PONTIFICIA UNIVERSIDAD CATOLICA DE CHILE
SCHOOL OF ENGINEERING

ALGORITHM DESIGN FOR THE DISTRIBUTED AVERAGE CONSENSUS PROBLEM OVER IOT ENVIRONMENTS

BORIS ENRIQUE ORÓSTICA NAVARRETE

Thesis submitted to the Office of Research and Graduate Studies
in partial fulfillment of the requirements for the degree of
Master of Science in Engineering

Advisor:
FELIPE NÚÑEZ RETAMAL

Santiago de Chile, October 2018

PONTIFICIA UNIVERSIDAD CATOLICA DE CHILE
SCHOOL OF ENGINEERING

ALGORITHM DESIGN FOR THE DISTRIBUTED AVERAGE CONSENSUS PROBLEM OVER IOT ENVIRONMENTS

BORIS ENRIQUE ORÓSTICA NAVARRETE

Members of the Committee:
FELIPE NÚÑEZ RETAMAL
ANDRÉS GUESALAGA MEISSNER
CRISTIAN DURÁN FAÚNDEZ
PEDRO GAZMURI SCHLEYER

Thesis submitted to the Office of Research and Graduate Studies
in partial fulfillment of the requirements for the degree of
Master of Science in Engineering

Santiago de Chile, October 2018

Gratefully to my parents, brother
and my sweet Mayerlaine

ACKNOWLEDGEMENTS

This project has been supported by the National Commission for Science and Technology Research of Chile (Conicyt) under Fondecyt grant 1161039.

I want to thank the Pontificia Universidad Católica de Chile for its support and the opportunity of growing as a professional during all these years.

I would like to express my complete gratitude to my advisor Felipe Núñez, for his guidance through this work and the teachings he has given me personally and professionally. I feel grateful for what he has shared with me during these years.

Also thanks to the members of the review committee for their participation in evaluating this work, and their comments and contributions during this investigation.

Finally, but most importantly, thanks to my parents Lorenzo and María, my brother Rodrigo and specially thanks to my sweet Mayerlaine who have given me support and encouragement during the process.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

As communication technologies have enlarged the set of devices with networking capabilities, a new conception of the Internet of Things (IoT) is emerging. With the incorporation of devices with advanced diagnosis and actuating capabilities, the IoT provides an appealing environment to control external processes using its sensing, actuating, and computational power. In this setting, consensus algorithms are an appealing alternative to support the operation of the IoT and to enable its potential as distributed control network. In particular, the problem of reaching a consensus to the average of some initial quantities is a challenging problem with potential applications in IoT environments. Although consensus algorithms are mature well studied strategies that naturally adjust to networks, their performance deteriorates when faced with phenomena such as stochastic delays, sequential transmissions and receptions, and unreliability in the information exchanging process; all pervasive in an IoT environment. In this work, a new algorithm for achieving average consensus over an IoT environment is designed. Theoretical analysis is developed in order to understand its working principles. Furthermore, the algorithm is coded as a protocol on real hardware and is extensively evaluated over a local low-scale network and over a public large-scale network. The algorithm is inspired by gossips and converges to the average in all the experiments over a real IoT environment facing non-ideal communication phenomena.

Keywords: Consensus Algorithms, Distributed Average Consensus, Internet of Things, Multi-agent Control, Distributed Control, Non-Homogeneus Matrix Products.

RESUMEN

A medida que las tecnologías de comunicación han ampliado el conjunto de dispositivos con capacidades de red, está surgiendo una nueva concepción de la Internet de las cosas (IoT). Con la incorporacin de dispositivos con diagnósticos avanzados y capacidades de actuación, el IoT proporciona un entorno atractivo para controlar procesos externos utilizando sus capacidades de detección, actuación y computación. En este contexto, los algoritmos de consenso son una alternativa atractiva para apoyar el funcionamiento del IoT y para habilitar su potencial como red de control distribuido. En particular, el problema de llegar a un consenso al promedio de algunas cantidades iniciales es un problema desafiante con potencial aplicaciones en el entorno IoT. Aunque los algoritmos de consenso son estrategias maduras y bien estudiadas que se ajustan naturalmente a las redes, su desempeño se deteriora cuando se enfrentan a fenómenos tales como retrasos estocásticos, transmisiones y recepciones secuenciales y falta de fiabilidad en el proceso de intercambio de información; todo presente en un entorno de IoT. En este trabajo, se diseña un nuevo algoritmo para lograr el consenso al promedio en un entorno IoT. Se desarrolla un análisis teórico para comprender sus principios de funcionamiento. Además, el algoritmo está codificado como un protocolo en hardware real el cual se evalúa en una red local de baja escala y en una red pública de gran escala. El algoritmo está inspirado en gossips y converge al promedio en todos los experimentos realizados en un entorno real de IoT donde enfrenta las no idealidades de los fenómenos de comunicación.

Palabras Claves: Algoritmos de Consenso, Consenso al Promedio Distribuido, Internet de las Cosas, Control Multi-Agente, Control Distribuido, Producto No Homogéneo de Matrices.

Chapter 1. INTRODUCTION

1.1. Motivation

Since its origin, the Internet of Things (IoT) has been usually referred to as a large-scale network of heterogeneous objects with Internet connectivity, uniquely addressable using standard protocols, mainly IPv6 (Mukhopadhyay & Suryadevara, 2014). However, as communication technologies have enlarged the set of devices with networking capabilities up to objects as varied as keys, appliances, and cars, a new definition of the IoT is emerging as a sparsely coupled, distributed system of interacting smart objects, or things. Smart objects are able to sense/actuate, store, and interpret information created within themselves and around the neighboring external world where they are situated, take decisions, cooperate, and exchange information with other objects and human users (Fortino & Trunfio, 2014).

The change in the conceptual view of the Internet of things has also been partnered with an extension of its purpose. In its origin, the Internet of things was viewed as a data providing network that promotes the appearance of data-driven services (Bessis & Dobre, 2014; Borgia, 2014) in what is known as a cloud centric view (Gubbi, Buyya, Marusic, & Palaniswami, 2013), with an associated service-oriented layered architecture (Sarkar et al., 2015). However, with the incorporation to the network of devices with advanced diagnosis and actuating capabilities, the IoT provides an appealing environment to control external processes using its sensing, actuating, and computational power, more in the line of the control perspective of a cyber-physical system (Borgia, 2014). However, to be able to perform control tasks, several intrinsic operational issues must be solved first.

From a systems and control point of view, the IoT is a complex network that can be abstracted using classical tools such as graph theory (Biggs, 1993; Godsil & Royle, 2013) and hybrid systems (Goebel, Sanfelice, & Teel, 2009), which facilitates the synthesis of controllers to support its operation. Among the existing control techniques, distributed multi-agent control, in particular consensus algorithms, can be used to address several open problems in the operation of the IoT. Consensus algorithms are simple distributed protocols that

require only minimal computation and communication to reach a network-wide common value for locally observed variables. The agreement, or consensus, value is a function of the initial value at all the participating agents, with the specific function determined by the structure of the consensus protocol (Jadbabaie, Lin, & Morse, 2003; Saber & Murray, 2003; Xiao & Boyd, 2004; Cao, Morse, & Anderson, 2005; Moreau, 2005; Ren & Beard, 2005; Olfati-Saber, Fax, & Murray, 2007; Cao, Morse, & Anderson, 2008a, 2008b; Denantes, Benezit, Thiran, & Vetterli, 2008; Aysal, Yildiz, Sarwate, & Scaglione, 2009; Iutzeler, Ciblat, & Jakubowicz, 2012). The most popular formulation is the one ensuring convergence to the average of the initial conditions: the average consensus algorithm. Consensus algorithms have their origin in the analysis of Markov chains (Seneta, 2006) and have been applied since by the computer science community for load balancing (J. Tsitsiklis & Athans, 1984; Ghosh, Muthukrishnan, & Schultz, 1996) and by the linear algebra community for the asynchronous solution of linear systems (Frommer & Szyld, 2000; Strikwerda, 2002). They have been rediscovered and applied by the control, communications, and robotics communities (Jadbabaie et al., 2003; Olfati-Saber et al., 2007; Bullo, Cortés, & Martínez, 2009). Some typical applications includes mobile agents such as vehicle formation (Fax & Murray, 2002; Eren, Belhumeur, & Morse, 2002; Fax & Murray, 2004; Cao, Morse, Yu, Anderson, & Dasguvta, 2007) rendezvous to a common point (Lin, Morse, & Anderson, 2003; Cortes, Martinez, & Bullo, 2006; Lin, Morse, & Anderson, 2007a, 2007b) and flocking (Tanner, Jadbabaie, & Pappas, 2003; Olfati-Saber, 2006; Tanner, Jadbabaie, & Pappas, 2007; Cucker & Smale, 2007). Also, consensus has been applied in the context of sensor fusion (Gupta, Hassibi, & Murray, 2005; Olfati-Saber & Shamma, 2005; Spanos & Murray, 2005; Xiao, Boyd, & Lall, 2005), distributed Kalman filtering (Olfati-Saber, 2007; Carli, Chiuso, Schenato, & Zampieri, 2008) and even distributed optimization (Tsianos, Lawlor, & Rabbat, 2012a; Zhang & Kwok, 2014; Varagnolo, Zanella, Cenedese, Pillonetto, & Schenato, 2016).

Furthermore, the scientific community has tried to improve the performance of consensus and average consensus algorithms specifically by fastening the convergence speed. Classical analysis of this can be found in (Olfati-Saber, 2005; Kim & Mesbahi, 2006; Olshevsky & Tsitsiklis, 2006, 2011) and even more sophisticated strategies have been developed

(Kokiopoulou & Frossard, 2007; Johansson & Johansson, 2008; Aysal, Oreshkin, & Coates, 2009; Kokiopoulou & Frossard, 2009; Xiong & Kishore, 2009; Cavalcante & Mulgrew, 2010; Oreshkin, Coates, & Rabbat, 2010; Liu, Anderson, Cao, & Morse, 2013; Olshevsky, 2015).

As an example of the potential application of consensus algorithms to the IoT consider network-wide time synchronization, which can be regarded as a consensus problem over an heterogeneous network. Pulse-coupled synchronization (Schenato & Gamba, 2007; Wang, Núñez, & Doyle III, 2012; Núñez, Wang, & Doyle III, 2012; Nunez et al., 2017), a consensus-inspired technique which is naturally scalable and simple enough to be adopted by any smart object in the network, and Average TimeSynch (Schenato & Fiorentin, 2011), which is a double-consensus-based algorithm, have been applied with success in pilot-scale wireless sensor networks. Another example of a critical operational aspect is scheduling, where a set of agents compete for limited processing and networking resources (Longo, Su, Herrmann, & Barber, 2013). Scheduling can also be viewed as a consensus problem where agents agree, in a distributed manner, on processing and network access periods; a consensus-based solution has been used in (He, Duan, Hou, Cheng, & Chen, 2015) to schedule work modes in wireless sensor networks. It should be noted that in wireless sensor networks nodes usually have very similar hardware specifications, common communication requirements and a shared goal. While in IoT environments, the situation is drastically different. In fact, the following key factors have been recognized as challenges to deal with for any IoT research (Sarkar et al., 2015): heterogeneity, scalability, interoperability, security, and privacy.

Although consensus algorithms are mature well studied strategies that naturally adjust to networks, their performance relies heavily on strong assumptions such as perfect synchronization, instantaneous transmissions, concurrent updates, and nominally identical agent dynamics. Indeed, several works have been tried to address non-ideal phenomena such as packet losses (Fagnani & Zampieri, 2006; Patterson, Bamieh, & Abbadi, 2007; Kar & Moura, 2007a, 2009), communication delays (Olfati-Saber & Murray, 2004; Blondel, Hendrickx, Olshevsky, & Tsitsiklis, 2005; Sun, Wang, & Xie, 2008; Bliman & Ferrari-Trecate, 2008; Tsianos & Rabbat, 2011) and data quantization (Kashyap, Basar, & Srikant, 2006; Kar & Moura, 2007b; Frasca, Carli, Fagnani, & Zampieri, 2008; Nedic, Olshevsky, Ozdaglar, & Tsitsiklis, 2009;

Lavaei & Murray, 2012). Hence, the usefulness of consensus techniques in real large-scale complex networks is yet to be proven. In particular when the interaction between agents involves a full-stack communication network, which introduces stochastic delays, sequential transmissions and receptions, and unreliability in the information exchanging process. In fact, recent experiments in IoT testbeds (Oróstica & Núñez, 2017) have shown that classical average consensus formulations like the broadcast gossip (Aysal, Yildiz, & Scaglione, 2008; Aysal, Yildiz, et al., 2009) based on the gossip strategy (Boyd, Ghosh, Prabhakar, & Shah, 2005; Mehyar, Spanos, Pongsajapan, Low, & Murray, 2005; Boyd, Ghosh, Prabhakar, & Shah, 2006; Mehyar, Spanos, Pongsajapan, Low, & Murray, 2007; Bnzit, Blondel, Thiran, Tsitsiklis, & Vetterli, 2010; Dimakis, Kar, Moura, Rabbat, & Scaglione, 2010; Liu, Mou, Morse, Anderson, & Yu, 2011, 2018) and the push-sum algorithms (Kempe, Dobra, & Gehrke, 2003; Liu & Morse, 2012; Iutzeler, Ciblat, Hachem, & Jakubowicz, 2012; Tsianos, Lawlor, & Rabbat, 2012b; Iutzeler, Ciblat, & Hachem, 2013) deteriorate their performance when faced with phenomena such as asynchronous transmissions, heterogeneity, multiple transmission rates, and packet losses, all of which are present in an IoT environment.

In this work, the design of a new algorithm for achieving average consensus over an IoT environment is presented. The proposed strategy is inspired by gossips and is able to deal with several characteristic phenomena of the IoT as heterogeneity in terms of processing and networking, packet losses, and time varying communication topologies, among others.

1.2. Objectives, Contributions and Organization

The main objective of this work is to gain a deep understanding of the principles leading to average consensus in a multi-agent network subject to non-ideal communication phenomena, and to combine these principles to design a distributed algorithm capable of solving the average consensus problem over an IoT environment.

To achieve the main objective, several intermediate objectives must be met, among them conducting a deep bibliographic revision on distributed consensus and average consensus

problems is one of the first tasks to do. In parallel, the understanding of the non-ideal phenomena over an IoT environment must be gained in order to link the classical results with a future hardware implementation. At this point, it is worth performing an evaluation of classical average consensus algorithms in order to evidence their poor performance over real IoT environments due to non-ideal phenomena. To understand this poor performance, it is necessary to make a theoretical analysis of the system focusing on the non-ideal phenomena of asynchronicity and packet losses, which also can be used to properly design a new algorithm to solve the average consensus problem considering not just asynchronicity and packet losses, but also delays. Once the algorithm is designed, a theoretical convergence analysis is needed, in which asynchronous updates and packet losses will be regarded. Also, numerical simulations are performed in order to verify the theoretical reasoning with computational tools. After that, the proposed algorithm will be evaluated over real IoT environments to demonstrate the robustness even when all the pervasive non-ideal phenomena are present. Finally, a comparison with another state-of-the-art protocol is presented to put in context the designed algorithm with the latest results in the topic.

The thesis is organized as follows. In the remaining of the introduction chapter preliminaries about the notation, graph theory and infinite products are exposed.

The second chapter treats the framework and the problem under study. Here, the context of the problem is described and assumptions are clearly stated. It is worth emphasizing that these assumptions try to represent what actually happens in a real environment. Three non-ideal phenomena are considered in the communication process: asynchronous updates, packet losses and delays. Notions of the communication topology and asynchronous linear updates play a central role in modeling the dynamics of the system to later see how the information is distributed over the whole network. Also insights about the consensus and average consensus problem are discussed thinking on a future implementation over a real environment and finally the distributed average consensus is formally defined.

The third Chapter makes clear that the convergence behavior will depend directly on the infinite product of matrices, here the analysis is divided in the study of homogeneous and non-homogeneous matrix products.

The fourth Chapter analyzes the convergence of some existing algorithms with the tools shown in chapter three, an evaluation of the multi-cast protocols is conducted in both simulations and over a real hardware implementation in which, due to packet losses, the average consensus is not reached properly, thus some ideas from recent algorithms designed to face an unreliable communication channel are exposed.

In the fifth Chapter is presented the design of the a new algorithm based on previous ideas, the convergence analysis of the average consensus is done considering asynchronous updates and unreliable communication channels but not delays.

The sixth Chapter evaluates the algorithm over real IoT environments with excellent results, it is worth mentioning that in the real evaluation delays and other unfavorable phenomena are present and despite these disturbances the designed algorithm reaches the average; furthermore, a comparison with a push-sum based state-of-the-art protocol also is presented, in which our algorithm presents certain degree of superiority in terms of reliability to eventually faulty nodes.

Finally the final chapter closes the work by reviewing the insights developed throughout this manuscript and also giving some guidelines for future research on the distributed average consensus area.

1.3. Preliminaries

In this work, $\mathbb{R}$ denotes the real numbers, $\mathbb{R}_{\geq 0}$ the set of non-negative real numbers, $\mathbb{Z}_{\geq 0}$ the set of non-negative integers, $\mathbb{R}^n$ the Euclidean space of dimension n, and $\mathbb{R}^{n\times n}$ the set of $n \times n$ square matrices with real coefficients. I denotes the square identity matrix, $\mathbb{1}$ denotes the vector of all ones of appropriate dimension. The canonical vector which has a one entry in the i position and zero elsewhere is denoted as e_i. For a countable set χ, $|\chi|$ denotes its cardinality. For a complex number λ, $|\lambda|$ denotes its absolute value. Let A, B be matrices

or vectors of the same dimension, we will say that $A < B$ and $A \leq B$ if every entry of $(A - B)$ is strictly negative $(A - B) < 0$ and $(A - B) \leq 0$ respectively, similarly we will say that $A > B$ and $A \geq B$ if every entry of $(A - B)$ is strictly positive $(A - B) > 0$ and $(A - B) \geq 0$ respectively. The functions $\max\{.\}$ and $\min\{.\}$ whose argument can be a vector or a row-vector gives as a result the maximum of the minimum entry of the corresponding vector.

Let $A \in \mathbb{R}^{N \times N}$, λ_i denotes one of its N eigenvalues. The eigenvalues will be indexed according their magnitude, in particular the following inequalities always hold, $|\lambda_N| \leq \ldots \leq |\lambda_2| \leq |\lambda_1|$, where λ_1 is one of the eigenvalues with maximum absolute value. An eigenvalue λ is associated with an eigenvector v as long as:

$$Av = \lambda v \quad ,$$

since every eigenvector v multiplied by a real scalar value αv also gives an eigenvector associated with the eigenvalues λ, a subspace associated with an eigenvalue, which is called λ-eigenspace, can be introduced. In particular, we are interested in the 1-eigenspace of a matrix A, that represents all the fixed points of a matrix and is defined as:

$$E_1(A) = \{v \in \mathbb{R}^N : \, Av = v\}$$

More about matrix analysis can be found in (Horn, Horn, & Johnson, 1990).

Vector norms have the standard notation $\| \, . \, \|$. The p-norm is denoted by $\| \, . \, \|_p$. Correspondingly, induced matrix norms use the same notation. Additionally, every induced matrix norm used throughout this thesis is submultiplicative, i.e. $\| \, AB \, \| < \| \, A \, \| \| \, B \, \|$. Particular cases of induced p-norms are the following:

$$\begin{aligned} \| \, A \, \|_1 &= \max\{\mathbb{1}^T |A|\} \\ \| \, A \, \|_2 &= \sqrt{\lambda_1 \left(A^T A\right)} \\ \| \, A \, \|_\infty &= \max\{|A| \mathbb{1}\} \quad . \end{aligned}$$

In this thesis, infinity products of matrices taken from a finite set Σ will be treated. A finite set of matrices is denoted by Σ. Given a finite sequence of matrices $A_0,\ A_1,\ \ldots,\ A_{k-1}$ taken from Σ, with k a finite integer, the expression $(A_{k-1}\cdots A_1A_0)$ represents the finite product of length k of the particular finite sequence of matrices taken from Σ. We define Σ^k as the union of all finite products of length k of matrices taken from Σ. Analogously, given a infinite sequence of matrices $A_0,\ A_1,\ \ldots,\ A_{k-1},\ \ldots$ taken from Σ, the expression $(A_\infty\cdots A_1A_0)$ denotes the infinite product of the particular infinite sequence of matrices taken from Σ. We define Σ^∞ as the union of all infinite products of matrices taken from Σ. We say that a matrix A in Σ appears infinitely often in the infinite product $(A_\infty\cdots A_1A_0)$ if there is a sub-sequence of matrix products $A_0,\ A_1A_0,\ A_2A_1A_0,\ \ldots$ with a leftmost factor A, i.e., $AB_0,\ AB_1,\ AB_2,\ \ldots$, where the B_j's are products of A_k's. The infinite product $(A_\infty\cdots A_1A_0)$ converges if the sequence of products $A_0,\ A_1A_0,\ A_2A_1A_0,\ \ldots$ of matrices taken from Σ is a Cauchy sequence, note that convergence does not depend on the norm used. Σ is LCP if for any infinite sequence of matrices taken from Σ the infinite product $(A_\infty\cdots A_1A_0)$ converges. Σ is product bounded if there is a positive constant c such that $\parallel A_k\cdot A_1A_0\parallel\leq c$ for all k and all $A_0,\ A_1,\ \ldots,\ A_k\in\Sigma$, i.e. for any sequence of matrices the norm is upper bounded (regardless the norm used). Given a finite set of matrices Σ the joint spectral radius $\rho(\Sigma)$ is equal to the generalized spectral radius $\hat{\rho}(\Sigma)$ and they represent the supreme absolute value among all the eigenvalues taken from all the products of matrices taken from Σ (Berger & Wang, 1992). More about non-homogeneous matrix products can be found in (Hartfiel, 2002) and related to the joint spectral radius in (Jungers, 2009).

In a network with N nodes, the communication topology is modeled as a directed graph (or, for simplicity a graph) $\mathcal{G}=\{\mathcal{V},\mathcal{E},\mathcal{A}\}$, where $\mathcal{V}=\{1,\ldots,N\}$ is the node set; the arc set is defined as $(i,j)\in\mathcal{E}\subseteq\mathcal{V}\times\mathcal{V}$ if and only if node i receives information from the node j, which sometimes will be referred as path of length one $(i\leftarrow j)$ or $(j\rightarrow i)$; and $\mathcal{A}=[a_{ij}]\in\mathbb{R}_{\geq 0}^{N\times N}$ is the adjacency matrix, for which $a_{ij}=1$ if and only if $(i,j)\in\mathcal{E}$ and 0 elsewhere, note that $a_{ii}=1$ corresponds to the presence of a self-loop which means that a node receives information from itself. For a given node i, the set $\mathcal{V}_{in}^{(i)}$ denotes its in-neighbor set, i.e. all the nodes $j\in\mathcal{V}$ such that $(i\leftarrow j)$, and the set $\mathcal{V}_{out}^{(i)}$ denotes its out-neighbor set,

i.e. all the nodes $j \in \mathcal{V}$ such that $(i \to j)$. In the case when communications are bidirectional in the whole graph, i.e. if the arc $(i \leftarrow j)$ exists then the arc $(i \to j)$ also exists, we just say that a node i has a neighbor set $\mathcal{V}^{(i)}$. Further treatment about graph theory can be found in (Biggs, 1993; Godsil & Royle, 2013).

We say that there is a path to node i from node j if a sequence of arcs taken from $\mathcal{E}$ can be formed, such that $(i \to l_1)$, $(l_1 \to l_2)$, ..., $(l_{n-1} \to j)$, where l_1, l_2, ..., l_{n-1} are nodes in $\mathcal{V}$, in such case the path has a length n. In particular, the arc $(i \leftarrow j)$ is a path of length one. Some nodes in $\mathcal{V}$ from a graph G can be classified as:

- **Sink:** if there is a path of arbitrary length from every agent in the network to the sink node including itself.
- **Strong Sink:** if there is a path of length one from every agent in the network to the sink node including itself.
- **Root:** if there is a path of arbitrary length from the root node to every other agent including itself.
- **Strong Root:** if there is a path of length one from the root node to every other agent including itself.

Depending on the paths or the types of nodes, a graph can be classified as:

- **Strongly Connected:** if every node can receive information from all the agents in the network including from themselves with a path of arbitrary length (or, all the agents are sink nodes). This is equivalent to that every node can transmit information to all the agents in the network including to themselves with a path of arbitrary length (or, all the agents are root nodes).
- **Rooted:** if at least there is one node that can transmit information to all the agents in the network including to itself with a path of arbitrary length. Equivalently, a graph is rooted if at least has one root node.
- **Sinked:** if at least there is one node that can receive information from all the agents in the network including from itself with a path of arbitrary length. Equivalently, a graph is sinked if at least has one sink node.

- **Complete:** if every node receives information from all the agents in the network including from themselves with a path of length one (or, all the agents are strong sink nodes). This is equivalent to every node can transmit information to all the agents in the network including to themselves with a path of length one (or, all the agents are strong root nodes).
- **Strongly Rooted:** if at least there is one node that can transmit information to all the agents in the network including to itself with a path of length one. Equivalently, a graph is strongly rooted if at least has one strong root node.
- **Strongly Sinked:** if at least there is one node that can receive information from all the agents in the network including from itself with a path of length one. Equivalently, a graph is strongly sinked if at least has one strong sink node.

Dealing with a non-negative matrix A such that all the diagonal entries are positive, we will say that the matrix A induces an underlying graph such that it has an adjacency matrix $\gamma(A)$ which is simply another matrix such that all the positive entries of A are replaced by one and the zero entries of A remain zero in $\gamma(A)$. This underlying graph represents a directed graph $G = \{\mathcal{V}, \mathcal{E}, \mathcal{A}\}$ with the set of nodes $\mathcal{V} = \{1, \ldots, N\}$, the adjacency matrix $\mathcal{A} = \gamma(A)$ and the set of arcs $\mathcal{E} = \{(i, j) \in \mathcal{V} \times \mathcal{V} : \mathcal{A}_{ij} = 1\}$.

The same characterization of a graph can be done for the underlying graph of a non-negative matrix A with positive diagonal entries depending on the positive terms that appears in the powers of A:

- **Strongly Connected:** if $\exists k < \infty : \; A^k > 0$.
- **Rooted:** if $\exists k < \infty$, $\exists i \in \{1, \ldots, N\} : \; A^k \, e_i > 0$.
- **Sinked:** if $\exists k < \infty$, $\exists j \in \{1, \ldots, N\} : \; e_j^T \, A^k > 0$.
- **Complete:** if $A > 0$.
- **Strongly Rooted:** if $\exists i \in \{1, \ldots, N\} : \; A \, e_i > 0$
- **Strongly Sinked:** if $\exists j \in \{1, \ldots, N\} : \; e_j^T \, A > 0$

Given a finite set of matrices Σ such that every matrix is non-negative with positive diagonal entries, sometimes we will say that the infinite product $(A_\infty \cdots A_1 A_0)$ of matrices taken

from Σ can be separated into finite products of matrices whose underlying graph are complete/strongly rooted/strongly sinked. This means that the infinite product can be viewed as a infinite product of matrices which underlying graphs are complete/strongly rooted/strongly sinked:

$$(A_\infty \cdots A_1 A_0) \quad = \quad \cdots \underbrace{(A_{q-1} \ldots A_{p+1} A_p)}_{\gamma(.)=*} \underbrace{(A_{p-1} \ldots A_{k+1} A_k)}_{\gamma(.)=*} \underbrace{(A_{k-1} \ldots A_1 A_0)}_{\gamma(.)=*}$$

where $k < p < q$ and $\gamma(.) = *$ means that the underlying graph can be complete/strongly rooted/strongly sinked. More about products of non-negative matrices can be found in (Seneta, 2006).

Chapter 2. PROBLEM FORMULATION AND GENERAL FRAMEWORK

This chapter states the system under study. It is organized in four sections. The first section presents the communication dynamics, this includes assumptions over the topology, the concept of treating the agents as digital devices which leads to asynchrony, the use of internal clocks which implies the multi-rate behavior, the process of transmission-reception of information which in practical IoT scenarios includes packet losses and delays; although the theoretical analysis does not consider delays.

Since the theoretical framework does not include delays and assuming that the asynchronous updates are linear, the dynamics of the whole system can be seen as a linear time varying system in which the time step is not constant. This is presented in the second section. It will be clear that in order to represent asynchronous updates, the matrix matters and possesses an specific and limited structure related to the communication process from one node to another. Also desired properties of the matrices involved in the asynchronous updates are pointed out.

The third section deals with the system under the assumption that every agent only has one interval state and the coefficients of the linear updates are non-negative. Both assumptions make it possible to relate a matrix with an induced underlying graph, but not just that, the matrix generated as the result of products of matrices induced during a period of time also represents directly the underlying graph of communications performed synchronously, as if it just were one update. Additionally, this section reveals that if the feasible communication topology is strongly connected/rooted/sinked then the system will induce repeatedly complete/strongly rooted/strongly sinked underlying graphs for every sequence of matrix product under the multi-rate and packet losses behaviors, this property is the base to prove convergence of many linear distributed algorithms.

