration, but the last m iterations ended in a failure and the $n-m-1$ iteration was a success. The probability for the first event is simply y_n, while the probability for the second one is $(1-y_{n-m})(1-p_v)p_v^m$. Being the two events mutually exclusive, the probability y_{n+1} is the sum of the two:

$$y_{n+1} = y_n + (1-y_{n-m})(1-p_v)p_v^m,$$

which ends the proof. □

It is possible to devise an explicit solution for (17) to obtain y_n, which reads:

$$y_n = 1 - a_n + p_v^m a_{n-m}$$

where:

$$a_x = \sum_{j=0}^{\frac{x}{m+1}} \binom{x-mj}{j} (-p_v^m(1-p_v))^j$$

but the proof it is rather long and thus omitted. Moreover, the numerical computation of a_n and a_{n-m} is time consuming. The recursion (17) is then the preferred method to compute y_n.

The knowledge of the overflow probability allows for a buffer design: it is possible to pick m such that the probability of having an overflow in n rounds is arbitrarily small, and the buffer size $\mathcal{B}_i$ to handle overflow events of m subsequent failures for node i is $\mathcal{B}_i = m|\mathcal{N}|$.

In Figure 2 the relation between the number of consensus iterations and the buffer size to achieve a probability of overflow smaller than 0.01 is presented.

Another way of designing the buffer size is to pick its size such that it handles a number of failures proportional to the expected value of the longest run of failures. An estimate of the length of the longest run of failures in a Bernoulli trial of length n is given in [19] as:

$$\mathcal{L} = \log_{1/p_v}(n(1-p_v))$$

and then it is possible to pick m equal to $\mathcal{L}$ for an intuitive design of buffer size.

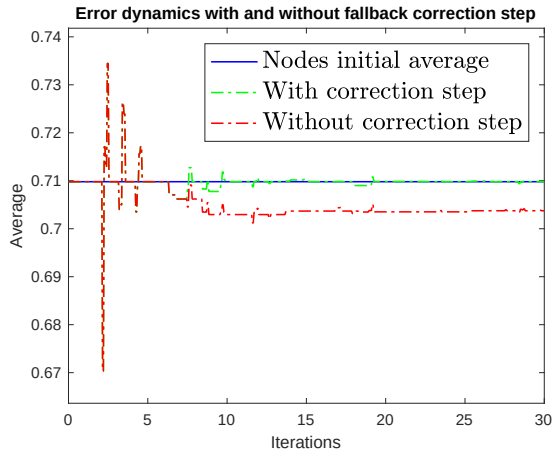

Figure 1: Evolution of the states' average of the network with and without asynchronous fallback correction.

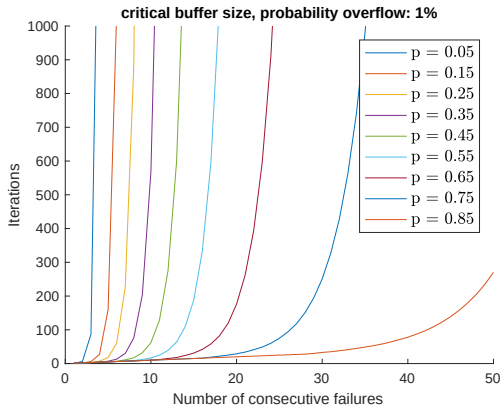

Figure 2: Buffer size to achieve a probability of overflow smaller than 1%.

V. Convergence speed and simulation results

After proving the convergence of the new compensation mechanisms and devising an algorithm to implement them, we are interested in evaluating their speed of convergence. Although it is possible to compute the exponential *rate of convergence* to the network average, it does not lead to insightful closed-form expressions in the general case [20], [21]. The rate of convergence could be bounded solving an eigenvalue problem: however, the computation of such eigenvalues is still generally difficult, unless the scenario is restricted to some specific topology: an extended study of the convergence rate bounds for circulant and complete topologies can be found in [16]. Moreover, when the update rule is modified with a gain α, the eigenvalue problem becomes even more difficult. While it is possible to find a value for α which assures convergence, the number of iteration required to converge can not be easily estimated beforehand, unless we restrict to some special topology. To obtain a general performance evaluation we run a set of simulations. To provide a meaningful set of results, the simulations are performed varying several factors:

- the probability of packet loss, p;
- the size of the network, n;
- the "connectivity probability", m, defined below.