Finally, the fourth section makes a discussion about useful distributed objectives in a network of agents. It gives insights on the contrast between centralized and distributed solutions. It also discusses on one hand that the consensus problem, since the point of convergence does not matters, in practice can be solved with non-linear algorithms that can successfully overcome all the non-ideal phenomenons of asynchronous updates, unreliable communication

channels and delays. On the other hand, the average consensus problem can not be solved easily and it is challenging to design a rule to reach the average in a distributed way.

2.1. Assumptions on the Communication Dynamics

The first point of interest is the system under study. We are going to work with a group of agents that have to cooperate distributively in order to accomplish a specific task. In this work the task is to solve the distributed average consensus problem whose definition will be introduced soon, however first it is necessary to characterize the system where the distributed algorithm will be implemented.

In this work, an agent must be regarded as a digital device, for instance a computer or a micro-controller, which has storage and computation capabilities, can communicate with other agents and, in some other frameworks, can sense properties of the environment. One immediate consequence of using a digital device as an agent is that any change in the system state occurs when an action triggers the behavior of a node, that naturally can be due to an internal timer or when the node detects a message from another agent, therefore the variables stored in the memory of the nodes cannot change continuously, evidently they change asynchronously. Also, in order to not over complicate the problem, every time that an agent changes its internal variables, the update will be a linear function that depends on its own variables and other variables received from another node. These ideas are summarized in the following definition, which is one of the cornerstones of the mathematical models that will be presented soon:

Definition 2.1. *A digital device i, also called agent or node, is an abstract object that possesses internal variables, or internal states, grouped in a vector $x^{(i)}$ of eventually variable dimension. Any action performed by the agent, including the change of its internal states and transmission of them, happens as a consequence of a detection event. Only two types of events can be detected by node: a timer event or a data reception event. Also, every change of the internal variables of a node will be a linear function of its own states and the information received from data reception events.*

REMARK 2.1. *Note that the abstract notion of a digital device actually can be materialized using real hardware. For instance, every device in the IoT category such as microcontrollers with communication capabilities, microcomputers and computers are regarded as appealing digital devices to perform distributed algorithms.*

The aforementioned definition will be considered as the general model of an agent that can be deployed using real hardware, for example an IoT device, these ideas are depicted in Figure 2.1. Note that under definition 2.1, our framework is not considering the case when an agent senses states from other nodes, for instance with a position sensor, and the only way an agent can detect information from its environment is by means of data reception events. However, it will be analyzed soon a distributed algorithm in the context of the Vicsek's problem in which indeed every agent senses instantaneously the states of its neighbor. Even though this case is not within definition 2.1 of digital devices, its theoretical study gives important insights about the consensus problem.

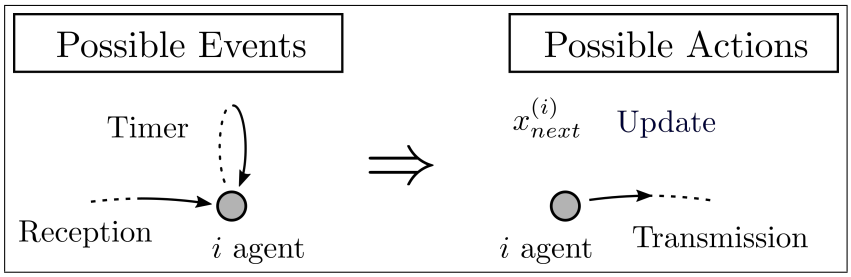

FIGURE 2.1. The actions of a digital device are triggered by events.

It is important to mention the process of transmission and reception of the information among the nodes because this restricts the system under analysis, in particular some results are not valid using one or another communication process and many times the analysis becomes easier if an unrealistic framework is used. In this work, we work with the idea of a feasible communication topology, an example is shown in Figure 2.2. Basically, a feasible communication topology can be described by a graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}, \mathcal{A}\}$, where $\mathcal{V}$ is the set of digital devices, $\mathcal{E}$ is the set of feasible communication links and $\mathcal{A}$ is its associated adjacency matrix. Note that qualifier "feasible" makes it explicit the idea that the communication

links can appear eventually during the dynamic of the system over time, i.e., the links are not present continuously, yet they only appear at certain times when a node transmits a message to another agent which detects an event of data reception. In the context of working over a network with agents which sometimes can communicate thanks to periodic internal clocks that make it possible transmissions of messages, the following assumption will be used throughout this manuscript:

ASSUMPTION 2.1. *The feasible communication topology $\mathcal{G}$ is given by external properties and processes, and its generation is out of the scope of this work, we consider it as given and static. Moreover, $\mathcal{G}$ has self-loops, since each agent knows its internal state, and is strongly connected. Also, but now dealing with connections over time, every communication link always must appear eventually, i.e., there is a finite time in which every arc in $\mathcal{E}$ appears during the evolution of the system.*

REMARK 2.2. *Assumption 2.1, which states that the feasible communication topology is given, is imposed to lighten the difficulty of the distributed average consensus problem that will be defined soon. However, making this is natural since the generation of a strongly connected feasible communication topology and reaching the average consensus can be regarded as separated problems. Also, note that since the information is being distributed repeatedly all over the network, assumption 2.1 imposes that the feasible communication topology is strongly connected and that every feasible link appears infinitely often during the evolution of the system.*

In the context of definition 2.1, the most suitable framework for the communication process in a network of agents would be the following: the generation of a message from a node is possible thanks to a timer event or a data reception event, the node j that desires to send a message must decide to whom to transmit the message, for example it could choose just one of its neighbors (unicast transmission) or can send information to all its neighbors (multicast transmission), as is depicted in Figure 2.3. Then if a neighbor i detects an event of data reception from agent j then it should perform a certain action. Thus, every time there is a

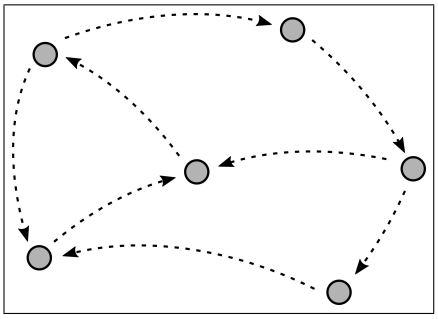

FIGURE 2.2. Example of a strongly connected feasible communication topology. The dotted arcs represent feasible links that will appear constantly during the deployment of a distributed task.

event over a node a communication process can occur and the sequence of all communication processes generates intricate dynamics all over the network.

Note that the agent which wants to transmit a message must know the receiver node or nodes. Therefore, it is necessary that every digital device must posses a unique identification in the network and every node needs to know the identification of its out-neighbors. This is stated in the following assumption:

ASSUMPTION 2.2. *Every agent $i \in \mathcal{V}$ knows its out-neighbor set $\mathcal{V}_{out}^{(i)}$. Also, at every reception event agent i knows which agent sent information to it, based on the addressing mechanism of the IoT environment.*

REMARK 2.3. *Note that, the knowledge of the out-neighbor sets follows from the generating process and network infrastructure mentioned in Assumption 2.1, which generates the feasible communication topology $\mathcal{G}$, e.g., a LAN with a DHCP server running on a router.*

Since we are working with digital devices, every action performed by a node is a consequence of the detection of an event, as depicted in Figure 2.1. In particular, internal timers are the original sources that trigger any action in the network. These clocks have frequencies that are not necessarily equal and, in a real deployment, rarely constant and likely out of phase. This property will generate what we call a multi-rate behavior. Note that as a consequence of the multi-rate and the communication process, any change in the state of any two different nodes rarely will occur at the same time and even if so, the change in their states

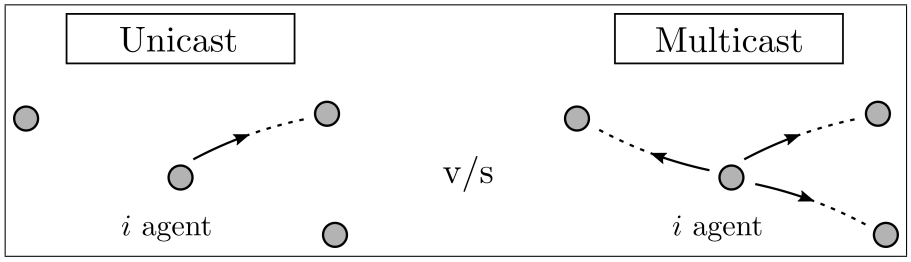

FIGURE 2.3. Two ways of sending information: unicast and multicast. Either way needs the unique identification of the agents and the knowledge of the out-neighbor set for every node.

can be regarded as sequential updates, i.e., if two or more nodes update their states instantaneously, then it can be viewed as consecutive updates. Therefore, is natural to assume that any change in the state is asynchronous, which means that every time there is an update just the state of one node changes. An identical behavior happens when receiving information, thus is improbable that multiple nodes detect events of data receptions exactly at the same time and even if so, the receptions can be regarded as asynchronous sequential events. In particular receptions from one source of multicast transmission probably will not reach its neighbors simultaneously due to delays as it is shown in Figure 2.4, this fact reveals that the communication process is fundamentally a transmission and a reception between two agents.

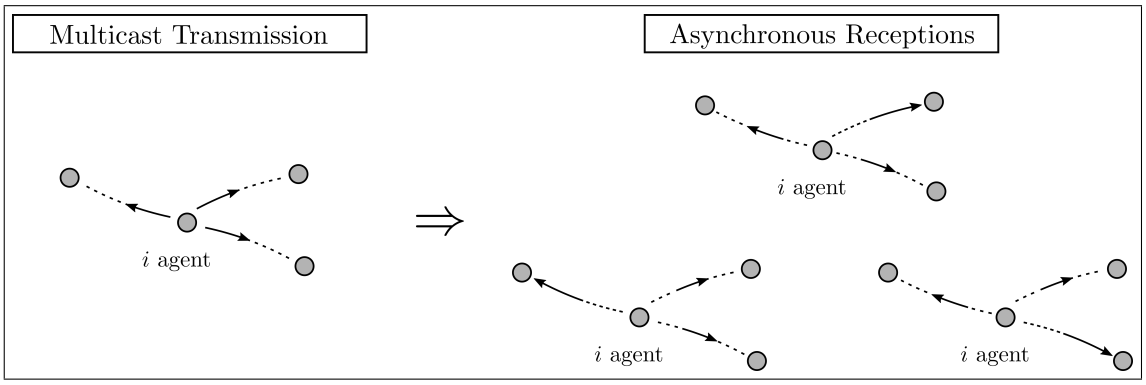

FIGURE 2.4. Events of data receptions are asynchronous. Even in the multicast transmission case the receptions are asynchronous due to delays.

Note that receiving information on a node is a delicate process because in general a digital device only can receive one message at a time and not multiple information from

other nodes. Even if that happens the processor should decide sequentially the processing of the information received, thus the reception of information from multiple sources to one agent are indeed asynchronous. A priori, the only way that a digital node can receive information almost simultaneously from multiple sources is by using own sensors, probably triggered by a timer, that detect information from its neighbors without waiting for transmission messages from them, for instance the Vicsek's problem assumes so (Vicsek, Czirók, Ben-Jacob, Cohen, & Shochet, 1995) along with some other findings (Jadbabaie et al., 2003; Cao et al., 2005, 2008a, 2008b). These ideas of receiving information are depicted in Figure 2.5.

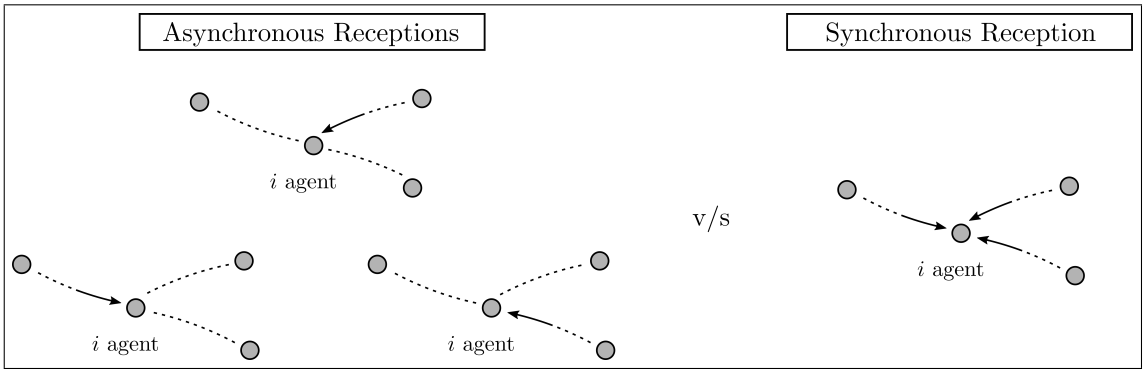

FIGURE 2.5. A node receiving information from multiple in-neighbors asynchronously and synchronously. The asynchronous case is the one that occur in a real implementation of communication via messages, the synchronous case only happens if a node can sense states from the environment, which will be analyzed because it gives important insights but is out the framework of this research.

The above discussion can be summarized in the following assumption:

ASSUMPTION 2.3. *Every detection event on a node is asynchronous which implies that at certain time just one agent in the whole network is performing an action.*

REMARK 2.4. *An immediate consequence of assumption 2.3 is that just one node can detect an event of data reception at a time and also just one node state can be updated at a time. Note that the asynchronous behavior is what actually occurs in a real IoT environment.*

In a real communication network there are some undesirable phenomena related to the transmission-reception process between a source agent and a sink node, among them the

most evident are: packet losses and delays. In the previous figures we have shown this idea implicitly by using dashed lines as communication links as is shown in Figure 2.6. We could talk about this extensively, but in practice the transmission-reception process works in a "best effort" way and there are some mechanisms that allow almost perfect communication in such a way that is just highly probable that a message can be received exactly once at the expense of more delay time. In order to privilege a shorter time and less communication congestion, we consider that there are no such mechanisms for perfect reception of messages transmitted and therefore packet losses is a phenomenon present in our framework. It is worth mentioning that a practical way of implementing fast transmission with possible losses of messages is by using the UDP protocol at the transport layer of an IoT stack. These ideas are stated in the following assumption:

ASSUMPTION 2.4. *The communication process is unreliable which means that can or cannot occur packet losses in the transmission-reception process. A packet loss is the phenomenon that happens when the data reception event associated with a transmission from another agent never occurs.*

REMARK 2.5. *Since the communication process is unreliable, the focus will be on the receiver instead of the transmitter. In particular, the design of any distributed algorithm over the network should regard that any update of the state of a node is a consequence of a reception event because otherwise (for instance, after a timer event) the change of the state would just consider own non-cooperative decisions since it would not listen any neighbor in the update. Therefore, timer events should just induce transmission actions and receptions events can generate both transmissions or state updates. However, this is just an advice to cope packet losses, because in general many distributed algorithms does not regard the unreliability phenomenon.*

In an IoT environment the communication process also will exhibit delays that are observed if the time interval Δt between a transmission and its associated event of data reception is a positive real value. The ideal case occurs when $\Delta t = 0$ for all transmissions and receptions and we will say that the communication channel is delay-free. In this work, the

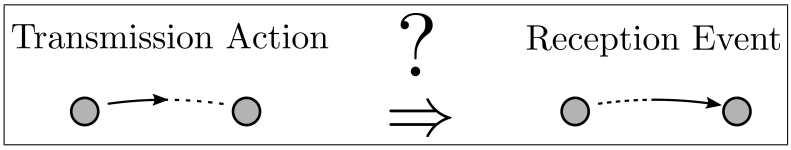

FIGURE 2.6. The communication process is unreliable and therefore only the receiver is aware of the message reception.

theoretical analysis will assume that the communication is delay-free, however the practical implementation of distributed algorithms will be conducted over a real IoT environment, which indeed presents communication delays. Regarding the delay-free theoretical assumption, we will draw complete filled arrows or complete dashed arrow in order to make explicit delay-free successful reception event or feasible communication link as depicted in Figure 2.7.

ASSUMPTION 2.5. *The theoretical analysis will assume that the communication process is delay-free which means that any transmission and its associated event of data reception is instantaneous.*

REMARK 2.6. *The main consequence of delay-free assumption on the theoretical analysis is that any update of a state of a node after a reception event which message contains the state of a in-neighbor regards indeed the current state without delays of both the transmitter and the receiver.*

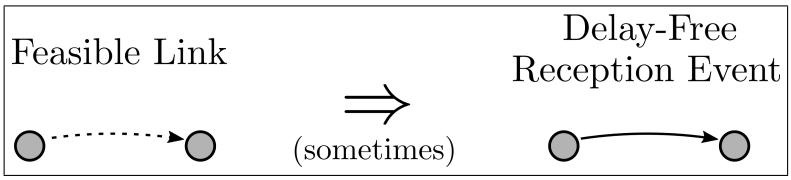

FIGURE 2.7. The communication process will be assumed delay-free in the theoretical framework. Regarding assumption 2.1, the feasible communication link is always appearing as the system evolves over time.

One could think that the above assumption is too strong since delay is a characteristic phenomena in wireless communications or in any real communication network; however, many findings do not treat this problem for distributed algorithms, and neither the asynchronous and packet losses phenomena, mainly because the theoretical analysis becomes quite

tough. Fortunately, we will see that the way to face the packet losses problem (by using some extra variables that store all past information) also treats the delays problem at least experimentally in a real environment. Therefore, the theoretical analysis throughout this manuscript will focus on the treatment of asynchronous actions and packet losses phenomena without regarding delays. With this in mind, the feasible communication topology presents an intricate dynamics which is depicted in Figure 2.8. We will see soon that the delay-free assumption is required in order to make the next state of the whole network only dependent on the current state and if the update rule is a linear function then the following state of the system will be given by a transition matrix.

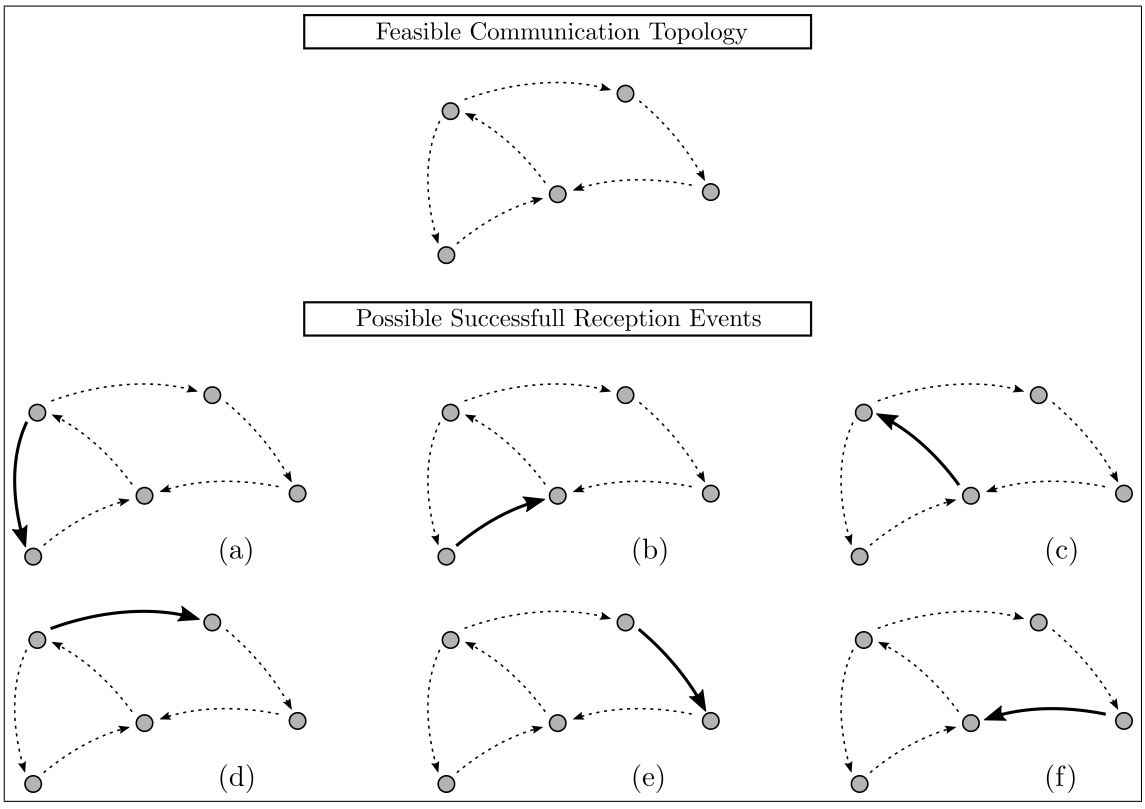

FIGURE 2.8. An example of a feasible communication topology and its possible asynchronous reception events. The dynamics of the communication links is intricate and the order in which they appear cannot be manipulated due to multi-rate behavior of the timers and packet losses, however we will assume that every communication link is always appearing during the evolution of the system which is reasonable as long as all the nodes are turned on and non-defective.

Apart from the non-ideal communication phenomenons, another undesirable problem is quantization of the data representation, which is an inevitable fact when we work with digital devices. However, we will not treat this issue in this work theoretically, and in practice we will evaluate the quantization phenomenon by testing the algorithm in a real IoT environment.

2.2. Linear Updates and Asynchronous Transition Matrices

In order to keep things simple but without losing the richness of the system dynamics, in a first approach we will analyze the situation when every agent just has one internal variable, or state, $x_k^{(i)} \in \mathbb{R}$, where $i \in \{1, \ldots, N\}$ is a label to index the agent and $k \in \mathbb{N}$ represents certain instant of time. Here, the state $x_k^{(i)}$ intuitively represents the estimated consensus value of the agent i at time k. If the agent receives information from its neighbors at the same time, recall that this is highly unlikely unless the agent senses the state of its neighbors without waiting for any message from them, a linear update of the state will be the following:

$$x_{k+1}^{(i)} = a_{ii} x_k^{(i)} + \sum_{j \in \mathcal{V}_{in}^{(i)}} a_{ij} x_k^{(j)},$$

where the $a_{ij} \in \mathbb{R}$ are the coefficients that weights the state of the involved agents which must be determined by a designer and $\mathcal{V}_{in}^{(i)}$ is the set of neighbors listened or sensed by the agent i. In the above expression it is natural to think that the $a_{ii} \neq 0$ since it means that the following state of the agent partly depends on its present state. Note that this case also can be possible by assuming that all the neighbors send information to the agent i at the same time and, as there are no delays, a unique event of data reception triggers the change of the state $x_{k+1}^{(i)}$, which is an unrealistic situation. If any other agent do not update its state, which is a reasonable assumption as it was stated in 2.3, the update of the whole state $x_k = [x_k^{(1)}, \ldots, x_k^{(N)}]$ can be calculated using a transition matrix, which has a diagonal with ones except in the row i of the updated agent in which every component is different from zero if the node i could receive information from some neighbors. It is important to highlight that in this framework the packet losses can be included since if the agent could not receive data from a neighbor j, then the coefficient related to that neighbor a_{ij} must be zero, which is

something that cannot be controlled by anyone and therefore the selection of the coefficients is critical. However, delay phenomena cannot be included in this analysis since otherwise the communicated states would not be the present states of the neighbors. To keep things clean, the following shows the update of the agent $i = 2$, which receives information from three neighbors $j = \{1, 3, 5\}$, where the number of agents are $N = 5$, the situation is also depicted in Figure 2.9, note that this example is inadequate under our framework since it does not represent an asynchronous update:

$$\begin{pmatrix} x_{k+1}^{(1)} \\ x_{k+1}^{(2)} \\ x_{k+1}^{(3)} \\ x_{k+1}^{(4)} \\ x_{k+1}^{(5)} \end{pmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ a_{21} & a_{22} & a_{23} & 0 & a_{25} \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} x_k^{(1)} \\ x_k^{(2)} \\ x_k^{(3)} \\ x_k^{(4)} \\ x_k^{(5)} \end{pmatrix}.$$

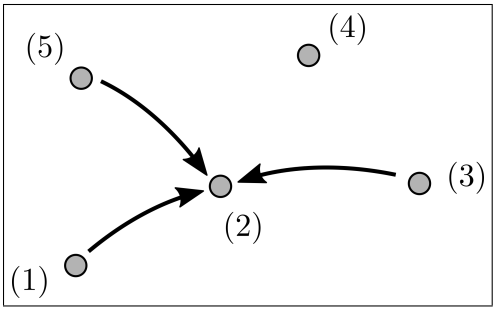

FIGURE 2.9. An example of synchronous multiple reception which naturally represents an update when there are not delays. The self-loop, which indicates that the next state of the node $i = 2$ depends on its present state was not added, neither the feasible links by simplicity. Since it is a synchronous update, it does not represent an adequate transition matrix under our framework.

As we mention before, if an agent i waits for data sent by a neighbor j it is unlikely that the update triggered by the event of data reception can be performed by using more than one neighbor, the natural update will be just a unique process of transmission from node j and reception from agent i, which implies that the update of the listener node state $x_{k+1}^{(i)}$ only depends on its own state $x_k^{(i)}$ and the state of the sender neighbor $x_k^{(j)}$. Therefore, the

asynchronous update will have the form:

$$x_{k+1}^{(i)} = a_{ii}x_k^{(i)} + a_{ij}x_k^{(j)},$$

and the transition matrix of the whole system will be a diagonal with ones (assuming that no other agent in the network update at that instant k) except in the row of the updated state in which there are two entries that can be different from zero a_{ii} and a_{ij}. Again, an example of this is depicted in Figure 2.10 with the following transition matrix where the listener agent is $i = 4$, the sender node $j = 1$ and the network is composed of $N = 5$ devices, this example represents an adequate update under our framework:

$$\begin{pmatrix} x_{k+1}^{(1)} \\ x_{k+1}^{(2)} \\ x_{k+1}^{(3)} \\ x_{k+1}^{(4)} \\ x_{k+1}^{(5)} \end{pmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ a_{41} & 0 & 0 & a_{44} & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} x_k^{(1)} \\ x_k^{(2)} \\ x_k^{(3)} \\ x_k^{(4)} \\ x_k^{(5)} \end{pmatrix}.$$

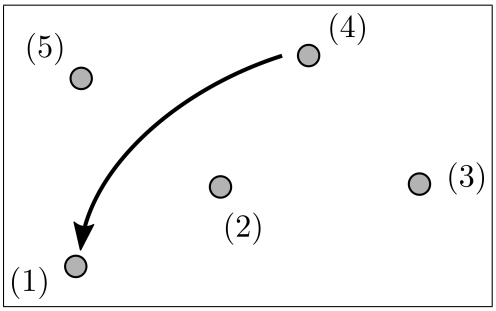

FIGURE 2.10. An example of asynchronous reception which induces and adequate asynchronous transition matrix in our framework. Just one node updates with its own state and the information listened from one neighbor.

The matrices with the form of the above example are the basic transition matrices and we will say that they represent the best way to model the updates of the system if we assume that the system is delay-free, the dimension of the state of each agent is one and the transmission-reception process are made by digital devices that waits to receive information. It is worth

mentioning that the asynchronous phenomena is reflected in the form of these matrices and the packet losses phenomena can be added to this framework.

Note that another possible asynchronous update valid in our framework could be the result of a non-cooperative decision after a timer event without listening any neighbor. If the agent i detects a timer event, then the linear update of its state would be:

$$x_{k+1}^{(i)} = a_{ii} x_k^{(i)},$$

and the asynchronous transition matrix of the whole system would be a diagonal with ones except in the a_{ii} entry. The Figure 2.11 shows an example in which agent $i = 3$ detects a timer event and updates its state, the asynchronous update will be the following:

$$\begin{pmatrix} x_{k+1}^{(1)} \\ x_{k+1}^{(2)} \\ x_{k+1}^{(3)} \\ x_{k+1}^{(4)} \\ x_{k+1}^{(5)} \end{pmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & a_{33} & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} x_k^{(1)} \\ x_k^{(2)} \\ x_k^{(3)} \\ x_k^{(4)} \\ x_k^{(5)} \end{pmatrix}.$$

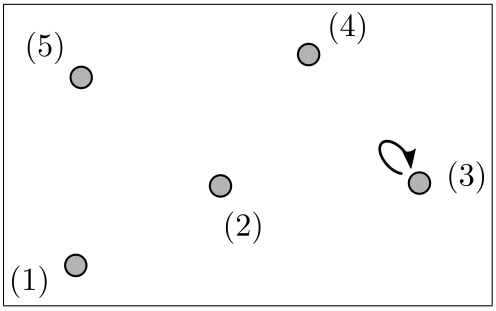

FIGURE 2.11. An example in which an agent updates its state based on its past state which is depicted as a self-loop. Even though it is an asynchronous update, this is not a advisable one since it does not regard information of the environment, i.e. it is a non-cooperative decision.

It is necessary to highlight that $a_{33} \neq 1$ is not a good value, since if we apply the same update infinitely keeping constant the a_{33} coefficient, i.e., the agent $i = 3$ is producing constantly a timer event without listening its neighbors (constant non-cooperative decisions),

then the state $x_k^{(3)}$ tends to zero if $|a_{33}| < 1$ or infinity if $|a_{33}| > 1$. It is clear that if a value diverges it is not practical at all for any algorithm, however stability to zero in many contexts is a desirable behavior. We will see that in the average consensus problem or in consensus in general reaching zero is a not good property and convergence to another vector is an appealing behavior, therefore we should try to avoid designs that allow the states decrease to zero. This analysis intuitively means that it is logical that if an agent does not have information of the network then it should not change its internal state and should not take a decision at all, but it should send its information to the rest of its neighbors. Therefore, updates that induces diagonal matrices should be the identity in order to avoid non-cooperative behavior.

In the previous discussion we stated that it is highly improbable that two agents update at the same time, therefore also is highly unlikely that several nodes update with the information sent by one agent, i.e., a multi-cast transmission. The difference with the aforementioned cases is that here the focus is on the agent which sends information. A multi-cast transmission can occur if after a timer event the sender node j, which may or may not update its state $x_k^{(j)}$, transmits its value to some neighbors which can perform proper updates and, if there are no delays, all the states change simultaneously, which is highly unlikely in a real environment due to delays. Thus, a transition matrix with a column with three o more entries different from zero cannot be valid in our framework since it represents synchronous updates. However, this can be viewed as consecutive asynchronous updates as long as is assumed a delay-free channel, and no other updates are performed. Just to illustrate, we show an inadequate matrix that performs a multi-cast transmission update on the state of the system that can be "asynchronized", as an example we take a network of $N = 5$ agents in which the agent $j = 4$ multi-casts its state, due to a timer event, and this is read by the neighbors $i = \{1, 5\}$. Note that if $a_{44} = 1$, i.e., the non-cooperation is avoided, then the order in which the matrices

appears does not matter and only adequate asynchronous matrices are induced.

$$\begin{pmatrix} x_{k+1}^{(1)} \\ x_{k+1}^{(2)} \\ x_{k+1}^{(3)} \\ x_{k+1}^{(4)} \\ x_{k+1}^{(5)} \end{pmatrix} = \begin{bmatrix} a_{11} & 0 & 0 & a_{14} & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & a_{44} & 0 \\ 0 & 0 & 0 & a_{54} & a_{55} \end{bmatrix} \begin{pmatrix} x_{k}^{(1)} \\ x_{k}^{(2)} \\ x_{k}^{(3)} \\ x_{k}^{(4)} \\ x_{k}^{(5)} \end{pmatrix}$$

$$\begin{pmatrix} x_{k+1}^{(1)} \\ x_{k+1}^{(2)} \\ x_{k+1}^{(3)} \\ x_{k+1}^{(4)} \\ x_{k+1}^{(5)} \end{pmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & a_{44} & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & a_{54} & a_{55} \end{bmatrix} \begin{bmatrix} a_{11} & 0 & 0 & a_{14} & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} x_{k}^{(1)} \\ x_{k}^{(2)} \\ x_{k}^{(3)} \\ x_{k}^{(4)} \\ x_{k}^{(5)} \end{pmatrix}$$

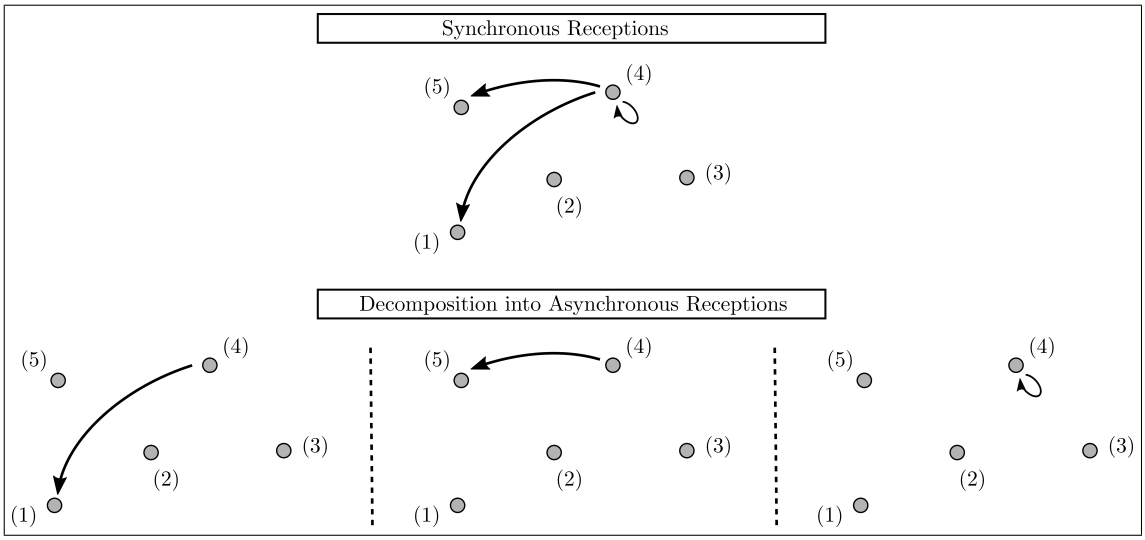

FIGURE 2.12. Example in which the synchronous updates of a multi-cast transmission can be decomposed into asynchronous receptions. The self-loop is not advisable since represents a non-cooperative update.

Analogously, there is an uni-cast version of the last process. If there is a timer event on a sender node j, first the next own state $x_{k+1}^{(j)}$ can be updated with the own internal state $x_k^{(j)}$ (not advisable) and then a message can be sent to one neighbor i, which detects an event of data transmission and can update its state $x_{k+1}^{(i)}$ by using its current state $x_k^{(i)}$ and the state of

the sender $x_k^{(j)}$. Therefore, a transition matrix of the system's state will be filled with ones on the diagonal excepting in the jth row in which the a_{jj} entry can be different from 1 and excepting in the ith row, in which there are two entries that can be different from zero a_{ii} and a_{ij}. Again, this matrix represents a synchronous update that can be decomposed into two asynchronous updates. A simple example just to clarify is shown in Figure 2.13 where $N = 5$, the node whose timer expires, can update its state and then send it is the agent $j = 2$, and the receiver is the node $i = 5$. The update of the system and its asynchronous version for the example are shown below:

$$\begin{pmatrix} x_{k+1}^{(1)} \\ x_{k+1}^{(2)} \\ x_{k+1}^{(3)} \\ x_{k+1}^{(4)} \\ x_{k+1}^{(5)} \end{pmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & a_{22} & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & a_{52} & 0 & 0 & a_{55} \end{bmatrix} \begin{pmatrix} x_k^{(1)} \\ x_k^{(2)} \\ x_k^{(3)} \\ x_k^{(4)} \\ x_k^{(5)} \end{pmatrix}$$

$$\begin{pmatrix} x_{k+1}^{(1)} \\ x_{k+1}^{(2)} \\ x_{k+1}^{(3)} \\ x_{k+1}^{(4)} \\ x_{k+1}^{(5)} \end{pmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & a_{22} & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & a_{52} & 0 & 0 & a_{55} \end{bmatrix} \begin{pmatrix} x_k^{(1)} \\ x_k^{(2)} \\ x_k^{(3)} \\ x_k^{(4)} \\ x_k^{(5)} \end{pmatrix}$$

Note that transition matrices with the form of the example above are almost the same as asynchronous reception matrices. Indeed, if we assume that the non-cooperative update due to the timer event is avoided, i.e., $a_{22} = 1$, then the update in the uni-cast transmission process induces an asynchronous reception matrix which is adequate for our framework as it was previously discussed.

We have just discussed that the asynchronous reception matrix is the way that adequately models an update of the network, what would happen if in this situation the message from the sender node j was not received by the receiver node i, i.e., a packet loss, the answer is nothing, because the agent i that should have updated its state could not detect the event of

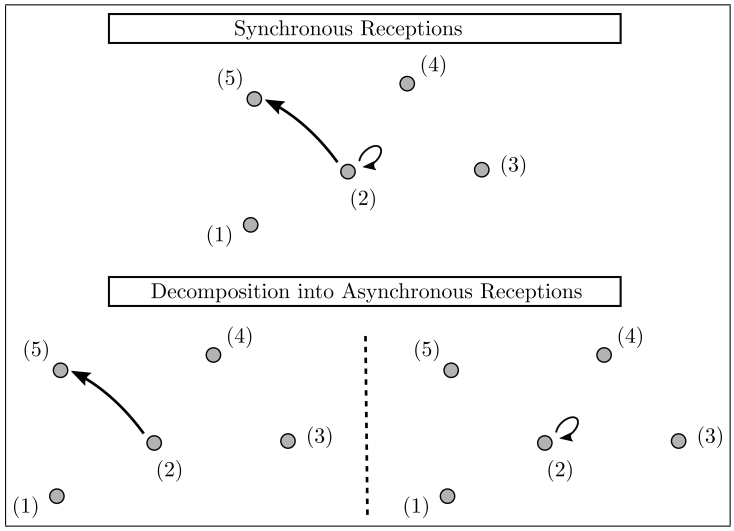

FIGURE 2.13. Example in which the synchronous updates of a uni-cast transmission can be decomposed into asynchronous receptions: an update due to a timer event an another as a consequence of a reception event. The update due to a timer event depicted as a self-loop is not advisable since represents a non-cooperative decision.

data reception and therefore it could not update the state, which is equivalent to thinking that the transition matrix is the identity, and, although there was a packet loss, the system does not gain or lose information. Thus, the system is robust to packet losses in the sense that does not change values of the state, however this phenomenon indeed can slow down the convergence rate of the system. An example of a successful reception contrasted with a packet loss is depicted in Figure 2.14 and its respective updates are shown below:

$$\begin{pmatrix} x_{k+1}^{(1)} \\ x_{k+1}^{(2)} \\ x_{k+1}^{(3)} \\ x_{k+1}^{(4)} \\ x_{k+1}^{(5)} \end{pmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ a_{41} & 0 & 0 & a_{44} & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} x_{k}^{(1)} \\ x_{k}^{(2)} \\ x_{k}^{(3)} \\ x_{k}^{(4)} \\ x_{k}^{(5)} \end{pmatrix} \quad v/s \quad \begin{pmatrix} x_{k+1}^{(1)} \\ x_{k+1}^{(2)} \\ x_{k+1}^{(3)} \\ x_{k+1}^{(4)} \\ x_{k+1}^{(5)} \end{pmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} x_{k}^{(1)} \\ x_{k}^{(2)} \\ x_{k}^{(3)} \\ x_{k}^{(4)} \\ x_{k}^{(5)} \end{pmatrix}.$$

The lesson learned here is that since the communication network is affected by packet losses and the transmission-reception process is conducted by digital devices, then not every matrix is valid or adequate to represent the change of the system's state. In particular, one

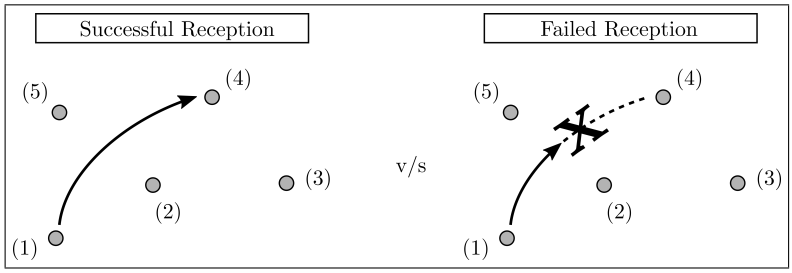

FIGURE 2.14. Example of a uni-cast transmission process in which the reception is successful of failed due to packet loss phenomenon. Note that since the timer event does not induces an update on the sender, the packet loss do not affect the state of the system and therefore there are some degree of robustness.

could naively think that the whole state denoted by $x_k = [x_k^{(1)}, \ldots, x_k^{(N)}]^T$ could be updated by a transition matrix dependent on the time instant A_k in the following way:

$$x_{k+1} = A_k \, x_k,$$

however, not every matrix is a valid one in order to represent what actually happens in a realistic transmission-reception process or desirable to keep good convergence properties. Notwithstanding the aforementioned, this is what the literature extensively does, which is a wrong perspective if the system is asynchronous, but reasonable if all the agents perform changes synchronously for example by sharing the same clock. We will see soon that the analysis for synchronous updates gives insights on convergence properties for algorithms that try to solve the distributed consensus and average consensus problem.

Furthermore, in the previous analysis we saw that even though timer events and data reception events could change the state of a node, just the latter is adequate to update the state. Changes due to timer events without having information of the network it is just an autonomous decision and have undesirable convergence behavior if these changes occur repetitively often, regardless whether there is a reliable communication transmission-reception process or not. In the latter case the situation is even worse because the more selfish updates without listening to the network, the more non-cooperative (non-distributed) results will yield the algorithm. Note that this can happen as the result of the multi-rate behavior, i.e., internal clocks triggers with different frequencies timer events on the agents, or due to packet losses

since in both cases selfish decisions could be taken more of less frequently by the agents. Thus, changes in the internal state of an agent should happen when events of data reception are detected, which naturally would make the updates non-dependent of the multi-rate and packet losses phenomena. Just to generalize, we will regard that the adequate updates will have the following form considering that the node i detects an event of data reception from the agent j:

$$\begin{pmatrix} x_{k+1}^{(1)} \\ \vdots \\ x_{k+1}^{(i)} \\ \vdots \\ x_{k+1}^{(j)} \\ \vdots \\ x_{k+1}^{(N)} \end{pmatrix} = \begin{bmatrix} 1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \cdots & a_{ii} & \cdots & a_{ij} & \cdots & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 1 & \cdots & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & 1 \end{bmatrix} \begin{pmatrix} x_{k}^{(1)} \\ \vdots \\ x_{k}^{(i)} \\ \vdots \\ x_{k}^{(j)} \\ \vdots \\ x_{k}^{(N)} \end{pmatrix}. \tag{2.1}$$

Another appealing behavior is the following. Imagine that agent i receives the exact same information from sender agent j twice or more, one next to the other. Once the first state change occurs it should be reasonable that the following update do not represent any change because there is not new information. This is equivalent to imposing that the induced transition matrix, let us say A, is an oblique projection, i.e., $A^2 = A$. This behavior also makes the algorithm more robust to packet losses and multi-rate which is shown in the following expression.

$$x_{k+2} = A\, x_{k+1} = A\, A\, x_k = A\, x_k$$

It is important to highlight that even though this analysis was made assuming that every agent has only one internal state, the intuition of what should happen in the iteration matrix when every agent has an state of larger dimension is the same as before, i.e., timer events should not update states, just transmit messages and changes in the state occur only when a data reception event is detected. The difference in this case is that the analysis should be made thinking on "block matrices entries" that relates two agents. Again, in order to generalize, we

are thinking on adequate updates of the following form:

$$\begin{pmatrix} \vec{x}_{k+1}^{(1)} \\ \vdots \\ \vec{x}_{k+1}^{(i)} \\ \vdots \\ \vec{x}_{k+1}^{(j)} \\ \vdots \\ \vec{x}_{k+1}^{(N)} \end{pmatrix} = \begin{bmatrix} I & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \cdots & A_{ii} & \cdots & A_{ij} & \cdots & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & I & \cdots & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & I \end{bmatrix} \begin{pmatrix} \vec{x}_{k}^{(1)} \\ \vdots \\ \vec{x}_{k}^{(i)} \\ \vdots \\ \vec{x}_{k}^{(j)} \\ \vdots \\ \vec{x}_{k}^{(N)} \end{pmatrix}, \tag{2.2}$$

where the sender is agent j and the receiver is the node i, $\vec{x}_k^{(i)}$ is the is state of node i at instant k with possibly more than one component, the dimensions of each state among the agents are not necessarily the same, and I (identity), A_{ii}, A_{ij} are matrices with adequate dimensions.

2.3. Non-Negativeness and Underlying Graphs

From the previous discussion, if we regard that every agent i has a unique state $x_k^{(i)}$ at instant k and if the state of the network $x_k = [x_k^{(1)}, \ldots x_k^{(N)}]^T$ can change synchronously, then any transition matrix $A_k \in \mathbb{R}^{N\times N}$ can represent the change of the state as long as there are no delays:

$$x_{k+1} = A_k\, x_k$$

However, if we consider the framework where just one agent i updates its state at certain instant k when it senses information from its neighbors due to a timer event, which is not under our framework but will be analyzed throughout this work, the change of the state will be the following:

$$\begin{aligned} x_{k+1}^{(i)} &= a_{ii}x_k^i + \sum_{j\in\mathcal{V}_i^{in}} a_{ij}x_k^j \\ x_{k+1}^{(r)} &= x_k^{(r)}, \end{aligned}$$

which can be represented by a transition matrix with components different from zero in the row of the updated agent. For example, in a network with $N = 5$ nodes, a timer event occurs at node $i = 4$ and it senses the states of all the nodes:

$$\begin{pmatrix} x_{k+1}^{(1)} \\ x_{k+1}^{(2)} \\ x_{k+1}^{(3)} \\ x_{k+1}^{(4)} \\ x_{k+1}^{(5)} \end{pmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} x_k^{(1)} \\ x_k^{(2)} \\ x_k^{(3)} \\ x_k^{(4)} \\ x_k^{(5)} \end{pmatrix}.$$

Even though the above transition matrix represents an asynchronous update of the whole state, it is not an adequate one for a real network in which there are asynchronous transmission-reception processes where the updates are performed only when an agent i detects an event of data reception from another node j. In this new framework, it is not allowed that the nodes can sense its in-neighbors, thus when there is a timer event in the node j, it does not have information from its neighbors, so it is not a good idea to change its state However, agent j transmits its state to one of its out-neighbors i (uni-cast) or maybe many of them (multi-cast). When the message produces an event of data reception on the node i and assuming delay-free, the whole state changes as follows:

$$\begin{aligned} x_{k+1}^{(i)} &= a_{ii}x_k^i + a_{ij}x_k^j \\ x_{k+1}^{(r)} &= x_k^{(r)}, \end{aligned}$$

which again can be represented by a transition matrix. For instance, in a network with $N = 5$ nodes, a timer event occurs at node $j = 5$ and it sends information to agent $i = 2$, then the

update of the network state is:

$$\begin{pmatrix} x_{k+1}^{(1)} \\ x_{k+1}^{(2)} \\ x_{k+1}^{(3)} \\ x_{k+1}^{(4)} \\ x_{k+1}^{(5)} \end{pmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & a_{22} & 0 & 0 & a_{25} \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} x_{k}^{(1)} \\ x_{k}^{(2)} \\ x_{k}^{(3)} \\ x_{k}^{(4)} \\ x_{k}^{(5)} \end{pmatrix},$$

and these matrices are the ones to work with in our framework. They represent asynchronous changes of the state when there are no delays. Thus, when we work with changes of the state the model will be a linear time varying system of the form:

$$x_{k+1} = A_k \, x_k,$$

however, not every matrix is a valid one. Only the matrices that represents asynchronous updates of the form in equation 2.1 are adequate to represent our framework.

It is clear that when an update matrix appears, there is a natural relationship with the communication topology. In fact, if agent i updates its state due to a data reception event produced by agent j, then the arc $i \leftarrow j$ appears in the network topology at that time, which is directly related to the non-zero entry a_{ij} in the iteration matrix. Note that, the arc is not always present, it appears only when a data reception event happens. Also note that with this logic it is clear that the a_{ii} entry should be different from zero for every agent i, as long as the update of the agent depends on its own state, thus at every time instant k an agent knows its state regardless whether or not detects an event of data reception and if it receives information and its updated state regards its past state, then in the communication topology every time there is an update self-loops are present, $i \leftarrow i$, for every node i.

To represent the data transferred directly from one node to another, we regard the set of non-negative matrices $A \geq 0$ such that the diagonal entries are positive $diag(A) > 0$. Note that this set of matrices is closed under multiplication. Also, it is clear that the entries different from zero of such matrices are related to the links formed in order to update the state of a node, in particular if $a_{ij} > 0$, then the updated state of the i agent $x_{k+1}^{(i)}$ at the instant

$k+1$ is the result of incorporating the state of the j agent $x_k^{(j)}$ at time k, this means that there is a link $i \leftarrow j$ of length 1 in the network topology at instant k. Note that we just want to represent whether or not there is a communication link at certain instant then the value of the coefficient a_{ij} is not relevant, therefore if $a_{ij} > 0$ at certain instant then there is a link $i \leftarrow j$ of length one and if $a_{ij} = 0$ then there is no link at all. Therefore, for any non-negative matrix $A \geq 0$ with positive diagonal $diag(A) > 0$ we can associate a communication topology or an underlying graph $\gamma(A)$ replacing all the positive entries by one and holding the zero entries and then draw the nodes of the network and the corresponding links.

Note that if the transition matrices of the state are non-negative with positive diagonal entries, then consecutive updates associated with consecutive products of matrices show up the equivalent underlying graph with links of length one. To make this claim clearer, let us regard an example of $N = 4$ nodes, and consider the following strongly connected feasible communication topology depicted in Figure 2.15, i.e., the links which are possible to appear after certain finite time which are induced thanks to the internal timers of the digital devices as long as the reception is successful:

$$\mathcal{A} = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}.$$

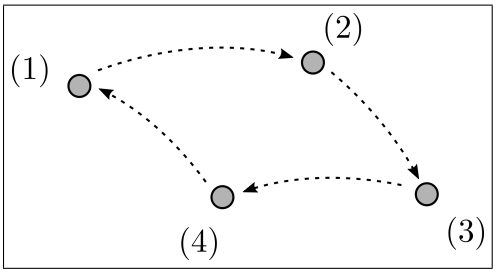

FIGURE 2.15. Example of a strongly connected feasible communication topology where the self-loops are omitted by simplicity.

Regard that the finite set of four asynchronous update matrices such that $A \geq 0$ and $diag(A) > 0$ which are induced by the algorithm every time a timer triggers on j with its respective data reception event on i:

$$A^{(21)} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ a_{21} & a_{22} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad A^{(32)} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & a_{32} & a_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

$$A^{(43)} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & a_{43} & a_{44} \end{bmatrix}, \quad A^{(14)} = \begin{bmatrix} a_{11} & 0 & 0 & a_{14} \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

these matrices not only represent asynchronous updates at certain time instants, but also represent communication links of length one of the underlying graphs shown in Figure 2.16.

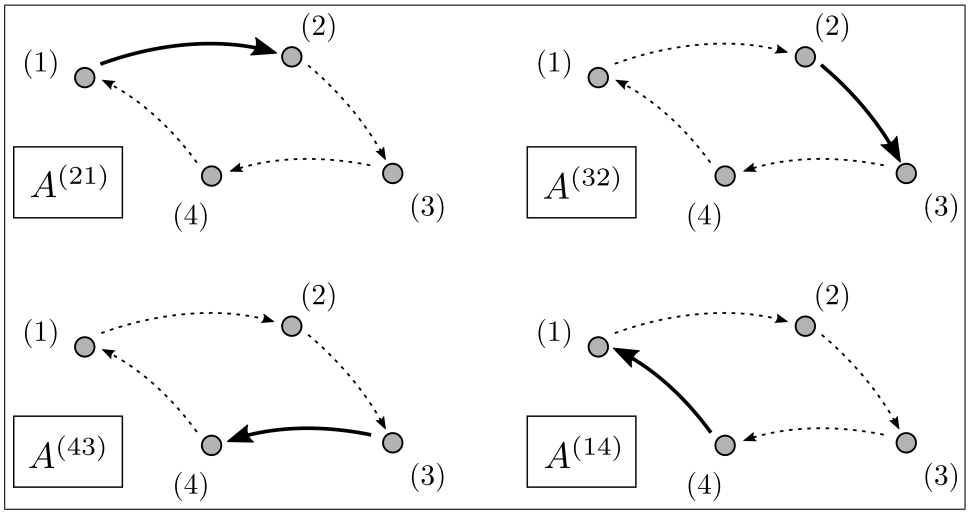

FIGURE 2.16. Example of different asynchronous underlying graphs induced by asynchronous transition matrices.

Now let us say that at time k agent 3 receives information from node 2 ($3 \leftarrow 2$), and after that at time $k+1$ agent 2 receives information from node 1 ($2 \leftarrow 1$), then the state at $k+2$

would be:

$$x_{k+2} = \left(A_{k+1}^{(21)} A_k^{(32)}\right) x_k$$

$$\begin{pmatrix} x_{k+2}^{(1)} \\ x_{k+2}^{(2)} \\ x_{k+2}^{(3)} \\ x_{k+2}^{(4)} \end{pmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ a_{21} & a_{22} & 0 & 0 \\ 0 & a_{32} & a_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} x_k^{(1)} \\ x_k^{(2)} \\ x_k^{(3)} \\ x_k^{(4)} \end{pmatrix},$$

and then a new underlying graph was induced by the algorithm after two asynchronous

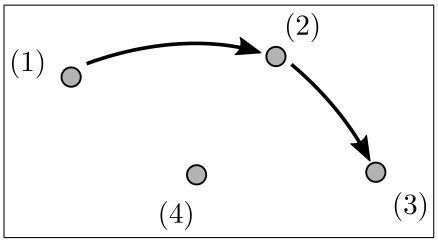

FIGURE 2.17. Example in which two consecutive asynchronous updates can be viewed as one synchronous update whose underlying graph possesses two links of the feasible communication topology.

updates shown in Figure 2.17. The important point here is that the resultant underlying graph can be viewed as one synchronous change after two updates even though the two original updates were asynchronous. But now, if we regard that the first update at time k is performed when the agent 2 receives information from node 1 ($2 \leftarrow 1$), and after that at time $k+1$ agent 3 receives information from the node 2 ($3 \leftarrow 2$), then the state at $k+2$ would be:

$$x_{k+2} = \left(A_{k+1}^{(32)} A_k^{(21)}\right) x_k$$

$$\begin{pmatrix} x_{k+2}^{(1)} \\ x_{k+2}^{(2)} \\ x_{k+2}^{(3)} \\ x_{k+2}^{(4)} \end{pmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ a_{21} & a_{22} & 0 & 0 \\ a_{32}a_{21} & a_{32}a_{22} & a_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} x_k^{(1)} \\ x_k^{(2)} \\ x_k^{(3)} \\ x_k^{(4)} \end{pmatrix},$$

and again the resulting underlying graph can be viewed as one synchronous change after two asynchronous updates shown in Figure 2.18. The difference in this case is that the link of length one $3 \leftarrow 1$ appeared, which was not in the feasible links of the original feasible

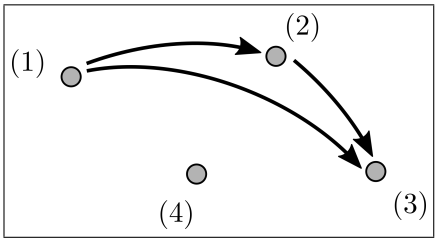

FIGURE 2.18. Example in which two consecutive asynchronous updates can be viewed as one synchronous update whose underlying graph possesses two links of the feasible communication topology and one additional link not in the feasible topology.

communication topology. Thus, products of non-negative matrices with positive diagonal induce new underlying graphs that even can generate more links of length one than the original feasible links.

Now, let us make the following complete round of updates:

$$x_{k+4} = \left(A_{k+3}^{(21)} A_{k+2}^{(32)} A_{k+1}^{(43)} A_{k}^{(14)}\right) x_k$$

$$\begin{pmatrix} x_{k+2}^{(1)} \\ x_{k+2}^{(2)} \\ x_{k+2}^{(3)} \\ x_{k+2}^{(4)} \end{pmatrix} = \begin{bmatrix} a_{11} & 0 & 0 & a_{14} \\ a_{21}a_{11} & a_{22} & 0 & a_{21}a_{14} \\ 0 & a_{32} & a_{33} & 0 \\ 0 & 0 & a_{43} & a_{44} \end{bmatrix} \begin{pmatrix} x_{k}^{(1)} \\ x_{k}^{(2)} \\ x_{k}^{(3)} \\ x_{k}^{(4)} \end{pmatrix},$$

again after four updates the algorithm induces synchronous communication links of length

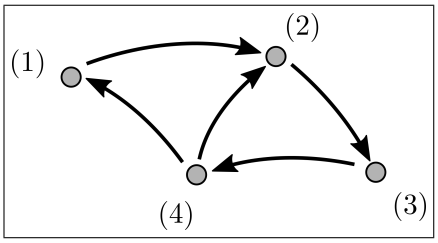

FIGURE 2.19. Example in which all the asynchronous updates of a strongly connected feasible communication topology appears which can be viewed as a synchronous update. Note that a new link not presented in the feasible topology appeared in the equivalent synchronous underlying graph.

one (see Figure 2.19) and furthermore a new arc $2 \leftarrow 4$ appears although it was not present in the feasible communication topology. Note that if at instant $k + 5$ the algorithm induces

the $A^{(21)}$ it will not induce new links but the former ones will not disappear either. Also note that any of the remaining asynchronous updates $A^{(32)}$, $A^{(43)}$, $A^{(14)}$ will add at least a new synchronous link of length one not present in the feasible communication topology. Naturally, if each one of these matrices appears eventually, after a certain finite time every entry of the induced synchronous matrix will be different from zero as long as the feasible communication topology is strongly connected.