The parameter n is the number of nodes – i.e. the agents – of the network. The parameter m defines the connectivity of the network: for each pair of nodes, there is a connection between them with probability m. A higher value for m implies a larger number of connections in the network. The networks are generated without any given structure, but when a network with disconnected components is generated, it is discarded. Lastly, during each consensus round, a node transmits a packet to its neighbours with a success probability equal to $1-p$. For each triplet (n, m, p), 145 simulations are performed, and the mean is displayed. We simulated the synchronous dynamics, described by (4), and the convergence is practically reached when:

$$||x(k) - \frac{1}{n}\mathbb{1}\mathbb{1}^T x(k)|| < 10^{-10}.$$

We are primarily interested in the performance difference when a gain is employed, and secondarily we compare different choices of the gain α. Therefore, the simulations are conducted using three different values for α:

- $\alpha = 1$, representing the case when the gain is not used;
- α_s, which assures convergence;
- α_h, the heuristic value for α in (16).

To increase readability, the number of iterations required to converge is shown over only two parameters: p and m, because the dependence of the number of iterations required to converge on the size of the network appears to be negligible in comparison with the dependence on the probability of packet loss and the connectivity of the network: for a given probability of packet loss, the simulations lie in an interval smaller than 10% of the displayed result.

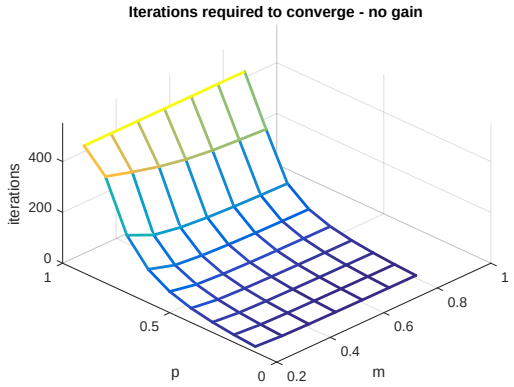

Figure 3: Iterations required to converge, for a network of 81 nodes, without gain, i.e. $\alpha = 1$.

The iterations required to converge, when $\alpha = 1$, are depicted in Figure 3. For high probability of packet loss, the number of iterations required to converge hits the limit of 500 – the upper limit imposed in the simulations – but we can infer that the actual number of iteration required would be bigger. We can notice two different trends: the number of iterations required decreases as the probability of packet loss decreases, and as the connectivity of the network increases. This result is expected and intuitive. In Figures 4 is displayed the number of iterations required to converge using the safe value for α.

First, we can notice how even for high probability of packet loss, the algorithm converges in less than 500 iterations. Moreover, the curve is always below the one in Figure 3: using the gain does actually increase the convergence speed. This result can be seen more clearly in Figure 5.

In Figure 5 it is possible to appreciate a better comparison between the choice of α_s or α_h: this plot is obtained averaging all the simulations, over m and n, to show only the dependence of the number of iteration from the probability of packet loss – which is the principal cause which slows down the convergence. The graph shows how the performances of the two methods are similar: employing the gain α_h leads to slightly better performances for lower probability of packet loss, but the differences are minimal. The bound for α is not

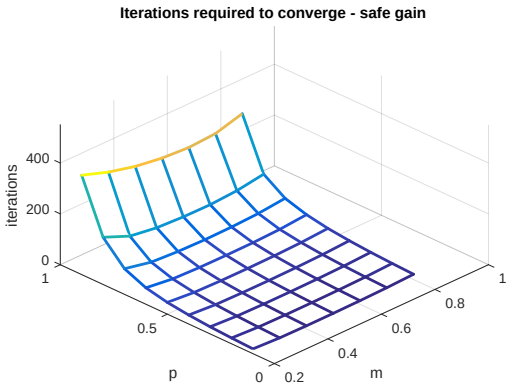

Figure 4: Iterations required to converge, for a network of 81 nodes, using a safe gain α_s.