The previous discussion can be generalized, and thus for any product of asynchronous iteration matrices non-negative and with positive diagonal entries, the induced synchronous underlying graph will include the links presented in each one of the synchronous matrices involved in the product (which also are presented in the feasible communication graph), but that is not all, also new links of length one not present in the feasible communication links may appear. Furthermore, as long as the algorithm induces eventually each one of the asynchronous matrices and if the feasible communication topology is strongly connected then the algorithm will induce a synchronous update matrix in finite time whose entries are all positive, i.e, complete underlying graphs. As the generation of complete underlying graphs is in finite time, this implies that the updates induce repeatedly complete graphs as the system evolves.

Similarly, if the feasible communication topology has at least one root/sink and if the asynchronous updates non-negative and with positive diagonal entries appears infinitely often, then in finite time synchronous updates will be induced such that the equivalent underlying graph is strongly rooted/sinked and therefore the algorithm induces repeatedly rooted/ sinked graphs. It is proposed to the reader to analyze these ideas with the feasible communication topologies in Figure 2.20 with the following adjacency matrices:

$$\mathcal{A}_{rooted} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \end{bmatrix} \quad , \quad \mathcal{A}_{sinked} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$

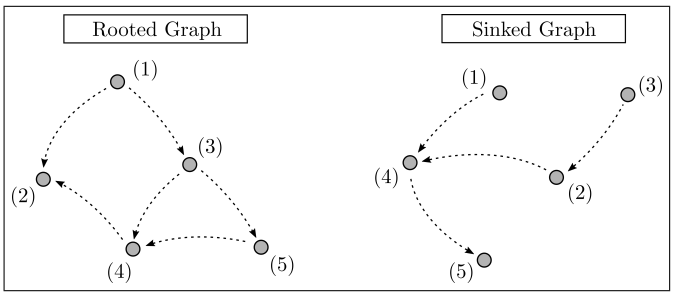

FIGURE 2.20. Example of feasible communication topologies rooted and sinked. The self-loops are omitted.

Note that the definition of the underlying graph could be extended to every transition matrix with non-zero diagonal entries without the non-negativeness restriction, in this case we could replace the non-zero elements by one and holding the zero entries and then draw the corresponding topology, however in this case a zero entry can mean both the non-presence of a link or multiple communications that as result gives a zero link, and therefore the analysis in which the links of the asynchronous matrices do not disappear after a certain number of products of matrices is not valid anymore and thus the resulting products of asynchronous updates induced by the algorithm whose feasible communication topology is strongly connected/rooted/sinked would not necessarily imply that the underlying graph is complete/strongly rooted/strongly sinked in finite time.

By defining that a non-negative matrix with positive diagonal entries A is generated from a feasible communication topology $\mathcal{G}$ as the matrix such that its underlying graph $\gamma(A)$ contains positive entries where the adjacency matrix of the feasible communication topology $\mathcal{A}$ contains 1 values but some of the entries of A can be zero where the adjacency matrix is 1, the discussed ideas for asynchronous matrices are summarized in the following theorem:

Theorem 2.1. *Consider the finite set of all the asynchronous matrices Σ of the form 2.1 non-negative and with positive diagonal entries generated from a feasible communication topology $\mathcal{G}$. Also assume that in the infinite product $(A_\infty \cdots A_1 A_0)$ of matrices taken from Σ every matrix is appearing infinitely often. If the feasible communication topology is strongly*

connected/rooted/sinked then the infinite product always can be separated by finite products of matrices whose underlying graphs are complete/strongly rooted/strongly sinked.

Analogously, in the context of non-homogeneous products of non-negative matrices with positive diagonal entries also we are interested in synchronous matrices. In this case we will define that a finite set of matrices Σ is generated from a feasible communication topology $\mathcal{G}$ if it satisfies that every matrix A in Σ is taken from the feasible topology $\mathcal{G}$ and the sum of all the matrices in Σ gives an underlying graph equal to the adjacency matrix of the feasible communication topology. With this definition, it is clear that the following theorem holds, which represents an extension of theorem 2.1:

Theorem 2.2. *Consider a finite set of matrices Σ non-negative and with positive diagonal entries which represent synchronous updates such that Σ is generated from a feasible communication topology. Also assume that in the infinite product $(A_\infty \cdots A_1 A_0)$ of matrices taken from Σ every matrix is appearing infinitely often. If the feasible communication topology is strongly connected/rooted/sinked then the infinite product always can be separated by finite products of matrices whose underlying graphs are complete/strongly rooted/strongly sinked.*

2.4. Practical Consensus and Average Consensus Goals

In this chapter we have presented the framework we will work with. Basically, we have a set of digital devices with an internal timer, possibly with different frequencies, that can transmit and receive information to/from its neighbors, and under the unfriendly phenomenons of asynchronous updates and packet losses (delays will not be regarded in the theoretical analysis, they will just be considered in the design of our new algorithm) they have to perform distributively a goal. However, we have barely talked about the objective of the system. Therefore, let us think about what would be good distributed goals. But before that, what do we mean when we say that the algorithm is a distributed one?.

Reaching an objective in a distributed way means that the goal must be the result of the cooperation among the agents involved without the use of a central entity. Behaviors like a

node receiving information from every agent in the whole network are not allowed since it involves a central entity. Therefore, communication with only a few neighbors is desired in a distributed algorithm. The distributed characteristic is not the same as decentralized since the latter does not have necessarily cooperation and can even have competition among the agents. A distributed algorithm in certain contexts is better than the centralized solution, for instance having equally the control in different nodes should be more robust than the central solution since in the case that a node disappears the network will continue working, in contrast to the centralized solution where the central entity is a single point of failure. Another example is information management, since the centralized solution has to store a large amount of data an then process it, which many times is impossible to do; however, distributed solutions are better because each agent could store an process just a few amount of data, but probably the hardness of the implementation will be higher. In this work we are interested in reaching a goal by using distributed algorithms, they are worth to be studied because it is a paradigm shift in contrast to the classical centralized solution.

Consensus is the first goal that shows up in mind when we talk about a distributed algorithm. The consensus objective considers that each agent on the network has a real value, different at the beginning, then they communicate among them in a distributed way and after a certain time every state has exactly the same value. Note that the consensus problem is not very interesting if it is faced with the centralized approach, because the solution is quite simple, first the central agent chooses a certain value and then it sends it to every node, nothing to analyze at all. However, the distributed linear approach has more richness in its dynamics, is not that simple. One of the firsts analysis was made with the Vicsek's problem (Vicsek et al., 1995) which consists in a group of mobile agents that are moving with certain heading or angle state and finally every node by sensing the headings of its neighbors and updating its states by a linear combination eventually reach the same state and move to the same direction. The situation is illustrated in the Figure 2.21. This is a really interesting behavior that was extensively studied, theoretically analyzed and tested using simulations. However, the context of the Vicsek's problem is different from an IoT environment in which the detection

of the information is by means of a transmission-reception process without sensing the states from the environment.

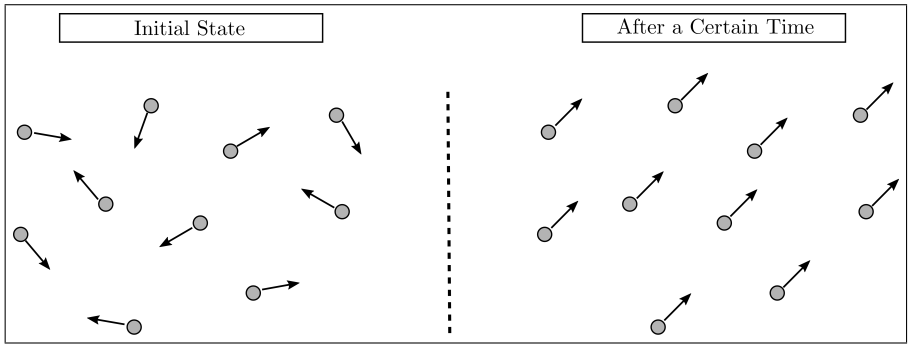

FIGURE 2.21. Illustration of the behavior in the Vicsek's Problem. At the beginning all the mobile agents have different headings, after a certain time all the headings point at the same direction, i.e. consensus is reached. The algorithm coded on each agent is linear and just regards information sensed from its neighbor. This algorithm cannot be implanted on our framework since the information is sensed and not obtained by a transmission-reception process.

According to my point of view, a key issue with the consensus problem is that the final state of the agents at the end just need to be equal but it does not care about what the final value is. For instance, if we want to reach the zero value (or any other), then it is not necessary to create any algorithm, it is enough to set the reference internally on each node and without any necessity of communication every agent will reach the desired zero value. Another more relevant example that regards initial conditions, if we want to calculate the maximum (or the minimum) of the initial states of the nodes a simple algorithm performed by each node would be the non-linear rule that if a node receives the state of another agent then it updates the state keeping the maximum (or the minimum):

$$x_{k+1}^{(i)} = \max\{x_k^{(i)}, x_k^{(j)}\},$$

naturally this rule will converge in finite time, the time of convergence depends on the largest path that the information can be transferred from every to each another in the network, it is even robust to packet losses, multi-rate, and even is intuitively resistant to delays since it works based on a "greedy" logic. Another example of consensus is to follow the state of a chosen leader (choosing a leader is a problem itself that could be solved for instance with the

aforementioned max-min algorithm), in this case every agent will get at the end the initial state of the leader node, in this case a natural rule would be that the leader does not listen to any state of other nodes and that the remaining agents just keep the state of the listened neighbor:

$$x_{k+1}^{(i)} = x_k^{(j)},$$

naturally, this rule will converge in finite time as we expected, again the time of convergence depends on the largest path, and, due to its "greedy" logic, it is robust to multi-rate, packet losses and delays, a property that is highly desirable in a real network and, in the design a our new algorithm, we will use a rule with a similar logic. Therefore, reaching consensus is not as challenging as we may think initially (clearly the above ideas should be analyzed in detail but the essential is already understood).

From the above examples the objective of consensus is desirable, however maybe it is better try to reach an specific value only dependent on the initial internal states, neither a random value nor a predefined point as zero nor a unique initial value which could be spread greedily on the whole network which beforehand can be solved satisfactorily over a really unfavorable communication environment without thinking too much.

And here appears the average consensus problem which consist in the group of agents reaching the average of the initial states in a distributed way. Note that average consensus is not an interesting problem if we do not add the distributed behavior. In fact, we can think in the centralized solution that would be to add a central entity, next recollect every initial state, then calculate the average and finally send this value back to every node in the network. Clearly this solution has the problem that there is a single point of failure and it has to process probably a large amount of information. Another non-distributed solution would be that every agent stores a table with each node in the network and then performs the average, which again is as if there were many central entities. The distributed average consensus is not as easy as you may think. Imagine for instance a group of people that only can access the information of their neighbors, what would be the rule to perform the average of the initial values of the whole system?, it is really a great question with a not easy answer and even more difficult if

the communication channel is unfriendly, and this is exactly the question that we will try to answer in this work.

The following definition states formally what we mean when the distributed average consensus problem is solved. Regard a set of non-defective nodes that can communicate under a strongly connected feasible communication topology. Also consider that every node i possesses at least one internal value $z_k^{(i)} \in \mathbb{R}$. Then:

Definition 2.2. *A distributed algorithm achieves average consensus if and only if:*

$$\lim_{k\to\infty} z_k^{(i)} = \frac{1}{N}\sum_{k=1}^{N} z_0^{(k)} \quad , \forall i \in \mathcal{V}.$$

Note that in the definition 2.2 an agent can store more than one internal variable, but at least one of its entries must represent the value of the estimated average $z_k^{(i)} \in \mathbb{R}$. This is pointed out because most of the examples where made considering that every node contains only one value, however the general case can include other variables (which can be added to a faster convergence response or in our case they will be added to cope the packet losses phenomenon). Also, even though the linear behavior is not a restriction on the distributed algorithm, we will work just with linear updates that makes the average consensus be reached geometrically fast.

Chapter 3. CONVERGENCE OF INFINITE MATRIX PRODUCTS

In this chapter we will understand that the stability analysis of the system to a consensus, including the average consensus, depends directly on an infinite non-homogeneous product of matrices taken from a finite set Σ as long as the system is delay-free. However, as it was discussed in the previous chapter with several examples, not every matrix that appears in the infinite product is a valid one in order to model the asynchronous updates after information reception. Soon, we will see that to cope with packet losses is necessary to add some extra variables that store information which can be used if packet losses occur. The approach with these new variables also is possible to analyze with a non-homogeneous infinite product of matrices, however in this chapter packet losses will not be treated directly, notwithstanding that, this phenomenon can be added easily to this theoretical framework.

This chapter will mention the main classical results in order to guarantee consensus and average consensus in a distributed manner. Not all the results treat the asynchronous phenomena directly, which is clear for instance in the section of Homogeneous Products, even most of the results in the Non-Homogeneous section will not cover directly the asynchronous phenomena (underlying graphs of matrices that are complete/rooted/sinked) but they prepare for the results that regard finite products of matrices whose underlying graphs are complete/strongly rooted/strongly sinked, which are formed naturally in a system with digital devices that performs asynchronous updates over a feasible communication topology strongly connected/rooted/sinked.

The analysis here assumes that every agent just possesses a unique state and, in order to proof convergence, it is imposed a strong assumption, but an intuitive one, which makes it clear when the product of matrices is closer to a consensus matrix, in particular non-negativeness of the coefficients and row sums equal to one make the state closer to the consensus as long as the information is distributed all over the nodes.

This chapter is organized first with the analysis of Homogeneous Products and next with Non-Homogeneous Products. The theorems that will be introduced here clarify the conditions to reach both consensus and average consensus and they will be applied in the following chapter to show the convergence of existing algorithms as study cases. Even though these results give insights to design an algorithm, they cannot be applied directly if we are considering asynchronous updates and packet losses in the case of the average consensus problem because some conditions are not valid in the framework that regards such non-ideal phenomena.

3.1. Homogeneous Products

In the classical approach, the updates and the transference of information between agents are reliable and occur instantaneously. It is clear that these assumptions in the model are quite strong and do not represent what happens in a real network with digital devices. Even though, there is a lot of research that works with this framework, mainly because it is relatively ease to prove theoretical results.

The classical approach regards that updates in a graph are performed synchronously and without communication delays, which means that there is a unique clock synchronized in every agent and the change of the states in the nodes is done at the same time. It is clear that in this case the distributed nature of the system is not entirely preserved. Furthermore, communications are reliable and therefore the updates have always the same dynamics every time the "clock of the whole network" triggers, which implies that linear protocols with time invariant coefficients change the state of the network homogeneously, i.e., with just one transition matrix at each step as is depicted with an example in Figure 3.1.

In this way, in the classical model we have the following update law:

$$x_{k+1} = Ax_k \qquad \Longrightarrow \qquad x_k = A^k x_0,$$

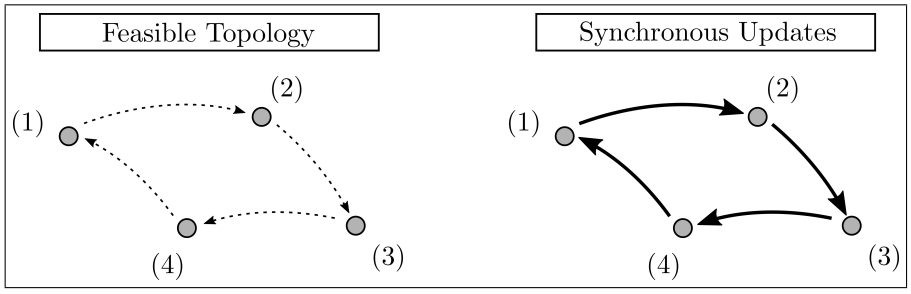

FIGURE 3.1. Example in which is shown a strongly connected feasible communication topology and the synchronous updates performed every time the "clock of the whole network" triggers, without presence packet losses. The updates are therefore constant every time. Clearly, this is not the behavior in our framework. Note that the underlying graph of the matrix induced by an update (non-negative with positive diagonal entries) is strongly connected and after a finite time the underlying graph of consecutive updates will be complete.

where $x_k \in \mathbb{R}^N$ is the state of the system at a given time instant $k \in \mathbb{R}$ that contains the individual states of all nodes, x_0 is the initial state of the graph and $A \in \mathbb{R}^{N\times N}$ represents a linear invariant transition matrix.

In classical control theory, the goal is to reach the zero state. However in the consensus problem we are interested in reaching a vector of the form $\alpha\mathbb{1}$ where $\alpha \in \mathbb{R}$, and in the average consensus $\alpha = \bar{x}$, where $\bar{x} = \frac{1}{N}\mathbb{1}^T x_0$ and N is the number of nodes, i.e., every component has to be the same in the consensus case and additionally the average consensus requires that the consensus must be equal to the average of the initial conditions. Note that in this context each component of the state $x_k \in \mathbb{R}^N$ represents the estimated average of each agent at certain time k, however we will see that we can add extra components to the state in order to make the algorithm robust to packet losses, in this new case we will be interested in the convergence to a vector of the form $\bar{x}\ \left[\mathbb{1}^T, \mathbb{0}^T\right]^T$ for the average consensus case.

It is easy to show that $\lim_{k\to\infty} x_k \to \bar{x}\,\mathbb{1}$ for all initial states x_0 if and only if $\lim_{k\to\infty} A^k = \frac{1}{N}\mathbb{1}\mathbb{1}^T$, and therefore the analysis of convergence to the average depends exclusively on the convergence of the powers of the matrix A. Focusing on this, the following result was proven in (Xiao & Boyd, 2004):

Theorem 3.1. $\lim_{k\to\infty} A^k = \frac{1}{N}\mathbb{1}\mathbb{1}^T$ *if and only if the next three statements are satisfied:*

(i) $\mathbb{1}^T A = \mathbb{1}^T$,

(ii) $A\mathbb{1} = \mathbb{1}$,

(iii) $\rho(A - \frac{1}{N}\mathbb{1}\mathbb{1}^T) < 1$,

where $\rho(\cdot)$ denotes the spectral radius of a matrix.

These conditions give us insights that we should have in mind when we treat with more general non-homogeneus matrix products to reach the average consensus.

The first condition means basically that the sum, and therefore the average, is preserved at each time for every value of the state vector, in fact $\mathbb{1}^T x_{k+1} = \mathbb{1}^T A x_k = \mathbb{1}^T x_k$, in particular this implies that $\mathbb{1}^T x_k = \mathbb{1}^T A^k x_0 = \mathbb{1}^T x_0$ and thus the average is conserved from the very beginning. It should be thought as a "conservation" property which is invariant every time a linear transformation is performed.

The second statement has an analogous analysis in the sense that it is another invariant property related to the "reachability of the consensus". Indeed, at each update is satisfied $x_{k+1} = Ax_k$, therefore if the actual state is $x_k = \alpha\mathbb{1}$, $\forall\alpha \in \mathbb{R}$ then the next state is the same $x_{k+1} = \alpha\mathbb{1}$, in particular $x_k = A^n x_{k-n}$, $\forall n < k$, and if $x_{k-n} = \alpha\mathbb{1}$ then $x_k = \alpha\mathbb{1}$. Thus the consensus vector $\mathbb{1}$ is a fixed or equilibrium point no matter how many updates have been carried out. However, it may be possible that there are other equilibrium subspaces besides the consensus subspace so there should be another condition that guaranties the uniqueness of the equilibrium point, which, as you may guess, is the third one.

But before mentioning the spectral radius condition, there is another insight hidden in the first two statements. Let's assume that there is a unique eigenvalue of A with value $\lambda = 1$, if the consensus is reachable (second condition), then the corresponding eigenvector space is a line with the direction of the consensus vector $\mathbb{1}$, so a priori the system has a subspace of dimension one with infinite equilibrium points and therefore if the powers of the A matrix converges it must have the form $A^\infty = \mathbb{1}\nu^T$, with $\nu \in \mathbb{R}^N$, i.e., every column of A^∞ can converge to any point in the consensus eigenvector space. However, if we append the conservation property and still assuming that the powers converge, the columns of the matrix A^∞ must be equal to $\frac{1}{N}\,\mathbb{1}$ and thus the convergence of the matrix must be of the form

$A^{\infty} = \frac{1}{N}\mathbb{1}\mathbb{1}^T$, so the equilibrium point of the system is unique but depends on the initial condition since it is equal to $\bar{x}\,\mathbb{1}$, something that is not usual in classical control frameworks in which the equilibrium in general does not depend on the initial state. Therefore, if we just regard the conservation property and that the consensus is the unique reachable subspace and additionally we assume that the powers of A converge, then the convergence has to be to the average matrix $\frac{1}{N}\,\mathbb{1}\mathbb{1}^T$.

Finally, the third condition along with the first and second statements imply that there is a unique eigenvalue $\lambda = 1$ and every other eigenvalue is strictly less than one therefore the convergence is guaranteed. The importance of the spectral radius condition is that it guarantees convergence. If we add conservation and reachability of the consensus with a unique one eigenvalue, then the average consensus is reached for every initial condition.

With the above result, we can make a simple analysis to define sufficient conditions to reach average consensus in a system of the form $x_{k+1} = Ax_k$. Basically consist in the analysis of the powers of A, and in a time invariant system, the Jordan decomposition is the right tool to use. Let J be the Jordan decomposition of A, such that $A = PJP^{-1}$, with P a matrix whose columns are eigenvectors and J a block diagonal matrix with its associated eigenvalues. Then the power of A satisfy:

$$A^k = PJ^kP^{-1},$$

therefore the convergence of A is equivalent to the convergence of J. From here, we see that the convergence depends on the values of the eigenvalues. We can say that if λ_i is an eigenvalue of A then we can classify the eigenvalues as follows, which is also depicted in Figure 3.2:

(i) If $|\lambda_i| < 1$, then λ_i is an stable eigenvalue whose powers reach zero.

(ii) If $\lambda_i = 1$ and simple, then λ_i is a critical stable eigenvalue which powers are one.

(iii) If $|\lambda_i| = 1$ with $\lambda_i \neq 1$ and simple, then λ_i is a critical stable eigenvalue which powers oscillate.

(iv) If $|\lambda_i| = 1$ and is not simple, then λ_i is an unstable eigenvalue.

(v) If $|\lambda_i| > 1$, then λ_i is an unstable eigenvalue.

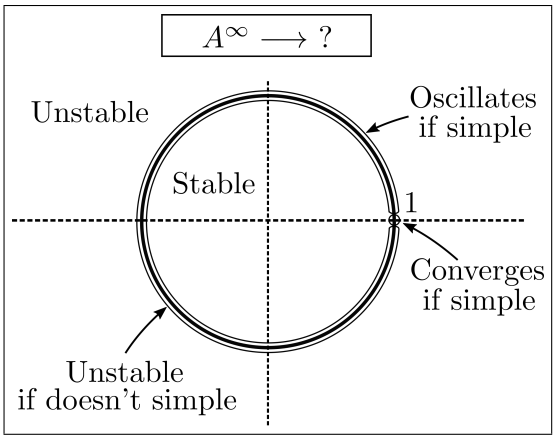

FIGURE 3.2. Classification of the eigenvalues according to its convergence behavior. The convergence analysis of the powers of a matrix is straightforward by using the Jordan decomposition.

With the aforementioned in mind, the following conditions are sufficient to achieve average consensus: $\mathbb{1}^T A = \mathbb{1}^T$, $A\mathbb{1} = \mathbb{1}$ and $|\lambda_n| \leq \ldots \leq |\lambda_2| < \lambda_1 = 1, \forall\, i \neq 1$, which intuitively are the properties of conservation, consensus is a reachable subspace, the consensus is the unique reachable subspace and the powers of A are convergent. Also, from the Jordan decomposition, it is clear that the convergence speed has a geometrical rate and is limited by second largest eigenvalue of A. The previous analysis is formalized in the following theorem:

Theorem 3.2. *Let $A \in \mathbb{R}^{N\times N}$ be a pseudo-double-stochastic matrix, i.e., $A\mathbb{1} = \mathbb{1}$ and $\mathbb{1}^T A = \mathbb{1}^T$, such that its eigenvalues satisfy $|\lambda_n| \leq \ldots \leq |\lambda_2| < \lambda_1 = 1, \forall\, i \neq 1$, then $\lim_{k\to\infty} A^k = \frac{1}{N}\mathbb{1}\mathbb{1}^T$, and convergence is at geometrical rate bounded by the second largest eigenvalue.*

Note that a priori it is not necessary to force non-negative coefficients in the iteration matrix A to design these values, in fact, faster convergence speed can be achieved by using negative elements. However many findings assume that the sum of the entries of each row are one and all are non-negative, i.e., $A\mathbb{1} = \mathbb{1}$ and $A \geq 0$. This is a common and natural way

to design the coefficients since it means that the next state x_{k+1} is the result of a linear convex combination of the states at time k and consequently it induces a row-stochastic matrix. Nevertheless, the design beforehand only requires that $A\mathbb{1} = \mathbb{1}$ to ensure that the consensus is a reachable subspace without the non-negative condition of the entries.

Notwithstanding the aforementioned, if we design with a row-stochastic transition matrix, i.e., $A\mathbb{1} = \mathbb{1}$ and $A \geq 0$, the analysis can be simplified. In fact, a priori it is known that the module of the eigenvalues of A are less or equal to 1 since the powers of A are row-stochastic matrices and thus every coefficient cannot be greater than one, so the powers are bounded and the matrix A is not unstable. Clearly, there exists the eigenvalue $\lambda = 1$, so the matrix A cannot be stable (cannot reach the zero). Even more, if we assume that all diagonal entries are non-zero, which is natural since it represents that the next state of a node $x_{k+1}^{(i)}$ depends on its own past state $x_k^{(i)}$, then the unique eigenvalue with magnitude equal to one is $\lambda = 1$ thanks to Gershgorin circle theorem as is shown in Figure 3.3. Thus, in order to reach consensus it is just necessary to add that A has a unique eigenvalue with magnitude one , which must be the trivial $\lambda = 1$. The latter can be obtained by appending the strongly connectivity property of the underlying graph induced by the non-negative matrix A, which again is a logical condition that must be presented in order to guaranty that the information is distributed in the whole network. Finally, if the conservation property $\mathbb{1}^T A = \mathbb{1}^T$ is appended, then not only the consensus is reached but also the value must be the average. The previous analysis can be summarized in the following theorems (see for instance (Olfati-Saber et al., 2007)):

Theorem 3.3. *Let $A \in \mathbb{R}^{N\times N}$ be a row-stochastic matrix, i.e., $A\mathbb{1} = \mathbb{1}$ and $A \geq 0$, with positive diagonal entries $diag(A) > 0$ and with a strongly connected underlying graph $\gamma(A)$, then $\lim_{k\to\infty} A^k = \mathbb{1}\nu^T$ with $\nu \in \mathbb{R}^N$.*

Theorem 3.4. *Let $A \in \mathbb{R}^{N\times N}$ be a double-stochastic matrix, i.e., $A\mathbb{1} = \mathbb{1}$, $\mathbb{1}^T A = \mathbb{1}^T$ and $A \geq 0$, with positive diagonal entries $diag(A) > 0$ and with a strongly connected underlying graph $\gamma(A)$, then $\lim_{k\to\infty} A^k = \frac{1}{N}\mathbb{1}\mathbb{1}^T$.*

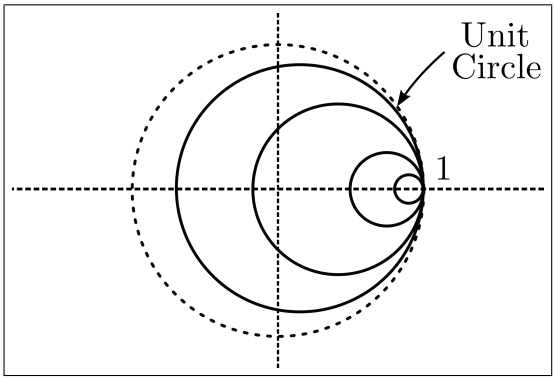

FIGURE 3.3. Thanks to the Gershgorin theorem, any row-stochastic matrix, i.e. $A\mathbb{1} = \mathbb{1}$ and $A \geq 0$, with positive diagonal entries $diag(A) > 0$ has its eigenvalues inside the unit circle or they are equal to one.

In summary, according to Theorem 3.2 to achieve average consensus it is necessary the conservation property, i.e., $\mathbb{1}^T A = \mathbb{1}^T$; the consensus is the unique reachable subspace, i.e., $A\mathbb{1} = \mathbb{1}$ such that there is one eigenvalue $\lambda = 1$ and is the unique one with magnitude equal to 1; and convergence property, i.e., every other eigenvalue of A must be strictly less than one. Also, if the transition matrix A is designed such that it is doubly stochastic (and therefore each element is non-negative $A \geq 0$) with strictly positive diagonal entries $diag(A) \geq 0$, convergence is guaranteed but still the uniqueness of the one eigenvalue is not necessarily true, which can be achieved by forcing the mild assumption of strongly connectivity of the underlying graph according to Theorem 3.4. Note that even though non-negativeness is not a necessary condition, it simplifies the analysis. In particular, the notion of underlying graph of a matrix is only well-defined for non-negative matrices and soon we will see that row-stochastic matrices have and implicit "passivity" property.