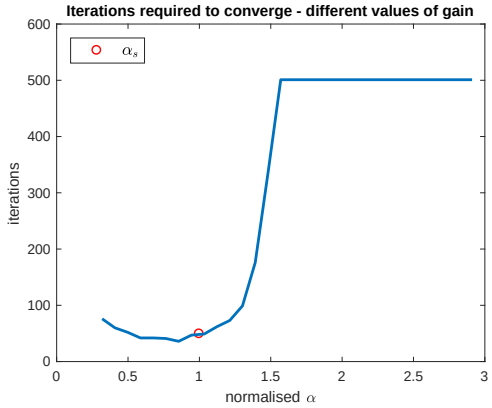

Figure 6: Iterations required to converge for different values of α. The x axis is normalised: the value 1 correspond to a_s.

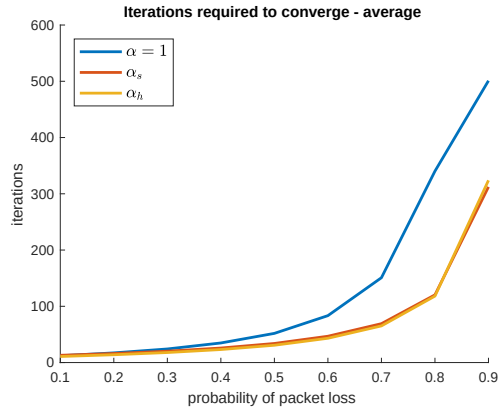

Figure 5: Iterations required to converge for different values of α, average over different network sizes n and connectivity degrees m.

the optimal one. It is possible to find different values of gain to speed up the convergence of the network. However, it is interesting to evaluate how good the value of α_s is, to check whether this choice is limiting or not. Figure 6 displays the number of iteration required for a network to converge, for different values of α. The values are normalised over α_s, in a range from $0.5\alpha_s$ to $3\alpha_s$. Choosing the safe value – circled in red – is indeed not a limiting factor: it would be possible to pick a different gain value to converge faster, but the increase in performance is not dramatic. Moreover, the value of α that achieves the fastest convergence is not known a priori, and it can be extrapolated only trying different gains, while α_s can be computer beforehand.

Lastly, it is interesting to compare our algorithm with another from the literature. For a fair comparison, the algorithm presented by Hadjicostis in [14] has been chosen, due to its good performance and to its structure: it is possible to simulate both algorithms at the same time, relying on the same infrastructure and message passing mechanisms, so that their speed difference is only due to the compensation strategy. The two algorithms differ in the use of memory: the Hadjicostis algorithm requires every node to have two variables, instead of one, and the nodes have to agree on these two variables, namely the average and the distance from the average introduced by packet drops. To keep their implementation simple, a circulant topology has been chosen: the nodes are organised in a ring, and each node communicates with a symmetric set of neighbours, $\frac{l-1}{2}$ in the clockwise direction and $\frac{l-1}{2}$ in the anticlockwise direction. This ensures a topology where each node has the same degree, and Hadjicostis' algorithm requires from each node the knowledge of its degree.

The results of this comparison, for a network of 80 nodes, connected to 20 neighbours each, are presented in Figure 7. The two algorithms have similar performance, but the novel

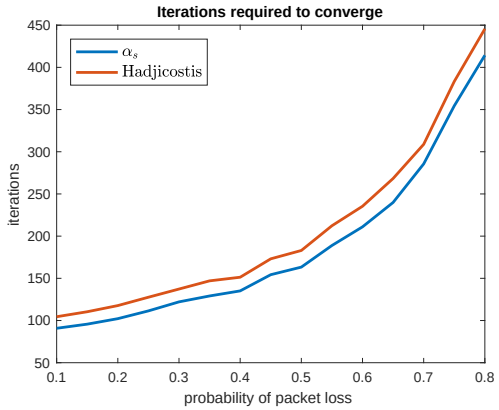

Figure 7: Average number of iterations required to converge. Comparison between the Hadjicostis algorithm and ours .

one is slightly faster. This trend appears in a variety of different parameter choices, but our novel algorithm is slower than Hadijcostis' for more *dense* networks, i.e. when a node is connected to a higher percentage of other nodes in the network, while it is faster when the network is more sparse.

VI. Conclusion and future work

The compensation methods presented in this work achieve their principal aim: they both make the network converge to the *exact* average value of the nodes' initial conditions, regardless of packet losses, and the convergence speed can be improved using a gain. The presence of such a gain might lead to instability in the network's dynamic, but it is possible to find conditions to both secure stability and accelerate the convergence of the network.

An algorithm to implement the compensation methods has also been proposed: it is capable of implementing the dynamics described by the two methods, relying on implicit acknowledgements. The issue of virtual losses, due to the loss of the implicit acknowledgements, can be addressed by the proposed asynchronous fallback strategy.

As future work, it is possible to further investigate how tight the stability bound is: the heuristic value of α_h appears to stabilise the network, even when it is bigger than the safe value α_s. Moreover, the analysis of performance and the search for a gain is made without taking into account the differences in the nodes: every node uses the same gain α. It would be possible to find a set of different gains, one for each node, stabilising the network. This would allow – hopefully – faster convergence stil preserving stability. Lastly, it can be investigated how the use of a gain affects the convergence rate of the consensus algorithm in a more explicit way.

References

[1] M. Alanyali, S. Venkatesh, O. Savas, and S. Aeron, "Distributed Bayesian hypothesis testing in sensor networks," in *American Control Conference (ACC)*, vol. 6, June 2004, pp. 5369–5374.

[2] D. Blatt and A. Hero, "Distributed maximum likelihood estimation for sensor networks," in *IEEE International Conference on Acoustics, Speech, and Signal (ICASSP)*, vol. 3, May 2004, pp. 929–932.

[3] R. Carli, A. Chiuso, L. Schenato, and S. Zampieri, "Distributed Kalman filtering based on consensus strategies," *IEEE Journal on Selected Areas in Communications*, vol. 26, no. 4, pp. 622–633, 2008.

[4] S. Kar and J. Moura, "Distributed consensus algorithms in sensor networks with imperfect communication: Link failures and channel noise," *Signal Processing, IEEE Transactions on*, vol. 57, no. 1, pp. 355–369, Jan 2009.

[5] T. Li and J.-F. Zhang, "Consensus conditions of multi-agent systems with time-varying topologies and stochastic communication noises," *Automatic Control, IEEE Transactions on*, vol. 55, no. 9, pp. 2043–2057, Sept 2010.

[6] U. Schmid and C. Fetzer, "Randomized asynchronous consensus with imperfect communications," in *International Symposium on Reliable Distributed Systems*, Oct 2003, pp. 361–370.

[7] K. Bilstrup, E. Uhlemann, E. Ström, and U. Bilstrup, "Evaluation of the IEEE 802.11p MAC method for Vehicle-to-Vehicle communication," in *IEEE Vehicular Technology Conference (VTC)*, Oct. 2008, pp. 1 – 5.

[8] F. Fagnani and S. Zampieri, "Average consensus with packet drop communication," *SIAM Journal on Control and Optimization*, vol. 48, no. 1, pp. 102–133, 2009.

[9] P. Frasca and J. M. Hendrickx, "On the mean square error of randomized averaging algorithms," *Automatica*, vol. 49, no. 8, pp. 2496 – 2501, 2013.

[10] ——, "Large network consensus is robust to packet losses and interferences," in *European Control Conference (ECC)*, Zürich, Switzerland, Jul. 2013, pp. 1782–1787.

[11] Y. Chen, R. Tron, A. Terzis, and R. Vidal, "Corrective consensus: Converging to the exact average," in *IEEE Conference On Decision And Control (CDC)*, Dec 2010, pp. 1221–1228.