From the last paragraph it is clear that there are two ways to obtain the average consensus in the homogeneous case which are stated in theorems 3.2 and 3.4. Note that using the latter, theorem 3.4, is easier to design the coefficients of the matrix A and still guarantee convergence to the average since only requires that A is double-stochastic with positive diagonal entries, in contrast to the former, theorem 3.2, which requires an eigenvalue analysis beforehand to obtain average consensus. Here appears the following question: is it possible to establish a condition to guarantee both convergence and uniqueness of eigenvalue with

magnitude one with a pseudo-doubly-stochastic matrix, i.e., $\mathbb{1}^T A = \mathbb{1}^T$ and $A\mathbb{1} = \mathbb{1}$, without obtaining directly its eigenvalues?. This is still an open question.

In this section it was analyzed the convergence of the power of one single matrix which does not represent the updates that actually occur in our framework, notwithstanding that, many insights appeared in order to reach the average consensus. A theorem which imposes mild assumptions on the updates and the non-negativeness of the entries establishes sufficient conditions to design the coefficients on each agent. However, the matrices involved represent synchronous updates. What would happen if only one agent updates its internal state by sensing its neighbors an the others remain constant?. The intuition says that at least consensus should be reached, since the conservation property does not hold in this case. What would be the behavior if the updates where made by a transmission-reception process? This is a more difficult question, but again intuitively the consensus should be reached in the same way. The next section will treat the infinite product of non-homogeneous matrices in which sufficient conditions will be announced to reach the average consensus, however, the design of the coefficients cannot be applied on either of the two aforementioned contexts due to the conservation property is no longer satisfied with asynchronous updates at least with the use of one single internal variable per agent.

3.2. Non-Homogeneous Products

As we mention before, if the the system is synchronous, without packet losses and without delays then the model can be reduced to the analysis of an homogeneous matrix product of the form $x_{k+1} = Ax_k$, which is also a discrete linear time invariant system in which the time steps are not equally spaced. However, this is an unrealistic situation for a real IoT environment where the communication is limited by a transmission-reception process. Some authors have faced synchronism by thinking that the agents exchanging information at each time can change, and even the parameters of the algorithm may vary. If the updates are linear, then the change of the state can be performed by a transition matrix, which this time is not constant as in the previous section, however we will assume that the matrices that appear due to an update are taken from a finite set of matrices Σ. As an example, Figure 3.4 shows the

type of communications that can be performed in this context. Note that in the analysis of this section it is allowed the use of synchronous updates, however it is possible just to regard asynchronous matrices and equally use the results of this section for our asynchronous framework.

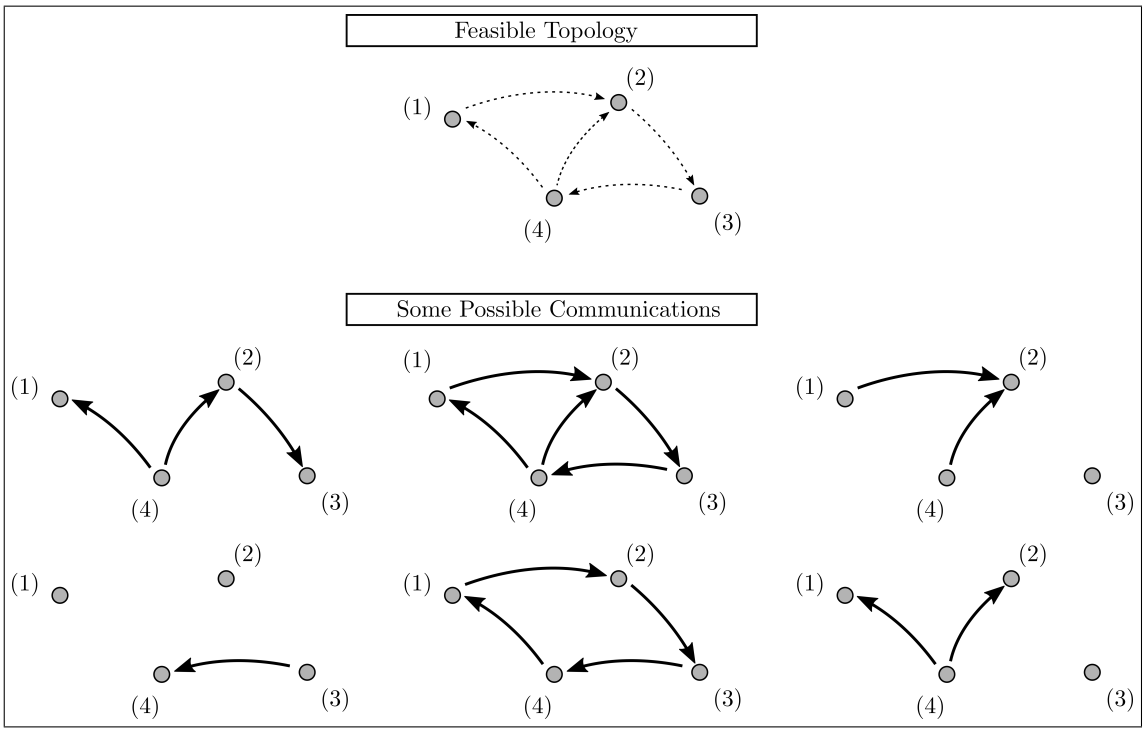

FIGURE 3.4. Example of a feasible communication topology and some of its possible communications. Note that the communication can induce synchronous updates, but also they can induce asynchronous updates as in our framework.

Thus, the mathematical model can be expressed by:

$$x_{k+1} = A_k x_k \qquad \Longrightarrow \qquad x_k = (A_{k-1} \cdots A_1 A_0) x_0,$$

where the meaning of each vector and matrix are the same as before, i.e., $x_k \in \mathbb{R}^n$ is the state of the graph at a given time instant $k \in \mathbb{R}$ that contains the individual states of all nodes, x_0 is the initial state of the graph and $A_k \in \mathbb{R}^{n \times n}$ represents different linear invariant transition matrices which are taken from a finite set Σ.

As in the homogeneous case, we are interested in the consensus and average consensus goals. Thus, in the consensus problem we focus on reaching a vector of the form $\alpha\mathbb{1}$ where $\alpha \in \mathbb{R}$, and on the average consensus $\alpha = \bar{x}$, where $\bar{x} = \frac{1}{N}\mathbb{1}^T x_0$ and N is the number of nodes and every component of the state represents the estimated average of the whole network.

The update law can be seen as a time varying linear system with not equally spaced time updates and the converge analysis of x_k is equivalent to the convergence of the non-homogeneous matrix product $(A_{k-1} \cdots A_1 A_0)$. In fact, $\lim_{k\to\infty} x_k \to \bar{x}\,\mathbb{1}$ for all initial states x_0 if and only if $\lim_{k\to\infty}(A_{k-1} \cdots A_1 A_0) = \frac{1}{N}\mathbb{1}\mathbb{1}^T$ for every sequence of matrices taken from the finite set Σ, and therefore the analysis of the average convergence depends exclusively on the non-homogeneous matrix products. For more information about the relationship between the linear system and the infinite matrix products see (Gurvits, 1995; Vladimirov, Elsner, & Beyn, 2000)

In general, it is desired that for any infinite sequence of matrices $A_0,\ A_1,\ \ldots,\ A_{k-1},\ \ldots$ taken from Σ, the infinite product $(A_\infty \cdots A_1 A_0)$ converges to a certain matrix. This set of matrices Σ is so called LCP. So far, only hard characterizations to prove if a set Σ is LCP exist, some findings on this can be found in (Daubechies & Lagarias, 1992, 2001; Berger & Wang, 1992; Hartfiel, 2002). In fact, there are results dealing with the convergence to zero or divergence of the infinite product where the analysis is given by the generalized spectral radius, which is equivalent to the joint spectral radius $\rho(\Sigma)$ when the set of matrices Σ is finite and represents the absolute value of the largest eigenvalue among all the possible products of matrices in Σ. If the joint spectral radius is strictly less than one, then any infinite product taken from Σ converges to zero. Also, if the joint spectral radius is strictly larger than one, then the stability is not guaranteed for some product sequences. Finally, if the joint spectral radius is equal to one, then it is uncertain if the infinite product taken from Σ converges or not (Jungers, 2009). This joint spectral radius analysis for convergence of infinite products is depicted in Figure 3.5.

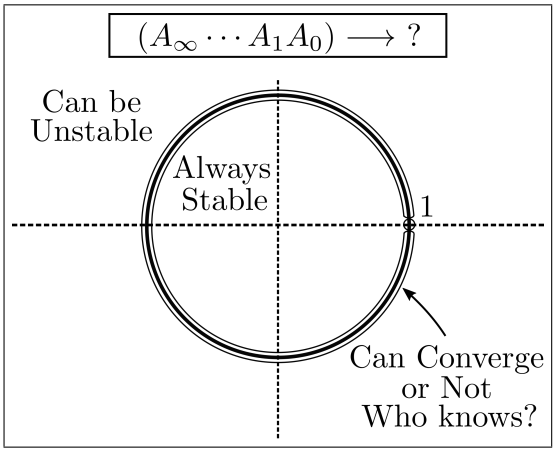

FIGURE 3.5. The convergence analysis of an infinite product matrices taken from a finite set Σ can be analyzed by studying the joint spectral radius $\rho(\Sigma)$. Just to illustrate, it is shown the complex plane, however the joint spectral radius is defined as an absolute value. Note that it is unknown if the system converges when the $\rho(\Sigma) = 1$.

It is important to highlight that computing the joint spectral radius is a really hard task. Researches have shown that obtaining its value from a finite set of matrices is a difficult task, indeed, computing the joint spectral radius is an NP-hard problem and, even worst, just to determine if the joint spectral radius is less than or equal to one is an undecidable problem (J. N. Tsitsiklis & Blondel, 1997; Blondel & Tsitsiklis, 2000). However, even though computing the joint spectral radius is difficult in general, in some cases obtaining its value is easy to handle by exploiting the properties of the matrices involved.

In the context of the average consensus problem, the joint spectral radius must be exactly one in order to converge to a state different from zero and not to diverge for some infinite sequences of products, however, in this case nothing can be said about the convergence behavior of the infinite product of matrices taken from Σ (compare this with the homogeneous case of eigenvalues with magnitude one in Figure 3.2). A simple analysis indicates that any finite product taken from Σ must give as a result a matrix which possesses all its eigenvalues inside the unit circle and at least one eigenvalue is equal to one and simple (Hartfiel, 2002), and also that every infinite product $A_\infty \cdots A_1 A_0$ must converge to matrix with exactly one eigenvalue equal to one an the remaining must be zero, which is a projection matrix.

The main reason why the analysis of non-homogeneous matrix products becomes really hard is because there is not a clear decomposition as in the homogeneous case, where the eigenvalue analysis is the key to understand the stability and even the convergence speed. Note that in the homogeneous case for a convergent matrix A, i.e., eigenvalues are in the unit circle or they are equal to one and simple, every time a new matrix is added to the finite homogeneous product A^k, the eigenvalues of the resultant matrix A^{k+1} are less than or equal to the eigenvalues of A^k. However, in the non-homogeneous context, this property does not hold and, therefore, for a finite sequence of matrices A_0, A_1, $\ldots$, A_{k-1}, A_k the eigenvalues of $(A_k A_{k-1} \cdots A_1 A_0)$ are not necessarily less than or equal to the eigenvalues of $(A_{k-1} \cdots A_1 A_0)$. Soon we will use a "coefficient" that possesses a similar sub-multiplicative property for the non-homogeneous products of row-stochastic matrices.

Let us recall the four properties discussed in the homogeneous case section that should be satisfied by the unique transition matrix A in order to converge to the average consensus: the sum is preserved $\mathbb{1}^T A = \mathbb{1}^T$, the consensus is a reachable subspace $A\mathbb{1} = \mathbb{1}$, the consensus is the unique reachable subspace or equivalently there is a unique eigenvalue equal to one, and the infinite product A^∞ converges or every other eigenvalue is strictly less than one. From here, it is reasonable that the conservation $\mathbb{1}^T A = \mathbb{1}^T$ and the reachability of the consensus $A\mathbb{1} = \mathbb{1}$ also should be necessary in the non-homogeneous case for every matrix $A \in \Sigma$ finite. Indeed, if the conservation is not preserved in each update then it would not be expected that after a certain period of time the sum of the components of the state hold the same as the sum of the initial conditions, also the reachability of the consensus is necessary because it is desired that once the consensus is reached the state should remain permanently no matter what update is applied to the system. However, the remaining two conditions of uniqueness of the reachable subspace and convergence are characterized in the homogeneous case in terms of eigenvalues properties; yet, they cannot be applied directly in infinite products of non-homogeneous matrices taken from a finite set Σ since here the notion of eigenvalues depends on different product sequences of matrices.

The uniqueness of the reachable subspace for the non-homogeneous case can be characterized without using eigenvalue properties as long as some or all the matrices in Σ are

appearing in the infinite product $(A_\infty \cdots A_1 A_0)$. Note that this assumption of repeating infinitely often the matrices in Σ is valid in our framework since it is consistent with the assumption 2.1 which indicates that every asynchronous communication link is always appearing as the system evolves. Also, it is worth pointing out that with this assumption we are not regarding all the infinite sequences of matrix products since only a subset of them is being considered (the subset in which all of the matrices in Σ appears infinitely often). Therefore, it is not necessary to analyze if a finite set of matrices Σ is LCP, instead it is just necessary to prove convergence of products of all infinite sequences of matrices which are repeated infinitely often. Even though this is lighter problem, it is still very challenging. However, the LCP property is desired because this shows certain degree of robustness since it implies that every component of the state is bounded as the system evolves, which can be useful, for instance, to estimate the size of the memory required for the digital devices.

The following result indicates the form of the resulting matrix of an infinite product. This is directly concerned with the uniqueness of the consensus as a reachable subspace for non-homogeneous matrix products, however it requires beforehand to know that the infinite product converges, which is a very strong assumption, and that some matrices are repeated infinitely often, something that is satisfied naturally in our context where the digital devices are non-defective (Hartfiel, 2002):

Theorem 3.5. *Let an infinite product $(A_\infty \ldots A_1 A_0)$ of matrices taken from a finite set Σ, such that some matrices are repeated infinitely often in the product. If $(A_\infty \ldots A_1 A_0)$ converges, then every column of the infinite matrix product must be in intersection of the 1-eigenspaces of the matrices that are repeated infinitely often.*

Let us assume that the finite set of matrices Σ has the conservation property $\mathbb{1}^T A = \mathbb{1}^T$ and the reachability of the consensus $A\mathbb{1} = \mathbb{1}$ and also the intersection of the 1-eigenspaces of the matrices in Σ is the trivial consensus subspace $\alpha\mathbb{1}, \forall \alpha \in \mathbb{R}$. Now consider that all the matrices in Σ appears infinitely often for all the infinite sequences of matrices $A_0,\ A_1,\ \ldots,\ A_{k-1},\ \ldots$. What is the form of the infinite product $(A_\infty \ldots A_1 A_0)$ if it converges?. The answer is quite straightforward because according to Theorem 3.5 every column of $(A_\infty \ldots A_1 A_0)$

is in the consensus subspace $\alpha\mathbb{1}$ and thanks to the conservation property every column of $(A_\infty \dots A_1 A_0)$ must sum 1 and therefore α must be equal to $\frac{1}{N}$. Consequently, $(A_\infty \dots A_1 A_0) = \frac{1}{N}\mathbb{1}\mathbb{1}^T$. This result is formalized in the following theorem:

Theorem 3.6. *Consider an infinite product $(A_\infty \dots A_1 A_0)$ of matrices taken from a finite set Σ such that all matrices in Σ are repeated infinitely often in the product. Also, assume that all the matrices A in Σ satisfies $\mathbb{1}^T A = \mathbb{1}^T$ and $A\mathbb{1} = \mathbb{1}$ such that the intersection of the 1-eigenspaces of the matrices in Σ is the trivial consensus subspace $\alpha\mathbb{1}$. If $(A_\infty \dots A_1 A_0)$ converges, then $(A_\infty \dots A_1 A_0) = \frac{1}{N}\mathbb{1}\mathbb{1}^T$.*

Theorem 3.6 gives sufficient conditions to reach the average consensus matrix as long as the convergence is guaranteed. However, as we discussed, the convergence property is really difficult to prove for general non-homogeneous matrix products.

Let us put aside for a while the average consensus goal and just think about the consensus problem, be aware that this is done in order to discover convergence properties. It is evident that in this case consensus must be a reachable subspace, so it is necessary that $A\mathbb{1} = \mathbb{1}$ for all A in Σ. What would happen if in addition we assume that every matrix in Σ is non-negative, i.e., we regard a finite set of row-stochastic matrices?. The question is open, yet we can say in advance that some convergence properties appear. For an extensive treatment of products of row-stochastic matrices see (Seneta, 2006; Rhodius, 1997).

Indeed, since the product of row-stochastic matrices gives as a result a row-stochastic matrix and that each entry of a row-stochastic matrix is between zero and one, it is clear that the set Σ is product bounded and does not diverge. Also, since every row sums one, then the infinite product of matrices taken from Σ cannot reach the zero matrix. Therefore, it is clear that the joint spectral radius is equal to one. However, it is not clear if any infinite product $(A_\infty \dots A_1 A_0)$ of matrices taken from Σ converges to the consensus matrix of the form $\mathbb{1}\nu^T$, i.e., a matrix with equal rows, which in the literature is sometimes called ergodicity property (Seneta, 2006; Rhodius, 1997; Hartfiel, 2002).

According to Theorem 3.5, if we assume that in the infinite product all the matrices in Σ appear infinitely often and the intersection of the 1-eigenspaces is exclusively the trivial consensus subspace $\alpha\mathbb{1}$ and it is known that the infinite product converges, then the convergence must be to the consensus matrix $\mathbb{1}\nu^T$. Thus, it only remains to prove that for any sequence of matrices the infinite product is not "oscillating" (convergence to zero and instability have already been ruled out). It is important to remark that the ergodicity property a priori does not require that the matrices involved are non-negative, indeed, as we discussed, it is just necessary that the consensus is the unique reachable subspace. However, the analysis of convergence becomes easier if the matrices in Σ are non-negative, and even further, with non-negative the notion of an underlying graph can be used.

One of the main reasons why the convergence analysis becomes less complicated is because there is a clear notion of "passivity" every time there is a change of the state when non-negative coefficients are involved (Cao et al., 2008a). In fact, let A_k be a row-stochastic matrix ($A_k\mathbb{1} = \mathbb{1}$, $A_k \geq 0$) taken from Σ, a non-negative vector $x_k \geq 0$ (which can be thought of as a column of the product of past matrices) and the following energy function $V(x_k) = \max\{x_k\} - \min\{x_k\}$. Then is easy to see that the next state $x_{k+1} = A_k x_k$ is bounded trivially by the maximum and minimum component of the past state x_k:

$$\min\{x_k\}\mathbb{1} = \min\{x_k\}A_k\mathbb{1} \leq A_k x_k = x_{k+1} = A_k x_k \leq \max\{x_k\}A_k\mathbb{1} = \max\{x_k\}\mathbb{1},$$

$$\min\{x_k\}\mathbb{1} \leq x_{k+1} \leq \max\{x_k\}\mathbb{1} \quad ,$$

then it must be true that the minimum grows and the maximum decreases monotonically:

$$\min\{x_k\} \leq x_{k+1}^{(i)} \leq \max\{x_k\}, \qquad \forall i \in \{1, \ldots, N\} \quad ,$$

and therefore the energy function $V(x_k)$ decreases monotonically at every step:

$$V(x_{k+1}) \leq V(x_k) \quad .$$

Note that this energy function is quantifying the distance to the consensus vector. In the case when all components are equal, both the maximum and the minimum are the same,

therefore the energy function is zero and once reached the consensus point is held forever. However, there is still no guaranties that the energy is strictly decreasing.

Other functions similar to the above have been extensively studied but this time a row-stochastic matrix is used as a domain with a real value range. They are called coefficients of ergodicity and quantify how different the rows of a matrix are. The following coefficient of ergodicity $\tau(A)$ is the most used:

$$\tau(A) = \frac{1}{2} \max_{i,j} \{\|(e_i - e_j)^T A\|_1\}$$

which obtains a value comparing every two rows of the matrix A. Among its properties, three of them are crucial to dealing with row-stochastic matrices. Let A, A_1 and A_2 be row-stochastic matrices then:

- $0 \leq \tau(A) \leq 1$,
- $\tau(A) = 0$ if and only if $A = \mathbb{1}\nu^T$, and
- $\tau(A_2 A_1) \leq \tau(A_2)\tau(A_1)$.

We can think of the coefficient of ergodicity as an energy function as in the previous case. The first property can be abstracted as every row-stochastic matrix is bounded by a energy between zero and one, the second property indicates that the zero energy is given when all the rows are equal or equivalently every column is in the consensus subspace, and finally, the third property along with the first one implies that the product of two row-stochastic matrices has an energy less than or equal to the product of the both energies separately. Note that this coefficient is somehow replacing the use of the eigenvalues which are submultiplicative in the homogeneous case and are not straightforward to use in the non-homogeneous context where the ergodicity coefficient is indeed submultiplicative.

With the above in mind, it is clear that we would like the coefficient of ergodicity of the infinite product converges to zero $\tau(A_\infty \dots A_1 A_0) \to 0$ in order to achieve consensus. Obviously, a sufficient condition is that every row-stochastic matrix in the product taken from

a finite set Σ has a ergodicity coefficient with a value strictly less than one:

$$\tau(A_\infty \dots A_1 A_0) \;\leq\; \dots \underbrace{\tau(A_{k-1})}_{<1} \dots \underbrace{\tau(A_1)}_{<1} \underbrace{\tau(A_0)}_{<1},$$

and therefore as $\dots \tau(A_{k-1}) \dots \tau(A_1)\tau(A_0) \to 0$ clearly $\tau(\dots A_{k-1} \dots A_1 A_0) \to 0$ too. However, in the context of asynchronous updates, in general all the matrices in Σ has a coefficient of ergodicity equal to one. In this case, it is possible to group in blocks some matrix products such that a property making its energy strictly less than one is satisfied:

$$\tau(A_\infty \dots A_1 A_0) \;\leq\; \cdots \underbrace{\tau(A_s \dots A_{r+1} A_r)}_{<1} \cdots \underbrace{\tau(A_q \dots A_{p+1} A_p)}_{<1} \cdots \underbrace{\tau(A_k \dots A_1 A_0)}_{<1},$$

where $k < p < q < r < s$, and again as the right term tends to zero, the infinite product $(A_\infty \dots A_1 A_0)$ converges to the consensus matrix $\mathbb{1}\nu^T$.

Therefore, it is necessary to find properties making the coefficient or ergodicity strictly less than one. It is easy to prove that if a row-stochastic matrix has at least one row with positive entries then its ergodicity coefficient is less than one, consequently a row-stochastic matrix whose entries are all positive also has a coefficient strictly less than one. At this point, the connection with underlying graphs induced by a non-negative matrix are relevant because a row-stochastic matrix, which represents a synchronous update, with a positive row means that all the nodes in the network receives information from one root agent, i.e., induces a strongly rooted graph. Similarly, a row-stochastic matrix with positive entries means that the underlying graph is complete or equivalently every node receives information from all the agents in the network at the same time. There is an equivalent characterization to determine if a row-stochastic matrix A has a coefficient of ergodicity strictly less than one $\tau(A) < 1$, in this case A is called a scrambling matrix and in terms of underlying graphs means that for every pair of distinct nodes there is an agent (that can be one of the pair) that sends information to the pair, this also is called a neighbor-shared graph. Even though neighbor shared graphs are equivalent to saying that $\tau(A) < 1$, it is more intuitive to think in terms of strongly rooted graphs or complete graphs, which are examples of neighbor-shared topologies.

From the previous discussion, the following theorem appears naturally:

Theorem 3.7. *Let Σ be a finite set of row-stochastic matrices whose underlying graphs are neighbors-shared or strongly rooted or complete, then the infinite product $(A_\infty \dots A_1 A_0)$ of matrices taken from Σ converges to the consensus matrix $\mathbb{1}\nu^T$.*

Note that in the previous theorem 3.7 the convergence speed is geometrical and upper bounded by the maximum value of the coefficients of the matrices in Σ, however, probably the convergence rate is faster as more matrices with less values of the coefficients of ergodicity appears in the first terms of the infinite product.

The problem with theorem 3.7 is that the matrices in Σ represent synchronous updates which are not valid in our asynchronous framework. However, since the infinite product can be separated by blocks of finite products and if these finite products have underlying graphs strongly rooted or complete (also can be neighbor-shared but this case is not intuitive to generate by matrix products), then the infinite product must converge to the consensus matrix. This ideas are formalized in the following theorem:

Theorem 3.8. *Let Σ be a finite set of row-stochastic matrices and assume that the infinite product $(A_\infty \dots A_1 A_0)$ of matrices taken from Σ is such that the product sequence can be separated into finite products of matrices which are strongly rooted or complete, then the infinite product $(A_\infty \dots A_1 A_0)$ converges to the consensus matrix $\mathbb{1}\nu^T$.*

Note that in theorem 3.8 the upper bound of the convergence speed depends on the largest coefficient of ergodicity among all the finite product of matrices taken from a particular infinite product sequence of matrices. This is impractical since in general is unknown when the matrices in Σ appear in the non-homogeneous product. Notwithstanding that, the stability behavior is still guaranteed.

Also in theorem 3.8 it is clear that the convergence property relies on the generation of underlying graphs which are either strongly rooted or complete, therefore it is necessary to impose a condition for this to be satisfied. It is clear that, as long as the row-matrices in Σ with positive diagonal entries are generated from a feasible communication topology which

is strongly connected and all the matrices in Σ appears infinitely often in the infinite product $(A_\infty \dots A_1 A_0)$, then every infinite sequence of matrix products can be separated by finite products of complete graphs and therefore the consensus is reached:

Theorem 3.9. *Let Σ be a finite set of row-stochastic matrices with positive diagonal entries generated from a strongly connected feasible communication topology, and assume that every matrix in Σ appears infinitely often in the infinite product $(A_\infty \dots A_1 A_0)$, then $(A_\infty \dots A_1 A_0)$ converges to the consensus matrix $\mathbb{1}\nu^T$.*

Even when the previous results only focus on row-stochastic matrices, they hold when the finite set Σ have only column-stochastic matrices, but the convergence of the infinite product $(A_\infty \cdots A_1 A_0)$ is to a matrix of the form $\nu\mathbb{1}^T$. This is true since the transpose of a left infinite product of row-stochastic matrices is a right infinite product of column-stochastic matrices (Liu & Morse, 2012). The analogous results are the following:

Theorem 3.10. *Let Σ be a finite set of column-stochastic matrices whose underlying graphs are strongly sinked or complete, then the infinite product $(A_\infty \dots A_1 A_0)$ of matrices taken from Σ converges to the matrix $\nu\mathbb{1}^T$.*

Theorem 3.11. *Let Σ be a finite set of column-stochastic matrices and assumes that the infinite product $(A_\infty \dots A_1 A_0)$ of matrices taken from Σ is such that the product sequence can be separated by finite products of matrices which are strongly sinked or complete, then the infinite product $(A_\infty \dots A_1 A_0)$ converges to the matrix $\nu\mathbb{1}^T$.*

Theorem 3.12. *Let Σ be a finite set of column-stochastic matrices with positive diagonal entries generated from a strongly connected feasible communication topology, and assume that every matrix in Σ appears infinitely often in the infinite product $(A_\infty \dots A_1 A_0)$, then $(A_\infty \dots A_1 A_0)$ converges to the matrix $\nu\mathbb{1}^T$.*

Recall that we want convergence to the average matrix $\frac{1}{N}\mathbb{1}\mathbb{1}^T$. In order to do so, it is necessary to have the properties of reachability of the consensus and conservation, i.e., every matrix A in Σ is such that $A\mathbb{1} = \mathbb{1}$ and $\mathbb{1}^T A = \mathbb{1}^T$. In fact, if the infinite product

$(A_\infty \dots A_1 A_0)$ of matrices taken from Σ converges to the consensus matrix $\mathbb{1}\nu^T$ and since every column of the infinite product must sum one, then every entry of ν has to be equal to $\frac{1}{N}$ and consequently the consensus matrix takes the form of the average consensus one $\frac{1}{N}\mathbb{1}\mathbb{1}^T$:

Theorem 3.13. *Let Σ be a finite set of double-stochastic matrices with positive diagonal entries generated from a strongly connected feasible communication topology, and assume that every matrix in Σ appears infinitely often in the infinite product $(A_\infty \dots A_1 A_0)$, then $(A_\infty \dots A_1 A_0)$ converges to the average consensus matrix $\frac{1}{N}\mathbb{1}\mathbb{1}^T$.*

In this section, the two main theorems 3.6 and 3.13 where stated for the infinite product $(A_\infty \cdots A_1 A_0)$, of matrices taken from a finite set Σ such that all the matrices in Σ appear infinitely often in the infinite product, to converge to the average consensus matrix $\frac{1}{N}\mathbb{1}\mathbb{1}^T$.