[12] K. Cai and H. Ishii, "Average consensus on general strongly connected digraphs," *Automatica*, vol. 48, no. 11, pp. 2750 – 2761, 2012.

[13] N. Vaidya, C. Hadjicostis, and A. Dominguez-Garcia, "Robust average consensus over packet dropping links: Analysis via coefficients of ergodicity," in *IEEE Conference On Decision And Control (CDC)*, Dec 2012, pp. 2761–2766.

[14] C. N. Hadjicostis, N. H. Vaidya, and A. D. Domínguez-García, "Robust distributed average consensus via exchange of running sums," *Automatic Control, IEEE Transactions on*, vol. 61, no. 6, pp. 1492–1507, Jun. 2016.

[15] S. Wang, C. An, X.-X. Sun, and X. Du, "Average consensus over communication channels with uniform packet losses," in *Chinese Control and Decision Conference (CCDC)*, May 2010, pp. 114–119.

[16] F. Acciani, G. Heijenk, and P. Frasca, "Achieving robust average consensus over wireless networks," in *European Control Conference (ECC)*, Jun. 2016, pp. 555–560.

[17] F. Acciani, P. Frasca, G. Heijenk, and A. Stoorvogel, "Using a linear gain to accelerate average consensus over unreliable networks," in *IEEE Conference On Decision And Control (CDC)*, Dec. 2017.

[18] V. Rai, F. Bai, J. Kenne, and K. Laberteaux, "Cross- channel interference test results: A report from the VSC-A project," IEEE 802.11 Task Group p, Tech. Rep., July 2007.

[19] M. F. Schilling, "The longest run of heads," *College Math. J*, vol. 21, no. 3, pp. 196–207, 1990.

[20] A. Tahbaz-Salehi and A. Jadbabaie, "A necessary and sufficient condition for consensus over random networks," *Automatic Control, IEEE Transactions on*, vol. 53, no. 3, pp. 791–795, 2008.

[21] F. Iutzeler, P. Ciblat, and W. Hachem, "Analysis of sum-weight-like algorithms for averaging in wireless sensor networks," *Signal Processing, IEEE Transactions on*, vol. 61, no. 11, pp. 2802–2814, Jun. 2013.

Francesco Acciani obtained his Master and Bachelor degree in Automation Engineering from Politecnico di Bari, Bari, Italy, in 2014 and 2011. He is currently a PhD candidate at the university of Twente, Enschede, The Netherlands, working on his project "Cooperative Adaptive Cruise Control over unreliable networks". His research interests are in cooperative control over unreliable network, graph theory and consensus.

Paolo Frasca received the Ph.D. degree in Mathematics for Engineering Sciences from Politecnico di Torino, Torino, Italy, in 2009. From 2013 to 2016, he has been an Assistant Professor at the University of Twente in Enschede, the Netherlands. Since October 2016 he is CNRS Researcher with GIPSA-lab, Grenoble, France. His research interests are in the theory of network systems and cyber-physical systems, with applications to robotic, sensor, infrastructural, and social networks.

Geert Heijenk is a full professor at the University of Twente, the Netherlands. Between 1995, when he obtained his Ph.D., and 2003, he was working for Ericsson EuroLab, Netherlands, leading a networking research group. Geert Heijenk is steering committee member of WWIC and IEEE VNC. He has been a visiting researcher at the University of Pennsylvania, Philadelphia, and a visiting professor at the University of California, Irvine, and INRIA, Rocquencourt. His area of research is wireless networks and mobility. He is particularly interested in architectures, algorithms, and protocols for cellular, ad-hoc, sensor, and vehicular networks.

Anton A. Stoorvogel received the M.Sc. degree in Mathematics from Leiden University in 1987 and the Ph.D. degree in Mathematics from Eindhoven University of Technology, the Netherlands in 1990. Currently, he is a professor in systems and control theory at the University of Twente, the Netherlands. Anton Stoorvogel is the author of five books and numerous articles. He is and has been on the editorial board of several journals.