Theorem 3.6 gives conditions to reach average consensus for general matrices, in which the conservation property $\mathbb{1}^T A = \mathbb{1}^T$ and the reachability of the consensus $A\mathbb{1} = \mathbb{1}$ are needed, and, in order to make consensus the unique reachable subspace, the intersection of the 1-eigenspaces of the matrices which are repeated infinitely often in the product has to be the trivial consensus subspace $\alpha\mathbb{1}$. However, Theorem 3.6 also requires that the convergence must be known beforehand, which is really hard to prove for general matrices.

On the other hand, in the case of Theorem 3.13 every matrix A in Σ is non-negative with positive diagonal entries such that $\mathbb{1}^T A = \mathbb{1}^T$ and $A\mathbb{1} = \mathbb{1}$. We saw that the convergence analysis can be done by analyzing the coefficient of ergodicity of row-stochastic matrices, in which the notion of "passivity" or non-increasing energy is satisfied every time a new matrix is appended to the finite product of matrices and the energy is zero if and only if the consensus matrix $\mathbb{1}\nu^T$ is reached. Here, convergence to the consensus is satisfied as long as finite products of matrices whose underlying graphs are complete are repeatedly induced, since this represents energy strictly less than one, which is true in the case that the finite set of matrices Σ is generated from a strongly connected feasible communication and all the matrices in Σ appears infinitely often in the infinite product $(A_\infty \cdots A_1 A_0)$. Note that, in contrast to the general case of theorem 3.6, it is not necessary to think in the property of uniqueness of the consensus as reachable subspace since the coefficient of ergodicity naturally indicates zero

energy when consensus is reached, and also, and more importantly, theorem 3.13 guarantees convergence, which in theorem 3.6 needs to be assumed.

Chapter 4. EXISTING ALGORITHMS

In this chapter we are going to discuss some existing algorithms to solve the consensus and average consensus problems. Basically, we will establish the framework of each communication network, then the rules that every agent must follow to reach the distributed goal and, assuming that the communication channel is not subject to packet losses or delays. It will be shown the form or the matrices induced by each algorithm and the properties that satisfies, to finally invoke convergence results discussed in the previous chapter, in particular theorems 3.9 and 3.13, which regard product of non-negative matrices with positive diagonal entries. Also, insights when packet losses occur will be pointed out and we will see that, in this unreliable context, the algorithms can reach the consensus but not to the average since the conservation property is no longer satisfied.

Furthermore, in the second section of the chapter an evaluation of the broadcast-gossip and push-sum algorithms will be conducted with numerical simulations using simulations and in real hardware over a local pilot-scale network. This evaluation reveals that non-ideal phenomena, mainly packet losses, are harmful to reach consensus to the average. In particular, some insights about the different behavior of the evaluated protocols will be revealed by analyzing packet losses and multi-rate phenomena.

4.1. Study Cases

4.1.1. Vicsek's Problem

This problem considers a group of N digital devices that can interact in a given feasible strongly connected communication graph that is unchanged over time, each agent i has a unique internal state $x_k^{(i)} \in \mathbb{R}$ at a certain time instant k. Every node has an internal clock with frequencies and phases that are not necessarily the same and therefore the multi-rate property is present. Once in a while thanks to its internal clock, the digital device i can sense the state of all its neighbors $x_k^{(j)}\ \forall j \in \mathcal{V}_{in}^{(i)}$ included itself $x_k^{(i)}$ and then updates its state $x_{k+1}^{(i)}$ as a function of them. It is assumed that just one agent changes its state at time k, also

the update rule on each node is a linear function with constant coefficients configured at the beginning. The final goal is to reach consensus.

The algorithm configured on each agent i is the following, where $x^{(i)}$ is the state variable of node i and $x^{(j)}$ is the state sensed from the in-neighbor j:

- Initialization:
$$x^{(i)} = m_i$$
- If node i generates a timer event: then it senses all the states of its neighbors and performs the following update:
$$x^{(i)} \leftarrow a_{ii}x^{(i)} + \sum_{j \in \mathcal{V}_{in}^{(i)}} a_{ij}x^{(j)},$$
where $a_{ii} > 0$, $a_{ij} \geq 0$, and $\sum_{j \in \mathcal{V}_{in}^{(i)} \bigcup \{i\}} a_{ij} = 1$, i.e., is a convex linear combination with positive coefficients.

Since there are not packet losses nor delays, the update of the states if the timer of node i triggers at instant k, and j is an agent different from i:

$$\begin{aligned} x_{k+1}^{(i)} &= a_{ii}x_k^{(i)} + \sum_{j \in \mathcal{V}_{in}^{(i)}} a_{ij}x_k^{(j)} \\ x_{k+1}^{(j)} &= x_k^{(j)} \\ x_{k+1}^{(r)} &= x_k^{(r)}, \end{aligned}$$

where $r \in \mathcal{V}$, $r \neq i$, $r \neq j$ is a node which does not perform any update, an therefore the update of the system state $x_{k+1} \in \mathbb{R}^N$ can be expressed as an iteration made by a transition matrix. For instance, for the case when there are $N = 5$ nodes, the timer of the agent $i = 2$

triggers and its neighbors are $j = \{1,\, 4\}$ the following update is generated:

$$\begin{pmatrix} x_{k+1}^{(1)} \\ x_{k+1}^{(2)} \\ x_{k+1}^{(3)} \\ x_{k+1}^{(4)} \\ x_{k+1}^{(5)} \end{pmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ a_{21} & a_{22} & 0 & a_{24} & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} x_k^{(1)} \\ x_k^{(2)} \\ x_k^{(3)} \\ x_k^{(4)} \\ x_k^{(5)} \end{pmatrix},$$

which underlying graph is depicted in Figure 4.1.

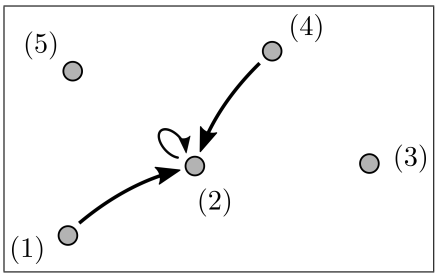

FIGURE 4.1. Example of the underlying graph induced by a matrix update of the algorithm to solve the distributed consensus problem. The self-loop was added which represents that the next state of the node depends on its actual state.

Note that if for any reason a node cannot sense a neighbor after a timer event, in the protocol the coefficients would be different from the ones considering all neighbors, however the sum of the coefficients must be one. An example of an update law which satisfies the aforementioned would be the following:

$$x_{k+1}^{(i)} = \tfrac{1}{1+|\mathcal{V}_{in}^{(i)}|} x_k^{(i)} + \sum_{j \in \mathcal{V}_{in}^{(i)}} \tfrac{1}{1+|\mathcal{V}_{in}^{(i)}|} x_k^{(j)}.$$

The matrices that are generated every time there is an update have a limited number. Also, due to the internal timer, we can assume that eventually all the matrices will appear as the system evolves even with packet losses. Hence, it is possible to say that the system $x_{k+1} = A_k x_k$ can be analyzed as an infinite matrix product $x_k = (A_{k-1} \cdots A_1 A_0) x_0$ where the order in which the matrices appear is unknown but all the matrices in the finite set Σ are repeated infinitely often as the system evolves. Note that every matrix in Σ is such that

all their rows sum one and is non-negative with positive diagonal entries, so the notion of underlying graph can be applied and furthermore, every time there is an update new links not presented in the feasible communication topology can appear.

Since the finite set of matrices Σ is generated from a strongly connected feasible communication topology, then the infinite product $(A_\infty \cdots A_1 A_0)$ always can be separated by finite products of matrices whose underlying graphs are complete according to theorem 2.2. Now, since every complete graph have a coefficient of ergodicity strictly less than one and they are appearing infinitely often in the product, then the infinite product $(A_\infty \cdots A_1 A_0)$ must converge to the consensus matrix of the form $\mathbb{1}\nu^T$ according to theorem 3.9.

It is necessary to point out that in general only consensus is reached but not to the average since the preservation property does not hold, unless the matrices in Σ were assumed column stochastic too. However, since the protocol only can be coded on every digital device independently, and asynchronous updates and multi-rate behaviors are present, there are not control over the other coefficients of the matrix more than the ones in the row of the "sensor" node. In particular, the columns of the non-updated sensed agents are larger than one. Also, note that consensus is robust to packet losses since every agent can sense the state of its neighbors, indeed the row-stochastic matrices induced can be controlled by the "sensor" node. Finally, we highlight that these updates are not possible in our framework since the updates are performed thanks to a transmission-reception process instead of sensing the states.

4.1.2. Gossip Approach

Again, this problem considers a group of N digital devices that can interact in a given feasible strongly connected communication graph that is constant over time. Each agent i has a unique internal state $x_k^{(i)} \in \mathbb{R}$ at certain time instant k. Every node has an internal clock whose frequencies and phases are not necessarily the same and therefore the multi-rate property is present. Once in a while thanks to its internal clock, the digital device i chooses one of its neighbors j and then it transmits its state to j with a message labeled as type 1. After that the node j detects an event of data reception that allows it to update its internal

state as the average of its own state and the state received from i, and then j sends back the state with a message labeled as type 2. Finally i detects the associated event of data reception and updates its state with the same value of the ith agent state. Every time a node i generates a timer event, it chooses a different neighbor to transmit data to in such a way that all neighbors are chosen after a certain finite time. It is assumed that the two agents that exchange information update their states at the same time instant, even though there are two sequential updates. The final goal is to reach the average consensus.

The algorithm configured on each agent i is the following, where $x^{(i)}$ is the state variable of node i and $x^{(j)}$ is the state received from the in-neighbor j:

- Initialization:
$$x^{(i)} = m_i$$
- If node i generates an timer event:
  1) Choose a neighbor j in such a way that every neighbor is chosen after a finite time.
  2) Transmit the state to agent j and labeled as type 1.
- If node i detects an event of data reception with type 1 label:
  1) Update its own state as follows:
$$x^{(i)} \leftarrow \tfrac{1}{2}\left(x^{(i)} + x^{(j)}\right)$$
  2) Send the updated state back to the transmitter node j.
- If node i detects an event of data reception with type 2 label:
  1) Update its own state as follows:
$$x^{(i)} \leftarrow x^{(j)}$$

Since there are not packet losses nor delays, and assuming that the updates in the nodes involved in the communication occur at the same time k, the updates of the states at time k when the timer of node i triggers and it chooses neighbor j, which thanks to the "instantaneous" assumption is equivalent to the timer of j triggering and choosing neighbor i, are as

follows:

$$\begin{aligned} x_{k+1}^{(i)} &= \tfrac{1}{2}x_k^{(i)} + \tfrac{1}{2}x_k^{(j)} \\ x_{k+1}^{(j)} &= \tfrac{1}{2}x_k^{(i)} + \tfrac{1}{2}x_k^{(j)} \\ x_{k+1}^{(r)} &= x_k^{(r)} \quad , \end{aligned}$$

where $r \in \mathcal{V}$ is a node not involved in the gossip communication process, and therefore the update of the system state $x_{k+1} \in \mathbb{R}^N$ can be expressed as an iteration made by a transition matrix. For instance, the case when there are $N = 5$ nodes, the timer of agent $i = 3$ triggers and its neighbor chosen is $j = 5$, generates the following update:

$$\begin{pmatrix} x_{k+1}^{(1)} \\ x_{k+1}^{(2)} \\ x_{k+1}^{(3)} \\ x_{k+1}^{(4)} \\ x_{k+1}^{(5)} \end{pmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1/2 & 0 & 1/2 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1/2 & 0 & 1/2 \end{bmatrix} \begin{pmatrix} x_k^{(1)} \\ x_k^{(2)} \\ x_k^{(3)} \\ x_k^{(4)} \\ x_k^{(5)} \end{pmatrix},$$

whose underlying graph is depicted in Figure 4.2.

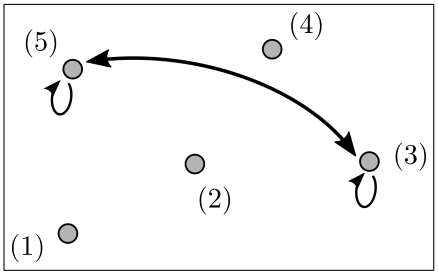

FIGURE 4.2. Example of the underlying graph induced by a matrix update of the algorithm to solve the distributed average consensus problem with the gossip approach. Note that it is assumed that the two agents involved in the communication update their states at the same time, however in practice this is not possible.

The matrices that are generated every time there is an update have a limited number. Also, due to internal timers, we can assume that eventually all the matrices will appear as the system evolves. So again the analysis of the system is based on the convergence of the infinite product $(A_\infty \cdots A_1 A_0)$ of matrices taken from a finite set Σ, where the order in which the

matrices appear is unknown but all the matrices in the finite set Σ are repeated infinitely often as the system evolves. Note that every matrix in Σ is such that all are double-stochastic with positive diagonal entries, so the notion of underlying graph can be applied and furthermore, every time there is an update new links not present in the feasible communication topology can appear.

Since the finite set of matrices Σ is generated from a strongly connected feasible communication topology, then the infinite product $(A_\infty \cdots A_1 A_0)$ always can be separated into finite products of matrices whose underlying graphs are complete according to theorem 2.2. Now, since every complete graph have a coefficient of ergodicity strictly less than one and they are infinitely appearing in the product, then the infinite product $(A_\infty \cdots A_1 A_0)$ must converge to the average consensus matrix of the form $\frac{1}{N}\mathbb{1}\mathbb{1}^T$ according to theorem 3.13.

Note that sequential unreliable communication is what happens in real life. If there is a packet loss in the type 1 message, then there is not update at all, thus the transition matrix can be regarded as the identity and there are not problems. If the type 1 message is successfully received by j but the associated message type 2 never reaches the i node, then the transition matrix is not doubly-stochastic anymore, instead is just row-stochastic. In the above example the matrix with a type 2 packet loss would be:

$$\begin{pmatrix} x_{k+1}^{(1)} \\ x_{k+1}^{(2)} \\ x_{k+1}^{(3)} \\ x_{k+1}^{(4)} \\ x_{k+1}^{(5)} \end{pmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1/2 & 0 & 1/2 \end{bmatrix} \begin{pmatrix} x_k^{(1)} \\ x_k^{(2)} \\ x_k^{(3)} \\ x_k^{(4)} \\ x_k^{(5)} \end{pmatrix},$$

thus, the average is not preserved when there is a packet loss, however the consensus is still an invariant reachable subspace because the matrices are row-stochastic and have the asynchronous form as in 2.1. If in this case the finite set of matrices Σ are generated from a strongly connected feasible communication topology and all the other assumptions remains unalterable, then by invoking the theorems 2.2 and 3.9 consensus is reached.

Note that if only messages type 1 are used in the protocol, the finite set Σ only contains asynchronous matrices of the form 2.1. Therefore, this is consistent with our framework of asynchronous updates, however it only achieves the consensus goal when faces packet losses and multi-rate phenomena.

It is important to highlight that average consensus cannot be reached with this approach due to packet losses. There are mechanisms that try to make the gossip process "reliable" in which the node that sends the type 1 message must wait the associated type 2 message. However these mechanisms have the dead-lock problem and produce slower convergence rate. Some of these mechanisms are treated in (Mehyar et al., 2005, 2007; Liu et al., 2018).

4.1.3. Broadcast-Gossip

This problem considers a group of N digital devices that can interact in a given strongly connected feasible communication graph that is constant over time, each agent i has a unique internal state $x_k^{(i)} \in \mathbb{R}$ at a certain time instant k. Every node has an internal clock whose frequencies and phases are not necessarily the same and therefore the multi-rate property is present. Once in a while thanks to its internal clock, the digital device j transmits its state to every out-neighbor $i \in \mathcal{V}_{out}^{(j)}$, which detect an event of data reception that allow to update their internal state as a convex linear combination with positive coefficients between the own state and the state received from j. The final goal is to reach consensus, however it has been proven that under certain statistical properties that the average is reached in expected value (Aysal, Yildiz, et al., 2009; Aysal et al., 2008).

The algorithm configured on each i agent is the following, where $x^{(i)}$ is the state variable of node i and $x^{(j)}$ is the state received from the in-neighbor j:

- Initialization:

$$x^{(i)} = m_i$$

- If node i generates an timer event: then it sends its state to all its out-neighbors.
- If node i detects a data reception event: then update its state as follows:

$$x^{(i)} \leftarrow \alpha x^{(i)} + (1-\alpha)x^{(j)},$$

where $0 < \alpha < 1$.

Since there are not packet losses nor delays, the update of the states if the timer of node j triggers at instant k and $i \in \mathcal{V}_j^{out}$ are as follows:

$$\begin{aligned} x_{k+1}^{(i)} &= \alpha x_k^{(i)} + (1-\alpha) x_k^{(j)} \\ x_{k+1}^{(j)} &= x_k^{(j)} \\ x_{k+1}^{(r)} &= x_k^{(r)}, \end{aligned}$$

where $r \in \mathcal{V}$ is a node which does not receive information, and therefore the update of the system state $x_{k+1} \in \mathbb{R}^N$ can be expressed as an iteration made by a transition matrix. For instance, the case when there are $N = 5$ nodes, the timer of the agent $j = 2$ triggers and its out-neighbors are $j = \{1, 4, 5\}$, generates the following update:

$$\begin{pmatrix} x_{k+1}^{(1)} \\ x_{k+1}^{(2)} \\ x_{k+1}^{(3)} \\ x_{k+1}^{(4)} \\ x_{k+1}^{(5)} \end{pmatrix} = \begin{bmatrix} \alpha & (1-\alpha) & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & (1-\alpha) & 0 & \alpha & 0 \\ 0 & (1-\alpha) & 0 & 0 & \alpha \end{bmatrix} \begin{pmatrix} x_k^{(1)} \\ x_k^{(2)} \\ x_k^{(3)} \\ x_k^{(4)} \\ x_k^{(5)} \end{pmatrix},$$

whose underlying graph is depicted in the Figure 4.3.

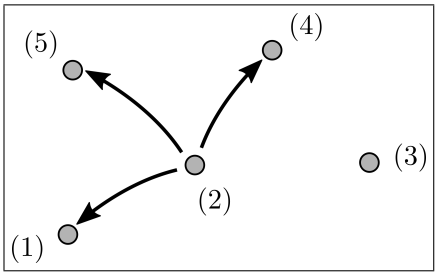

FIGURE 4.3. Example of the underlying graph induced by a matrix update of broadcast gossip protocol. Note that the self-loop was not added in node 2 because it does not update its state, however the induced matrix is positive in the a_{22} position. This synchronous update can be decomposed into three different asynchronous communications where the order does not matter.

Note that the iteration matrix of the above example is the result of the product of the three following matrices:

$$\begin{bmatrix} \alpha & (1-\alpha) & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \quad \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & (1-\alpha) & 0 & \alpha & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \quad \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & (1-\alpha) & 0 & 0 & \alpha \end{bmatrix},$$

independent of the order of the product, which is also true for the general case. Thus, the iteration matrices this algorithm induces can be though of as the asynchronous shown just above with the form of 2.1.

Since the set of matrices Σ is finite and taken from a strongly connected feasible communication topology, and also every matrix is row-stochastic with positive diagonal entries, and further, it can be regarded that the evolution of the system is an infinite product $(A_\infty \cdots A_1 A_0)$ of matrices taken from Σ in which every matrix in Σ is repeated infinitely often in the product. By invoking theorem 2.2, the infinite product $(A_\infty \cdots A_1 A_0)$ can by separated into finite products of matrices whose underlying graph is complete. Finally, by using theorem 3.9 the infinite product $(A_\infty \cdots A_1 A_0)$ converges to the consensus matrix of the form $\mathbb{1}\nu^T$.

Note that if there is a packet loss and since the iteration matrices can be regarded as asynchronous, it is as if an update never happens and therefore the induced iteration matrix can be regarded as the identity. With this in mind, it is clear that every matrix involved in this algorithm is row-stochastic even if there are packet losses. As long as all these asynchronous matrices are generated from a strongly connected feasible communication topology and they appear infinitely often in the infinite product, theorem 2.2 and 3.9 are still valid and therefore consensus is reached. However, even though in the findings (Aysal, Yildiz, et al., 2009; Aysal et al., 2008) the expected value of the consensus is in fact the average, there are not guaranties that for every realization of the algorithm the average is reached.

4.1.4. Push-Sum

This problem considers a group of N digital devices that can interact in a given strongly connected feasible communication graph that is constant over time. Each agent i has two internal states $z_k^{(i)} \in \mathbb{R}$ and $s_k^{(i)} \in \mathbb{R}$ at certain time instant k, which are updated under the same rule in parallel, and the estimated consensus value $x_k^{(i)}$ at each node is calculated by the division between the two internal states, i.e., $x_k^{(i)} = \frac{z_k^{(i)}}{s_k^{(i)}}$. Every node has an internal clock whose frequencies and phases are not necessarily the same and therefore the multi-rate property is present. Once in a while thanks to its internal clock, the digital device j updates its own states dividing by the number of out-neighbors plus itself $(1 + |\mathcal{V}_{out}^{(j)}|)$ and then transmits this updated states to every out-neighbor $i \in \mathcal{V}_{out}^{(j)}$ which consequently detect an event of data reception that allows to update their internal state as the sum between its actual states and the states received from j. The final goal is to reach the average consensus.

The algorithm configured on each agent i is the following, where $z^{(i)}$ and $s^{(i)}$ are the state variables of node i and $z^{(j)}$ and $s^{(j)}$ are the states received from the in-neighbor j:

- Initialization:

$$\begin{aligned} z^{(i)} &= m_i \\ s^{(i)} &= 1 \end{aligned}$$

- If node i generates a timer event:

  1) Update their own states as follows:

$$\begin{aligned} z^{(i)} &\leftarrow \tfrac{1}{1+|\mathcal{V}_i^{out}|}\, z^{(i)} \\ s^{(i)} &\leftarrow \tfrac{1}{1+|\mathcal{V}_i^{out}|}\, s^{(i)} \end{aligned}$$

  2) Transmit its updated states to every out-neighbor.

- If node i detects a data reception event from j: then update their states as follows:

$$\begin{aligned} z^{(i)} &\leftarrow z^{(i)} + z^{(j)} \\ s^{(i)} &\leftarrow s^{(i)} + s^{(j)} \end{aligned}$$

Since there are not packet losses nor delays, the update of the states if the timer of node j triggers at instant k and $i \in \mathcal{V}_j^{out}$, for the z variables is as follows:

$$\begin{aligned} z_{k+1}^{(i)} &= z_k^{(i)} + \tfrac{1}{1+|\mathcal{V}_j^{out}|}\, z_k^{(j)} \\ z_{k+1}^{(j)} &= \tfrac{1}{1+|\mathcal{V}_j^{out}|}\, z_k^{(j)} \\ z_{k+1}^{(r)} &= z_k^{(r)}, \end{aligned}$$

where $r \in \mathcal{V}$, $r \neq i$, $r \neq j$ is a node which does not performs any update, and the s variables have the same updates in parallel. Then the estimated consensus value on every node $x_k^{(i)}$ is determined by:

$$x_k^{(i)} = \frac{z_k^{(i)}}{s_k^{(i)}}.$$

Therefore, the updates of the system state $z_{k+1} \in \mathbb{R}^N$ and $s_{k+1} \in \mathbb{R}^N$ can be expressed as two iterations made by the same transition matrix. For instance, the case when there are $N = 5$ nodes, the timer of the agent $j = 3$ triggers and its out-neighbors are $j = \{1, 2, 5\}$, generates the following update for the s variables:

$$\begin{pmatrix} s_{k+1}^{(1)} \\ s_{k+1}^{(2)} \\ s_{k+1}^{(3)} \\ s_{k+1}^{(4)} \\ s_{k+1}^{(5)} \end{pmatrix} = \begin{bmatrix} 1 & 0 & 1/4 & 0 & 0 \\ 0 & 1 & 1/4 & 0 & 0 \\ 0 & 0 & 1/4 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1/4 & 0 & 1 \end{bmatrix} \begin{pmatrix} s_k^{(1)} \\ s_k^{(2)} \\ s_k^{(3)} \\ s_k^{(4)} \\ s_k^{(5)} \end{pmatrix},$$

whose underlying graph is depicted in the Figure 4.4.

Since the set of matrices Σ is finite and taken from a strongly connected feasible communication topology and also every matrix this time is column-stochastic with positive diagonal entries, and further, it can be regarded that the evolution of the system is an infinite product $(A_\infty \cdots A_1 A_0)$ of matrices taken from Σ in which every matrix in Σ is repeated infinitely often in the product. By invoking theorem 2.2, the infinite product $(A_\infty \cdots A_1 A_0)$ can by separated into finite products of matrices whose underlying graph is complete. Finally, by

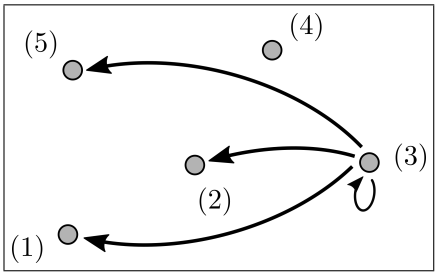

FIGURE 4.4. Example of the underlying graph induced by a matrix update of push-sum protocol. In this case it is assumed that all the updates occur at the same time, however they happen asynchronously. Note that the self loop is depicted since the timer event makes an update on the node.

using theorem 3.12 the infinite product $(A_\infty \cdots A_1 A_0)$ converges to a matrix of the form $\nu \mathbb{1}^T$.

Note that even when the z and s do not reach consensus independently, at the end all the estimated consensus variables $x_\infty^{(i)}$ reach the average. Indeed:

$$\begin{aligned} z_k &\rightarrow \nu \mathbb{1}^T z_0 = \left(\sum_{i=1}^{N} m_i \right) \nu \\ s_k &\rightarrow \nu \mathbb{1}^T \mathbb{1} = (N)\, \nu \end{aligned}$$

and therefore by using a division between two vectors as a component-wise division:

$$x_k = \frac{z_k}{s_k} \rightarrow \frac{\left(\sum_{i=1}^{N} m_i \right) \nu}{(N)\, \nu} = \left(\frac{1}{N} \sum_{i=1}^{N} m_i \right) \mathbb{1}$$

which means that all the states converge to average consensus.

Note that the matrices induced by this algorithm are column-stochastic. However, what would happen if there is a packet loss?. Let us say that in the previous example the $i = 2$

agent never listen the state from j, then the matrix would be the following:

$$\begin{pmatrix} s_{k+1}^{(1)} \\ s_{k+1}^{(2)} \\ s_{k+1}^{(3)} \\ s_{k+1}^{(4)} \\ s_{k+1}^{(5)} \end{pmatrix} = \begin{bmatrix} 1 & 0 & 1/4 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1/4 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1/4 & 0 & 1 \end{bmatrix} \begin{pmatrix} s_{k}^{(1)} \\ s_{k}^{(2)} \\ s_{k}^{(3)} \\ s_{k}^{(4)} \\ s_{k}^{(5)} \end{pmatrix}.$$

and thus the column-stochastic or conservation property does not hold anymore. Even if the communication channel is reliable, there is another problem related with the resolution of the numbers that can be stored in a digital device. What would happen if node $j = 3$ triggers its timer very fast, then the updates in the reliable above example when the update is repeated 4 times would be:

$$\begin{pmatrix} s_{k+1}^{(1)} \\ s_{k+1}^{(2)} \\ s_{k+1}^{(3)} \\ s_{k+1}^{(4)} \\ s_{k+1}^{(5)} \end{pmatrix} = \begin{bmatrix} 1 & 0 & 85/256 & 0 & 0 \\ 0 & 1 & 85/256 & 0 & 0 \\ 0 & 0 & 1/256 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 85/256 & 0 & 1 \end{bmatrix} \begin{pmatrix} s_{k}^{(1)} \\ s_{k}^{(2)} \\ s_{k}^{(3)} \\ s_{k}^{(4)} \\ s_{k}^{(5)} \end{pmatrix},$$

and the $s^{(3)}$ state would tend to zero, which is a problem since the estimated consensus value is calculated by a division. Even though the value would never reach zero theoretically, in a digital device the case is different since the maximum storage of a number. Thus, multi-rate should be avoided in this algorithm. Note also that the issue would persist in the case when a node could not detect information from its in-neighbors and therefore packet losses are harmful too.

4.2. Evaluation of Existing Multi-cast Algorithms

In this section an evaluation of the two multicast formulations is presented, they are the Broadcast-Gossip and the Push-Sum algorithms. The objective is to study how characteristic phenomena, such as multi-rate and unreliable communication, impact the properties of the algorithms in terms of accuracy and convergence to the average. Numerical simulations and

a hardware implementation in microprocessor-based development boards are presented as study case. We will see that in simulations and in the wireless testbed, the consensus value changes from the desired average due to packet losses. In general, by using just one internal value per agent is not possible to reach the average consensus. For further information about the simulation, the hardware code and some results, please refer to (Beorostica Github, 2018).

4.2.1. Numerical Simulations

Simulations were conducted using the HyEq Matlab-Simulink toolbox (Ricardo Sanfelice, 2017) to test the performance of both the broadcast gossip and the push sum algorithms. Two feasible communication topologies were considered, with $N = 6$: 1) an all-to-all network (five neighbors per agent), and 2) a sparse network where each agent has three neighbors. To account for possible heterogeneity of the agents, the timer frequencies were fixed according to four different scenarios: i) the agents have the same clock frequency (ideal case), ii) all timer frequencies have the same expected value with an error of 10% (similar to a real hardware implementation), iii) the agents with lower initial condition value have much lower constant frequencies, and iv) the agents with lower initial condition values have the higher constant frequencies (to simulate a possible failure or extreme heterogeneity in the network adapters). Moreover, to consider packet losses, different scenarios per topology were defined based on the value of the reception probability p. In fact, communication failures represent a source of inaccuracy that have been studied for the average consensus problem (Fagnani & Zampieri, 2006; Patterson et al., 2007).

For each simulation, the initial condition of the timer values are randomly chosen. The simulation finishes when the values among agents are sufficiently similar (a difference lower than 0.01). For all the simulations, the initial values of the initial states m_i are 10, 25, 40, 60, 75 and 90, respectively, which gives an average consensus value of $\bar{r} = 50$. For all cases, 100 realizations are executed per each reception probability. Finally, the chosen α parameter in the broadcast gossip algorithm is equal to $1/6$ since it gave the best performance in preliminary simulations.

Figure 4.5 shows plots of the consensus values reached by the agents at the end of simulation as a function of the reception probability, for the five neighbors topology and for each type of clock frequency considered. In all cases, the push-sum protocol has better performance with less dispersion and better accuracy for reception probabilities larger than 0.7. For the cases in which the timer frequencies have the same expected value (plots a) and b) of Figure 4.5), the dispersion of the consensus in both protocols are practically indistinguishable for reception probabilities lower than 0.7. It should be noted that, when the agents with lower initial measurements have lower frequencies (plotc c) of Figure 4.5), the expected consensus value in the broadcast gossip protocol is above the average and in push-sum the expected consensus decays below the average as packet losses increase. An analogous analysis can be made for plot d) of Figure 4.5.

Figure 4.6 shows plots of the consensus values as a function of the reception probability, for the three neighbors topology and for each type of clock frequency considered. As in Figure 4.5, the accuracy and dispersion of consensus values have a similar behavior despite the fact that the graph connectivity is different, which suggests that the qualitative performance does not depend on the particular network graph.

Table 4.1 summarizes the simulation results in terms of the Mean Absolute Error (MAE) and the Root Mean Squared Error (RMSE) indicators for clock frequencies with same expected value and an error of 10%. The MAE indicator is defined as:

$$MAE := \frac{1}{U}\sum_{u=1}^{U}|x_u - \bar{x}| \tag{4.1}$$

and the RMSE as:

$$RMSE := \sqrt{\frac{1}{U}\sum_{u=1}^{U}|x_u - \bar{x}|^2} \quad , \tag{4.2}$$

where all consensus values of the six agents are considered for calculation purposes, thus U is equal to number of simulations multiplied by $|\mathcal{V}|$.

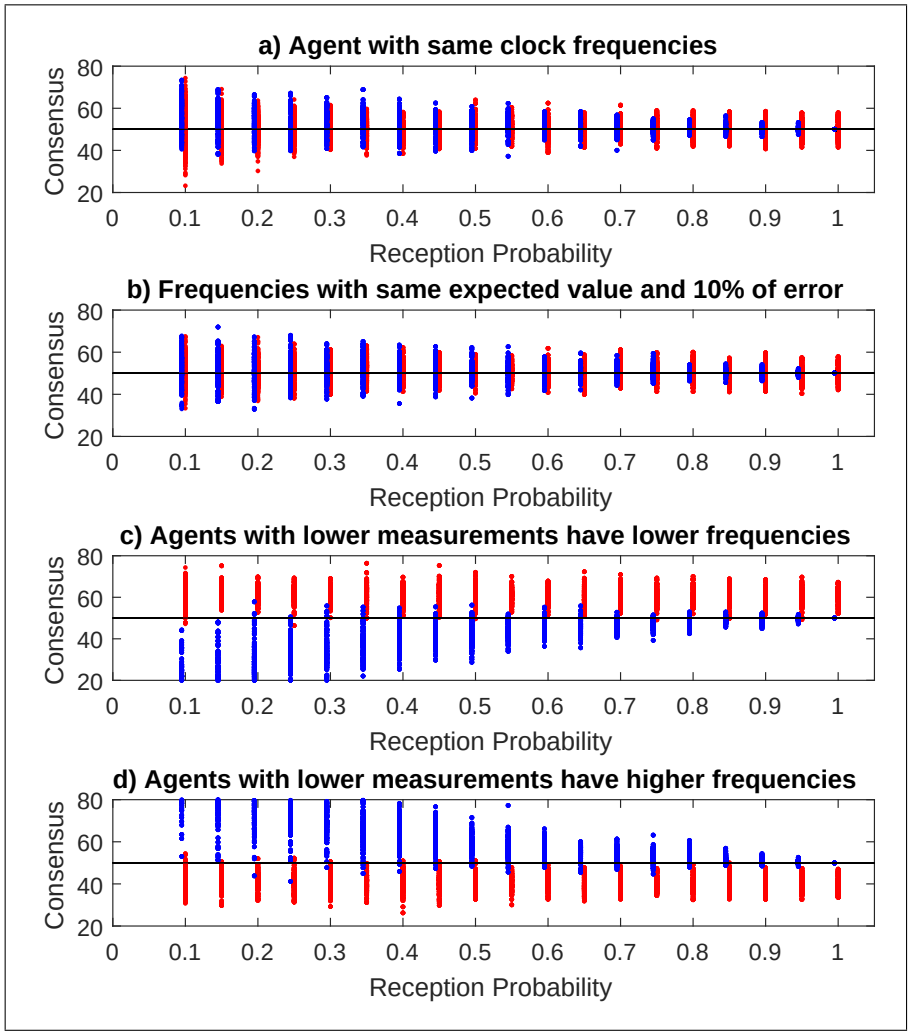

FIGURE 4.5. Consensus value vs. reception probability for 5-neighbors topology. 100 simulations for each reception probability. (Red: broadcast gossip. Blue: push-sum).

As with Figures 4.5 and 4.6, similar conclusions can be drawn from Table 4.1. It is clear that the push-sum algorithm has better performance with less MAE and RMSE within the receive probability interval between 0.6 and 1. It should be noted that without packet losses, the push-sum converges to the average value irrespective of differences in the clock frequencies, unlike the broadcast gossip that presents an error in all cases. When the reception probability is lower than 0.5, the broadcast gossip has an error lower than push-sum; however, these cases are not of practical interest since in most wireless channels the reception probability is close to 0.9 (Lee & Chanson, 2002).

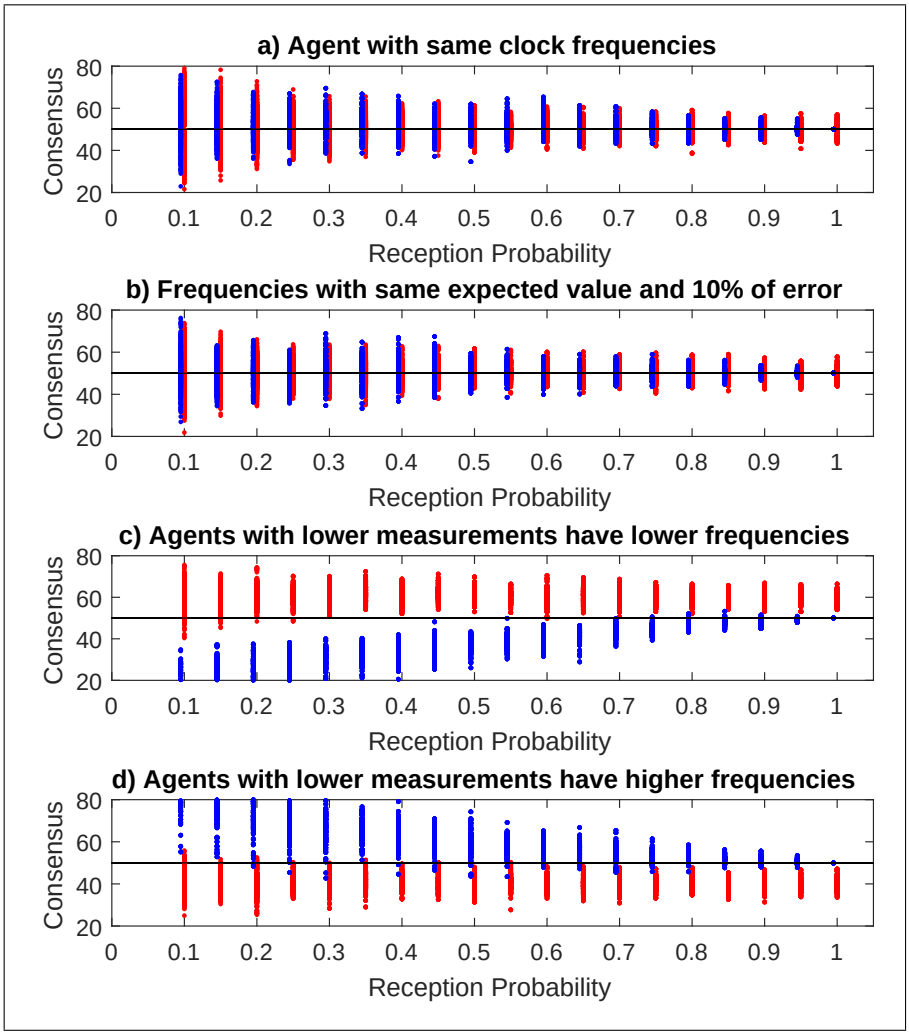

FIGURE 4.6. Consensus value vs. reception probability for 3-neighbors topology. 100 simulations for each reception probability. (Red: broadcast gossip. Blue: push-sum).

4.2.2. Hardware Implementation

To test the performance of the consensus algorithms in a real environment, they were implemented in a testbed consisting of 6 BeagleBone Black (BBB) development boards (BeagleBoard.org Foundation, 2017a). The BBB board is a low-cost, fully open-source, community-supported development platform, powered by a Texas Instruments Sitara AM3358 ARM Cortex-A8 processor running at 1 GHz, with 4 GB of onboard flash memory, 512 MB

TABLE 4.1. Simulation results for clock frequencies with same expected value and 10% error.

Top.	Prob. Rx	broadcast gossip		push-sum	
		MAE	RMSE	MAE	RMSE
5-neighbors	1.0	3.19	3.78	3.6e−14	4.2e−14
	0.9	3.60	4.23	1.03	1.37
	0.8	3.37	4.19	1.40	1.74
	0.7	3.21	3.90	2.16	2.72
	0.6	3.22	3.95	2.73	3.40
	0.5	3.26	4.07	3.17	4.07
	0.4	3.43	4.39	4.20	5.22
	0.3	4.33	5.11	4.92	5.96
3-neighbors	1.0	2.63	3.18	2.5e−9	3.5e−9
	0.9	2.85	3.46	1.18	1.44
	0.8	2.86	3.58	2.07	2.64
	0.7	2.93	3.62	2.35	2.93
	0.6	3.11	3.85	2.98	3.70
	0.5	3.61	4.47	3.49	4.23
	0.4	3.39	4.31	4.97	6.06
	0.3	3.74	4.70	5.61	7.15

of DDR3L DRAM, and a 3-D graphics accelerator. The BBB supports several Linux distributions, Debian, Angstrom, and Ubuntu among them. For the evaluation, Debian was selected as operating system. To enable WiFi communication, a TP-Link TL-WN722N USB dongle is attached to each board to create a IEEE802.11n ad-hoc network.

The protocols were coded in user space at the application layer using internal timers to trigger the transmissions periodically. The state values of the agents are sent via UDP datagrams that are multicasted after a timer event. The Cloud9 IDE programming environment on Node.js language is used for these purposes.

In the hardware implementation the agents have the same initial conditions as in the numerical simulations. The sensing event that starts the protocols consists in a UDP packet sent by an external computer able to reach every node in the network. The stop condition of the algorithms is triggered in each agent when an stop signal transmitted by the external computer is received by every node. The explicit protocols are the same as the shown in the past section.

Figure 4.7 presents the results of the hardware implementation for the five neighbors per agent topology. A total of 20 executions of the protocols are performed. MAE and RMSE indicators are calculated to evaluate accuracy of the broadcast gossip and push-sum algorithms.

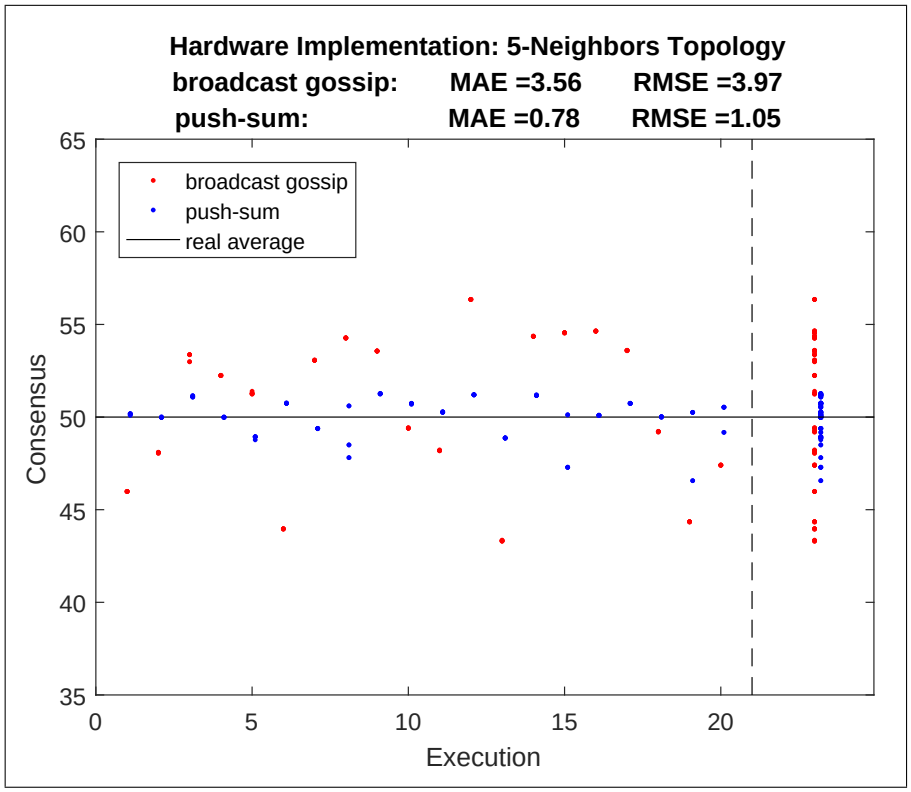

FIGURE 4.7. Results of the implementation in BBB development boards.

Comparing against the results obtained in the numerical simulations, it can be seen that the same expected timer frequencies for each agent and a reception probability of approximately 90% represents what happens in the real environment. Comparing the Table 4.1 and indicators shown in Figure 4.7, the model proposed and its simulation executions agree with the hardware implementation results, both have similar MAE and RMSE values and the qualitative behavior resembles plot three of Figure 4.5 at 0.9 reception probability.

Finally, it can be seen in Figure 4.7 that in some realizations not all agents converge close enough (the plot contains six consensus values for each realization and protocol), this is due to the time was not enough at least for the broadcast gossip protocol that should reach the

consensus, in contrast to the pus-sum protocol which not necessarily should reach it. Note that the push-sum algorithm should have an agreement period as short as possible since z_i and s_i variables approach zero due to packet losses (intuitively, the conservation property is not longer valid). The broadcast gossip algorithm does not present this problem since the agent that sends the data does not update its own state.

4.2.3. Observations

The performance evaluation of two average consensus algorithms was shown: broadcast gossip and push-sum, in pure-broadcasting infrastructure-free networks.

Asynchronous data transfer and unreliable communication are main issues that are confronted in numerical simulations to test average consensus convergence. In all cases where the probability of reception has a practical value, the push-sum protocol has better qualitative performance with better accuracy and less dispersion, also the MAE and RMSE indicators reveal the superiority of this algorithm over the broadcast gossip algorithm. It is emphasized that push-sum always reaches the average as consensus value in the absence of packet losses. However, the push-sum algorithm presents the extra requirement of knowing the cardinality of the out-neighbor set $|\mathcal{V}_{out}^{(i)}|$.

A testbed in real hardware was implemented to check the performance of the algorithms in a real environment. The results obtained for both protocols are consistent with the numerical simulations for a given practical reception probability. Although push-sum shows better performance, it needs to perform the algorithm in a shorter period of time since the variable z_i and s_i approach to zero due to packet losses, unlike broadcast gossip that does not require this practical consideration.

We have seen that one of the main reasons why the average consensus is not reached is because the preservation property is not invariant over time and this happens because the packet losses phenomenon. There have been some recent findings (Chen, Tron, Terzis, & Vidal, 2010, 2011; Bof, Carli, & Schenato, 2017) which proposes to add extra variables that store the quantities exchanged to use when packet losses occur. An extensive treatment of

this approach will be discussed in the following chapter in which the ideas of these authors are used to develop a new algorithm to solve the distributed the average consensus problem.

Chapter 5. A NEW ALGORITHM FOR DISTRIBUTED AVERAGING

We have seen that classical approaches for distributed averaging use quite strong assumptions which are not valid in our framework. Many of the analysis made was only possible because the formulations of the problem regard synchronous transmissions and receptions or do not take packet losses into account. We know that there are other non-ideal issues when the formulation problem faces transmissions delays and data quantization and, even though some authors cope with these facts (Olfati-Saber & Murray, 2004; Blondel et al., 2005; Sun et al., 2008; Bliman & Ferrari-Trecate, 2008; Tsianos & Rabbat, 2011; Kashyap et al., 2006; Kar & Moura, 2007b; Frasca et al., 2008; Nedic et al., 2009; Lavaei & Murray, 2012), the analysis of stability and convergence rate is really a difficult task.

The latest state-of-the-art findings can face the distributed average consensus problem regarding asynchronism and packet losses in order to prove stability and convergence rate and give convincing simulation results(Chen et al., 2010, 2011; Bof et al., 2017) . However the analysis in presence of delays and quantization is yet to be developed. Even the protocol that we will expose in this thesis will not treat formally these two latter non-ideal issues, but it will show the success of reaching averaging in a real IoT environment and it will give theoretical insights dealing with asynchronism and packet losses.

In this chapter, the most important contribution of this thesis will be presented. It is a new protocol based on gossip data exchanges and the use of corrective variables that store information in case of packet losses. It will be explained the way in which the algorithm was designed, with focus on the principles of its operation facing asynchronism and data losses and even some degree of robustness to delays. Since the logic of the algorithm relies on that eventually every node listen to all its local neighbors and that exists a convergence property, two versions of the algorithm could be formulated: a unicast version in which a node must select a neighbor and transmit information, once it is received the receiver updates and sends back another packet to finally the first node updates; and a multicast version, in which a node simply transmit to all its neighbors information and the receiver updates in case it successfully detects an event of data reception.

5.1. Ideas behind the proposed algorithm

Let us try to solve the average consensus problem from an intuitive point of view. The problem regards a set of agents that can communicate without a predefined order and somehow they need to reach a consensus but not to any value, it must be the average of their initial states. One way to think in the solution of this problem is to imagine that each agent as a vessel of equal dimensions but each one contains a certain (probably different) quantity of water at the beginning and they can transfer water to other vessels thanks to pipes on the ground that connect all the vessels. Assuming that the water in the pipes is negligible and at the beginning the pipes are closed by a valve, it is clear that once the valve is open every vessel will have the same height of water after a certain time and this value must be the average of the original heights because the quantity of water was preserved in the process. The situation is somewhat depicted in Figure 5.1.

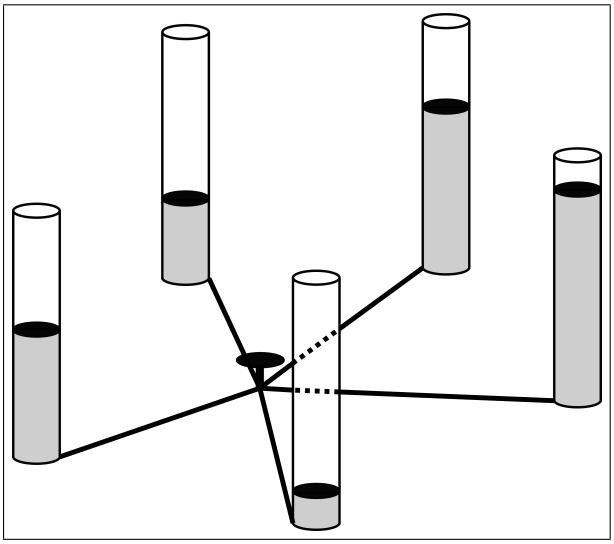

FIGURE 5.1. Communicant vessels with a complete underlying graph every time the valve is opened. As long as the valve is opened and closed indefinitely, every vessel will reach the same height corresponding to the average of the initial states.

This process can be changed a little and equally can solve the average consensus problem. This time we could open and close the valve intermittently without waiting for the complete establishment of the heights, every time the valve is open the heights are, in a way, nearer (closer to the consensus) and the water that is lost in some vessels are added into other vessels

(total water is preserved). The same Figure 5.1 illustrates the situation. Note that once the average consensus is reached, heights cannot be changed by the valve. Therefore, it is natural to think that any discrete step of a solution for the average consensus problem should have a conservation property ($\mathbb{1}^T A_k = \mathbb{1}^T$) and also every step should recognize that if the average consensus is reached then the state should be held ($A_k \mathbb{1} = \mathbb{1}$).

The aforementioned processes only considers one valve that connects or disconnects all the agents at the same time. We could join with pipes subsets of vessels with one valve, and open and close the different valves intermittently without any predefined order and as long as all the valves are being opened indefinitely (every A_k appear infinitely often in the product) and if the pipes allow that all the vessels are connected when all the valves are open (the feasible communication topology is strongly connected) then the average consensus will be reached naturally. The Figure 5.2 illustrates the situation.

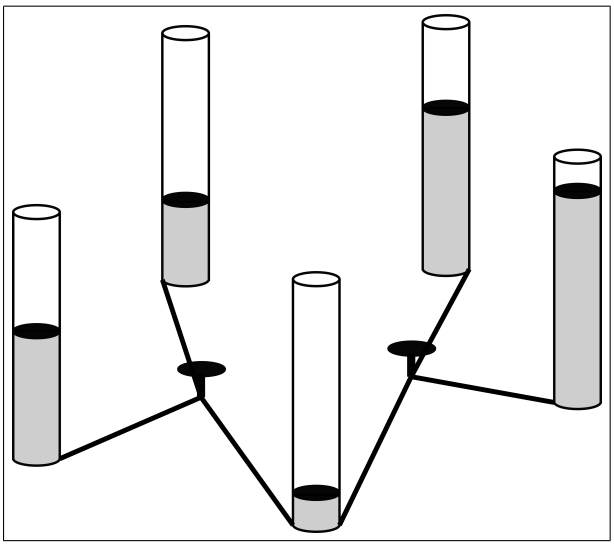

FIGURE 5.2. Communicant vessels with two different underlying graphs generated from a strongly connected feasible communication topology. As long as both valves are being opened and closed indefinitely the height of the vessels naturally will reach the average consensus.

However the above solution is assuming that in the communication many neighbors are involved in the exchange of information, and in an asynchronous process only two agents can communicate at certain instant. Therefore we should put pipes with valves between every two

pair of nodes that can communicate and again if the pipes can connect all the vessels when the valves are open and if the all the valves are opened and closed indefinitely then logically the average consensus will be reached eventually. Figure 5.3 illustrates the situation. Clearly, this process is what gossip algorithms do in which one half of the total water of the two vessels is transferred from the vessel with more water to the vessel with less water when a valve is opened, thus one half of water is added into one agent and another half is removed from the another involved node.

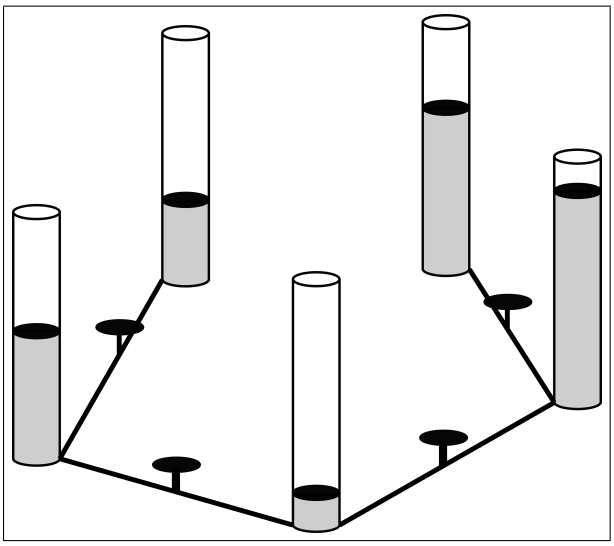

FIGURE 5.3. Communicant vessels with gossip underlying graphs generated from a strongly connected feasible communication topology. As long as both valves are being opened and closed indefinitely the height of the vessels naturally will reach the average consensus.

Note that the update of the water heights is performed thanks to continuous laws (gravity forces and mass conservation) that are enabled by the valves. Because of that, doing an exchange of water between two agents can be done at once, however in a communication process with digital devices it is not possible to do an exchange of information at once, just is possible to emulate this process. The way would be the following, when the timer of a node expires then it sends the information of its quantity of water to another node, when the second agent receives the information it can remove or add the half of water in its state and then sends back the information to the first node and finally the last one can properly add or

remove the half of water in order to preserve the quantity of water as it is illustrated in Figure 5.4. The problem here is that sometimes there are packet losses and if the second message is lost, then some quantity of water will be added or removed and therefore the initial quantity of water will be lost.

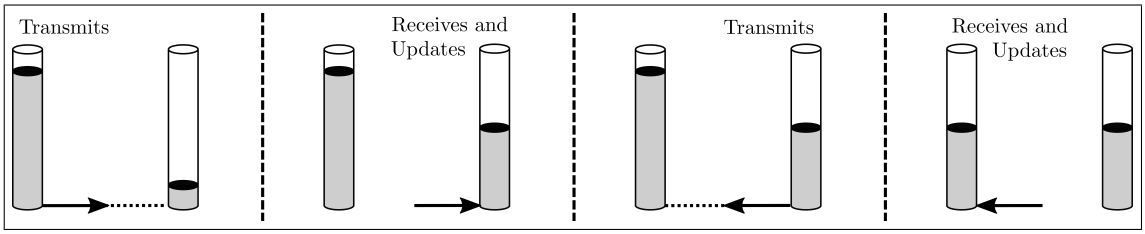

FIGURE 5.4. Digital devices emulating the communication vessel process by using a transmission-reception process. The ideal behavior of the gossip approach is not possible due to the unreliability of the communication channel.

Note that in this explanation it does not matter if the quantity of water is positive or negative, intuitively the average consensus will be equally reached too. However the analogy of the communicating vessels is quite straightforward and clearly shows the desired average consensus result. Because of this, we rather talk about another analogy that has a conservation property, in this new situation we regard a group of particles with unit mass in the free space with certain quantity of momentum at the beginning and then, regardless where the particles are, collisions are induced intermittently and indefinitely, in this system the quantity of momentum is preserved every time a collision is induced and if in each collision the velocities are closer, then average consensus will be reached. The situation is depicted in Figure 5.5. In this case if we use digital devices as agents the total quantity of momentum will be lost too in the presence packet losses.

Therefore it is necessary to preserve the momentum every time there is a change of the state which is equivalent to say that in a linear update every column of the transition matrix induced sums one ($\mathbb{1}^T A_k = \mathbb{1}^T$). The first time when the i agent sends its state to another j neighbor there is not update at all, thus if the packet is lost then the momentum is not altered and there is no problem, just left to wait the next timer event. If the first message is successful, then the second agent j received the state from the first one i and node j changes its state with

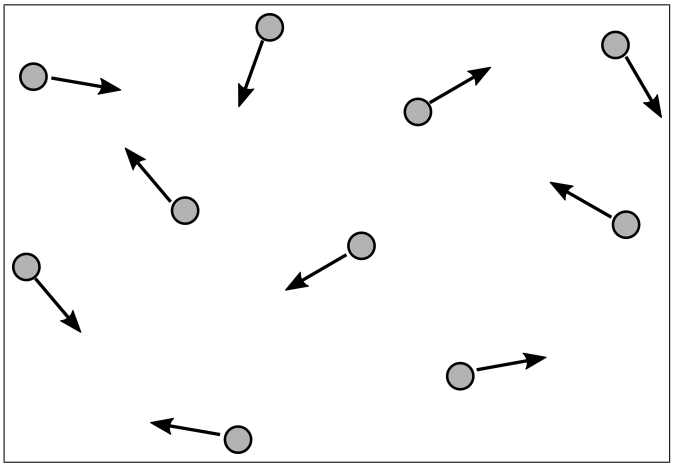

FIGURE 5.5. A particle system in which all agents have the same mass, the momentum is preserved and collisions are induced between pairs of agents also serves as an analogy of the gossip approach where naturally all the momentums will converge to the momentum of the center of mass or the desired average.

the form of equation 2.1 which represents an asynchronous update and is once again written below:

$$\begin{pmatrix} x_{k+1}^{(1)} \\ \vdots \\ x_{k+1}^{(i)} \\ \vdots \\ x_{k+1}^{(j)} \\ \vdots \\ x_{k+1}^{(N)} \end{pmatrix} = \begin{bmatrix} 1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \cdots & a_{ii} & \cdots & a_{ij} & \cdots & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 1 & \cdots & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & 1 \end{bmatrix} \begin{pmatrix} x_{k}^{(1)} \\ \vdots \\ x_{k}^{(i)} \\ \vdots \\ x_{k}^{(j)} \\ \vdots \\ x_{k}^{(N)} \end{pmatrix},$$

however the only way, in order to preserve the average, is that $a_{ii} = 1$ and $a_{ij} = 0$, i.e., there is no update at all. According to this, it would not be possible to preserve the momentum with linear asynchronous updates. However we can add internal states, some of them will represent the momentum that the agent possesses as always and the added states will serve to preserve the mass at each iteration. Thus, the linear asynchronous update will have the

following form of the expression 2.2:

$$\begin{pmatrix} \vec{x}_{k+1}^{(1)} \\ \vdots \\ \vec{x}_{k+1}^{(i)} \\ \vdots \\ \vec{x}_{k+1}^{(j)} \\ \vdots \\ \vec{x}_{k+1}^{(N)} \end{pmatrix} = \begin{bmatrix} I & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \cdots & A_{ii} & \cdots & A_{ij} & \cdots & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & I & \cdots & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & I \end{bmatrix} \begin{pmatrix} \vec{x}_{k}^{(1)} \\ \vdots \\ \vec{x}_{k}^{(i)} \\ \vdots \\ \vec{x}_{k}^{(j)} \\ \vdots \\ \vec{x}_{k}^{(N)} \end{pmatrix}$$

here we can see that it is possible to preserve the momentum, however, if we want an update different from the identity matrix it is necessary to add some negative coefficients, which is something that is not good because the theory of non-homogeneous products of matrices is developed mainly with non-negative matrices, even worst, here the idea of passivity is not longer valid because in order to preserve the momentum some quantities must be created with positive sign and others with negative sign in order to compensate the asynchronous phenomenon.

The new variables added to the algorithm are based on the ideas of previous findings (Chen et al., 2010, 2011; Bof et al., 2017) and, in order to understand the logic of them, first let us introduce the states involved. Every agent i stores three types of variables that evolve over time:

$$z^{(i)}\,, \quad \Sigma\Delta z^{(ij)}\,, \quad \hat{\Sigma}\Delta z^{(ji)},$$

for all $j \in \mathcal{V}^{(i)}$, where $\mathcal{V}^{(i)}$ is the neighbor set of agent i. We will soon see that the communication needs to be bidirectional in order to exchange momentum between two agents, and therefore it is not necessary to distinguish between $\mathcal{V}_{out}^{(i)}$ and $\mathcal{V}_{in}^{(i)}$, however we are going to keep the in-out notation of the neighbor sets in some cases to make it clear the distinction for future implementations. Also note that the dimension of an agent state depends on the cardinality of the neighbor set, so different agents possibly store a different number of variables, and in general if the number of agents in the network N is large we expect that the number of neighbors for each node $|\mathcal{V}^{(i)}|$ be much smaller in the design $|\mathcal{V}^{(i)}| \ll N$ in order to privilege

the distributed nature of the solution. The aforementioned variables represent the following:

$z^{(i)}$	:	the momentum of the agent.
$\Sigma\Delta z^{(ij)}$	:	the total momentum agent i loses because all the collisions that node k induced over i.
$\hat{\Sigma}\Delta z^{(ji)}$	:	the total momentum that i gains because all the collisions that node i believes that induced over agent k.

In order to understand why these variables represent the above quantities, we have to describe the updates done during the algorithm. The following section will describe the process. We will see that due to the preservation property of this algorithm, and thanks to an unravel convergence property, this algorithm can be implemented in two versions: uni-cast or multi-cast. The convergence analysis in these two versions is the same so first we are going explain the uni-cast process in order to understand why this algorithm works.

5.2. Communication Process and Qualitative Analysis

5.2.1. Uni-cast Version

The ideal flow of the algorithm in the uni-cast version is as follows: when the timer of agent i expires, it sends data to a selected agent j, which upon reception updates its state and sends back information to i, finally if agent i receives that response, it updates its internal state. In terms of events, the flow is the following: if a timer event occurs in agent i, then it sends its state to agent j, as long as agent j detects the event of data reception it updates its state with the received information and as fast as possible it sends information to agent i, then if agent i detects an event of data reception in response to its first message, it updates its own state. Note that this communication sequence is possible only if the feasible communication topology is bidirectional.

If agent i generates a timer event then it sends the internal variables $z^{(i)}$, $\Sigma\Delta z^{(ij)}$, $\hat{\Sigma}\Delta z^{(ji)}$ to a selected agent j. If agent j detects its associated event of data reception, then it creates

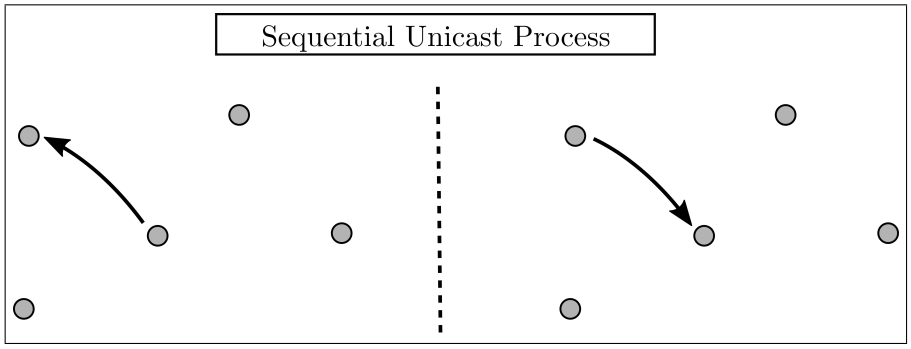

FIGURE 5.6. Unicast version of the protocol. An agent must transmit a message type 1 to a selected neighbor. If the neighbor receives then it updates its state and sends back a message type 2. Finally, if the first node listen then updates its state and the process ends. Note that if a message is lost, the conservation property is still preserved.

auxiliary variables that represent the corrected momentums due to changes of state between the agents i and j:

$$z_{corr}^{(j)} = z^{(j)} + \left(\Sigma\Delta z^{(ij)} - \hat{\Sigma}\Delta z^{(ij)}\right) \tag{5.1}$$

$$z_{corr}^{(i)} = z^{(i)} + \left(\Sigma\Delta z^{(ji)} - \hat{\Sigma}\Delta z^{(ji)}\right), \tag{5.2}$$

and also node j creates a manipulated variable $\Delta z^{(ji)}$ defined as:

$$\Delta z^{(ji)} = \frac{1}{2}\left(z_{corr}^{(j)} - z_{corr}^{(i)}\right)$$

Then, agent j updates its internal variables as follows:

$$z_{next}^{(j)} = z_{corr}^{(j)} - \Delta z^{(ji)} \tag{5.3}$$

$$\Sigma\Delta z_{next}^{(ji)} = \Sigma\Delta z^{(ji)} + \Delta z^{(ji)} \tag{5.4}$$

$$\hat{\Sigma}\Delta z_{next}^{(ij)} = \Sigma\Delta z^{(ij)} \tag{5.5}$$

and transmits back a message with the variables $\Delta z^{(ji)}$ and $\Sigma\Delta z^{(ji)}$ (values not updated). If the agent i detects its associated event of data reception, then it corrects its momentum due to i-j communications:

$$z_{corr}^{(i)} = z^{(i)} + \left(\Sigma\Delta z^{(ji)} - \hat{\Sigma}\Delta z^{(ji)}\right).$$

Then i updates its internal state as:

$$\begin{aligned} z_{next}^{(i)} &= z_{corr}^{(i)} + \Delta z^{(ji)} \\ \Sigma\Delta z_{next}^{(ij)} &= \Sigma\Delta z^{(ij)} \\ \hat{\Sigma}\Delta z_{next}^{(ji)} &= \Sigma\Delta z^{(ji)} + \Delta z^{(ji)} \end{aligned}$$

Note that the idea behind this algorithm is to induce "gossip collisions" between two agents such that the momentums of both agents after the collision will be the same and the total momentum remains constant, just like in the gossip approach. In fact, it is easy to see that if the communication channel is reliable, there is no need to add the variables $\Sigma\Delta z$ and $\hat{\Sigma}\Delta z$ and neither is necessary to make the corrections of the momentums and the variation of the momentum would be:

$$\Delta z^{(ji)} = \frac{1}{2}\left(z^{(j)} - z^{(i)}\right),$$

and the updates of the agents involved in the collision induced by i would be:

$$\begin{aligned} z_{next}^{(j)} &= z^{(j)} - \Delta z^{(ji)} \\ z_{next}^{(i)} &= z^{(i)} + \Delta z^{(ji)}, \end{aligned}$$

where is clear that $z_{next}^{(i)} = z_{next}^{(j)} = \frac{1}{2}\left(z^{(i)} + z^{(j)}\right)$ which is exactly what the gossip approach does. Note that from here we could change the $\frac{1}{2}$ factor in the manipulated variable $\Delta z^{(ji)}$ by one between zero and one and the algorithm would converge to the average, however, as we have discussed before, if two consecutive updates of the same two nodes happens then the momentums will change again and if just this update occurs indefinitely then eventually the nodes will converge to the average of the momentums of these two nodes. Since it is not logical that the state changes if the two agents do not have new external information, we will set the $\frac{1}{2}$ factor in the design of the algorithm.

However, since there are packet losses the variables $\Sigma\Delta z$ and $\hat{\Sigma}\Delta z$ are necessary in order to preserve the total momentum of the system. In fact, note that separately the two updates

preserve a quantity invariant. If the first update of the agent j happens, then:

$$\begin{aligned} z_{next}^{(j)} + \left(\Sigma\Delta z_{next}^{(ji)} - \hat{\Sigma}\Delta z_{next}^{(ij)}\right) &= z_{corr}^{(j)} + \Sigma\Delta z^{(ji)} - \Sigma\Delta z^{(ij)} \\ &= z^{(j)} + \left(\Sigma\Delta z^{(ji)} - \hat{\Sigma}\Delta z^{(ij)}\right), \end{aligned}$$

and something equivalent occurs with the second update of the agent i:

$$\begin{aligned} z_{next}^{(i)} + \left(\Sigma\Delta z_{next}^{(ij)} - \hat{\Sigma}\Delta z_{next}^{(ji)}\right) &= z_{corr}^{(i)} + \Sigma\Delta z^{(ij)} - \Sigma\Delta z^{(ji)} \\ &= z^{(i)} + \left(\Sigma\Delta z^{(ij)} - \hat{\Sigma}\Delta z^{(ji)}\right), \end{aligned}$$

which, by the way, is regardless the value of the manipulated variable $\Delta z^{(ji)}$. This fact is what makes work the conservation property whatever is the listened value, and therefore intuitively the algorithm should have a certain degree of robustness to delays, however the convergence is not a clear property assuming or not the delay phenomena.

5.2.2. Multi-cast Version

The multi-cast version appears as a variation of the uni-cast version. The flow in the multi-cast version is different from the former one in two main terms: first the communication process does not try to generate "gossip collisions" immediately, instead pairs of nodes rely on the fact that eventually they will receive information from each other, so effective "gossips collision" will occur eventually, this is possible because intuitively every time there is an update there is a natural local convergence property and conservation property; and second, the information can be sent in a multi-cast way since it is not necessary to rely on the second reply message.

The ideal flow of the algorithm in the multi-cast version is as follows: when the timer of agent i expires (a timer event), it sends data to every neighbor which upon reception (a data reception event) update their states. Note that probably the receptions will not occur at the same time, also is necessary that the feasible communication topology be bidirectional to "gossip collisions" happen eventually.

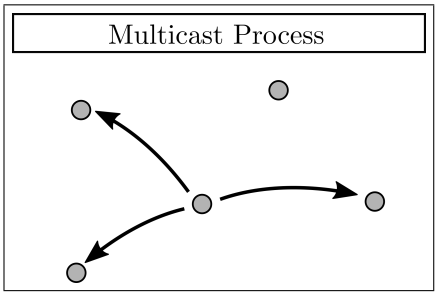

FIGURE 5.7. Multicast version of the protocol. An agent must transmit a message to all its neighbors. If they receive, then update their states. Note that if a message is lost, the conservation property is still preserved.

The updates are the same as in the uni-cast approach but they just regards the first message. To make it clearer, we will show explicitly the updates. If agent i generates a timer event, then it sends the internal variables $z^{(i)}$, $\Sigma\Delta z^{(ij)}$, $\hat{\Sigma}\Delta z^{(ji)}$ to all its neighbors. If neighbor j detects its associated event of data reception, then it creates auxiliary variables that represent the corrected momentums:

$$\begin{aligned} z_{corr}^{(j)} &= z^{(j)} + \left(\Sigma\Delta z^{(ij)} - \hat{\Sigma}\Delta z^{(ij)}\right) \\ z_{corr}^{(i)} &= z^{(i)} + \left(\Sigma\Delta z^{(ji)} - \hat{\Sigma}\Delta z^{(ji)}\right), \end{aligned}$$

and also node j creates the manipulated variable $\Delta z^{(ji)}$ defined as:

$$\Delta z^{(ji)} = \frac{1}{2}\left(z_{corr}^{(j)} - z_{corr}^{(i)}\right)$$

Then, agent j updates its internal variables as follows:

$$\begin{aligned} z_{next}^{(j)} &= z_{corr}^{(j)} - \Delta z^{(ji)} \\ \Sigma\Delta z_{next}^{(ji)} &= \Sigma\Delta z^{(ji)} + \Delta z^{(ji)} \\ \hat{\Sigma}\Delta z_{next}^{(ij)} &= \Sigma\Delta z^{(ij)} \end{aligned}$$

And as in the uni-cast version, the update preserves a quantity invariant:

$$\begin{aligned} z_{next}^{(j)} + \left(\Sigma\Delta z_{next}^{(ji)} - \hat{\Sigma}\Delta z_{next}^{(ij)}\right) &= z_{corr}^{(j)} + \Sigma\Delta z^{(ji)} - \Sigma\Delta z^{(ij)} \\ &= z^{(j)} + \left(\Sigma\Delta z^{(ji)} - \hat{\Sigma}\Delta z^{(ij)}\right). \end{aligned}$$

5.2.3. Qualitative Analysis

Note that regardless the value of the state read from a message, the preservation property is invariant. Therefore, if there are delays the total quantity of momentum is equally preserved, however, convergence is not clear beforehand. The previous discussion showed that regardless the version of the algorithm the quantity:

$$z^{(i)} + \left(\Sigma\Delta z^{(ij)} - \hat{\Sigma}\Delta z^{(ji)}\right)$$

is invariant when node i updates its state. As all the other states do not change while this update happens, then the following quantity:

$$z^{(i)} + \sum_{j\in\mathcal{V}^{(i)}} \left(\Sigma\Delta z^{(ij)} - \hat{\Sigma}\Delta z^{(ji)}\right)$$

holds constant too, and even this is true whether or not there is an update at node i. And as this quantity is constant between steps, then it must be invariant since the very beginning. Thus, thanks to the initial states are set zero for the total momentums exchanged (variables $\Sigma\Delta z$ and $\hat{\Sigma}\Delta z^{(ji)}$) and m_i for the initial momentum of the agent (variables $z^{(i)}$), we have:

$$z^{(i)} + \sum_{j\in\mathcal{V}^{(i)}} \left(\Sigma\Delta z^{(ij)} - \hat{\Sigma}\Delta z^{(ji)}\right) = m_i \tag{5.6}$$

and by rearranging this equation:

$$z^{(i)} = m_i - \sum_{j\in\mathcal{V}^{(i)}} \left(\Sigma\Delta z^{(ij)}\right) + \sum_{j\in\mathcal{V}^{(i)}} \left(\hat{\Sigma}\Delta z^{(ji)}\right) \tag{5.7}$$

which means that the actual momentum of the agent is given by all the momentums gained minus all the momentums lost due to collisions with neighbors. Also, as the variables involved in the above expression just regard own states of the i agent, then it is possible that the

agents independently reset their states to the initial condition without any harm to the invariant conservation property. Furthermore, if an agent decides not to listen anymore a neighbor and had not wanted to exchange any momentum with it, this can be done setting to zero the corresponding variables $\Sigma\Delta z$ and $\hat{\Sigma}\Delta z^{(ji)}$ and updating the state with the above expression, and still the conservation property remains unalterable.

Since equation (5.6) implies that a certain quantity is invariant over time on each node, this fact must be true in the whole system. Thus, if we sum the equation (5.6) over every node in the network we have:

$$\sum_{i=1}^{N}\left[z^{(i)}+\sum_{j\in\mathcal{V}^{(i)}}\left(\Sigma\Delta z^{(ij)}-\hat{\Sigma}\Delta z^{(ji)}\right)\right]=\sum_{i=1}^{N}m_i$$

and, by rearranging terms of the double-sum and because communication links are bidirectional, it is equivalent to:

$$\sum_{i=1}^{N}z^{(i)}+\sum_{i=1}^{N}\sum_{j\in\mathcal{V}^{(i)}}\left(\Sigma\Delta z^{(ji)}-\hat{\Sigma}\Delta z^{(ji)}\right)=\sum_{i=1}^{N}m_i \tag{5.8}$$

Since the protocol always is trying that the momentums of the nodes are together after a collision, which are distributed in the whole network, and also that the errors of momentums exchanges are zero after an exchange, it is natural to expect that $z_\infty^i = z_\infty^j = z_\infty$ and that $\left(\Sigma\Delta z_\infty^{(ji)}-\hat{\Sigma}\Delta z_\infty^{(ji)}\right)=0$ for all nodes i and j. And therefore replacing the values in the global conservation equation 5.8:

$$\sum_{i=1}^{N}z_\infty=\sum_{i=1}^{N}m_i$$

or equivalently:

$$z_\infty=\frac{1}{N}\sum_{i=1}^{N}m_i, \tag{5.9}$$

which means that average consensus would be reached, however the intuition that the algorithm converges is yet to be proven.

5.3. A Deeper Analysis

First of all, we are going to define the complete state of the network x. The dimension of such state will depend on the feasible communication topology regarded. At this point it is important to highlight that the analysis from now on will assume that all the nodes are reliable, i.e., they are always running the algorithm once it begins, this is because if suddenly an agent disappears then at least two problems arise: first the definition of the average could change, indeed, which average we desire? the one that regards the lost node, or the one that does not; and secondly, a more immediate problem, the momentum that the agent gains or loses would be retained in the failed node so the average consensus would not be obtained beforehand. In particular, and since every node can get the actual momentum regarding the initial momentum and the exchange of momentums with its neighbors according to equation 5.7, we could add a mechanism in the algorithm to face the problem in which the desired average is the one that does not regard the faulty nodes, such mechanism would consist in that the node still alive would not regard the exchanged momentums with the failed node as long as the former one could not listen the defective node after a certain time. By the time being, we will assume that all the nodes are reliable and thus we will not have the above problems, and furthermore we will not need to change the dimensions of the state of the transition matrices of the updates.

For analysis, we will say that an agent i possesses the states: its momentum $z^{(i)}$, the total lost momentum due to its neighbors $\Sigma\Delta z^{(ij)}$, $\forall j \in \mathcal{V}^{(i)}$, and the total gained momentum due to its neighbors $\hat{\Sigma}\Delta z^{(ji)}$, $\forall j \in \mathcal{V}^{(i)}$. Since in principle the variables that represent the exchange of momentum could have different convergence values at the end, it should be desirable not to work directly with them, instead we will use "error" variables that should converge to zero as time tend to infinity.

Let's define the following error variable:

$$e^{(ij)} := \Sigma\Delta z^{(ij)} - \hat{\Sigma}\Delta z^{(ij)}, \tag{5.10}$$

this variable represent the error that should be added to the state z^j in order to correct it due to i-j communications according to the equations 5.1 and 5.6. Note that the error variable is zero if the message type 2 is successfully received in the uni-cast version or if the consecutive communications ($i \rightarrow j$, $j \rightarrow i$) or ($j \rightarrow i$, $i \rightarrow j$) happens. Also note that in the error definition the $\Sigma\Delta z^{(ij)}$ variable is from agent i, and the $\hat{\Sigma}\Delta z^{(ij)}$ variable is from agent j, so here we are combining the original states in order to represent something like an state of the communication link between two agents. Actually, there are two error variables that indicate the state of the i-j link, the first one is shown in the right above equation 5.10 and the second one is the following:

$$e^{(ji)} := \Sigma\Delta z^{(ji)} - \hat{\Sigma}\Delta z^{(ji)}, \tag{5.11}$$

and every time there is a i-j communication, i.e., i sends information and j updates, or j sends information and i updates, one of them ($e^{(ij)}$ or $e^{(ji)}$) becomes zero. Thus, it is logical to introduce a new error variable that considers completely the i-j link as follows:

$$\epsilon^{(ij)} := e^{(ij)} + e^{(ji)}, \quad \forall i < j. \tag{5.12}$$

Let us regard the case when the timer event on agent i triggers and sends a message to agent j, which updates upon the detection of the data reception event. This case is valid regardless the version of the protocol used since a successful "gossip collision" in the uni-cast version can be regarded as a two consecutive update of the same pair of agents in the multi-cast version. From equation 5.3 or 5.6 and replacing the value of the manipulated variable $\Delta z^{(ji)}$ and the corrected states $z^{(j)}_{corr}$ and $z^{(i)}_{corr}$ we have:

$$\begin{aligned}
z^{(j)}_{next} &= \tfrac{1}{2}\left[z^{(j)} + \left(\Sigma\Delta z^{(ij)} - \hat{\Sigma}\Delta z^{(ij)}\right)\right] + \tfrac{1}{2}\left[z^{(i)} + \left(\Sigma\Delta z^{(ji)} - \hat{\Sigma}\Delta z^{(ji)}\right)\right] \\
\Sigma\Delta z^{(ji)}_{next} &= \Sigma\Delta z^{(ji)} + \Delta z^{(ji)} \\
\hat{\Sigma}\Delta z^{(ij)}_{next} &= \Sigma\Delta z^{(ij)},
\end{aligned}$$

and regarding the next state of agent i that did not perform any update:

$$\begin{aligned} z_{next}^{(i)} &= z^{(i)} \\ \Sigma\Delta z_{next}^{(ij)} &= \Sigma\Delta z^{(ij)} \\ \hat{\Sigma}\Delta z_{next}^{(ji)} &= \hat{\Sigma}\Delta z^{(ji)}, \end{aligned}$$

we can subtract some equations an obtain the following expressions:

$$\begin{aligned} z_{next}^{(j)} &= \tfrac{1}{2}\left[z^{(j)} + \left(\Sigma\Delta z^{(ij)} - \hat{\Sigma}\Delta z^{(ij)}\right)\right] + \tfrac{1}{2}\left[z^{(i)} + \left(\Sigma\Delta z^{(ji)} - \hat{\Sigma}\Delta z^{(ji)}\right)\right] \\ z_{next}^{(i)} &= z^{(i)} \end{aligned}$$

$$\begin{aligned} \left(\Sigma\Delta z_{next}^{(ji)} - \hat{\Sigma}\Delta z_{next}^{(ji)}\right) &= \left(\Sigma\Delta z^{(ji)} - \hat{\Sigma}\Delta z^{(ji)}\right) + \Delta z^{(ji)} \\ \left(\Sigma\Delta z_{next}^{(ij)} - \hat{\Sigma}\Delta z_{next}^{(ij)}\right) &= 0, \end{aligned}$$

and replacing the remaining manipulated variable $\Delta z^{(ji)}$ and using the error variables:

$$\begin{aligned} z_{next}^{(j)} &= \tfrac{1}{2}z^{(j)} + \tfrac{1}{2}z^{(i)} + \tfrac{1}{2}e^{(ij)} + \tfrac{1}{2}e^{(ji)} \\ z_{next}^{(i)} &= z^{(i)} \\ e_{next}^{(ji)} &= \tfrac{1}{2}z^{(j)} - \tfrac{1}{2}z^{(i)} + \tfrac{1}{2}e^{(ij)} + \tfrac{1}{2}e^{(ji)} \\ e_{next}^{(ij)} &= 0, \end{aligned}$$

finally, using the total error variable of the i-j link and assuming that $i < j$ (in the opposite case $j < i$ simply we replace $\epsilon_{next}^{(ij)}$ by $\epsilon_{next}^{(ji)}$) we have:

$$z_{next}^{(j)} = \tfrac{1}{2}z^{(j)} + \tfrac{1}{2}z^{(i)} + \tfrac{1}{2}\epsilon^{(ij)} \tag{5.13}$$

$$z_{next}^{(i)} = z^{(i)} \tag{5.14}$$

$$\epsilon_{next}^{(ij)} = \tfrac{1}{2}z^{(j)} - \tfrac{1}{2}z^{(i)} + \tfrac{1}{2}\epsilon^{(ij)}, \tag{5.15}$$

and for completeness and in order to clearly see the asynchronous updates matrices, if the timer of the agent j triggers and the agent i detects an event of data reception:

$$z_{next}^{(j)} = z^{(j)} \tag{5.16}$$

$$z_{next}^{(i)} = \tfrac{1}{2}z^{(j)} + \tfrac{1}{2}z^{(i)} + \tfrac{1}{2}\epsilon^{(ij)} \tag{5.17}$$

$$\epsilon_{next}^{(ij)} = -\tfrac{1}{2}z^{(j)} + \tfrac{1}{2}z^{(i)} + \tfrac{1}{2}\epsilon^{(ij)}. \tag{5.18}$$

Now, let us define the state of the network as follows:

$$x_k = \left(\begin{array}{c} z_k \\ \hline \epsilon_k \end{array}\right),$$

where $z_k \in \mathbb{R}^N$ is the vector that contains all the momentums of the N nodes in the network at time k and $\epsilon \in \mathbb{R}^E$ is the vector which entries are the E errors of every bidirectional link between at instant k. Note that the order of the entries indexed in the vectors z and ϵ does not matter, but once is defined then it must be unalterable in the analysis.

With the above definitions it is possible to define an asynchronous transition matrix of the state of the system depending on the pair of agents involved and on the agent that detects the reception event. Thus, the dynamics of the system between the instants k and $k+1$ can be expressed as:

$$x_{k+1} = M_k x_k \quad , \tag{5.19}$$

where M_k is one of the asynchronous matrices induced by the algorithm when someone detects an event of data reception given by the expressions in 5.13 or 5.16 and considering that all other states in the network remain unalterable due to asynchronicity.

Since the state of the system x_k is formed by the momentum vector z_k and the error of the links ϵ_k, we can separate the matrices by components and have the following updates:

$$\left(\begin{array}{c} z_{k+1} \\ \hline \epsilon_{k+1} \end{array}\right) = \left[\begin{array}{c|c} A_k & B_k \\ \hline C_k & D_k \end{array}\right] \left(\begin{array}{c} z_k \\ \hline \epsilon_k \end{array}\right) \qquad M_k := \left[\begin{array}{c|c} A_k & B_k \\ \hline C_k & D_k \end{array}\right],$$

where $A_k \in \mathbb{R}^{(N\times N)}$, $B_k \in \mathbb{R}^{(N\times E)}$, $C_k \in \mathbb{R}^{(E\times N)}$ and $D_k \in \mathbb{R}^{(E\times E)}$ for all $k \in \mathbb{N}_0$.

In general, if agent j updates its state due to the expiration of timer i, the update matrix will have the form in the expression 5.20.

$$
\begin{array}{c}
 \\
 \\
i^{th} \\
 \\
j^{th} \\
 \\
 \\
\hline
 \\
 \\
q^{th} \\
 \\
 \\
\end{array}
\begin{array}{c}
\begin{array}{ccccccc} & & i^{th} & & j^{th} & & \end{array} \\
\left[\begin{array}{ccccccc|ccccc}
1 & \cdots & 0 & \cdots & 0 & \cdots & 0 & 0 & \cdots & 0 & \cdots & 0 \\
\vdots & \ddots & \vdots & & \vdots & & \vdots & \vdots & & \vdots & & \vdots \\
0 & \cdots & 1 & \cdots & 0 & \cdots & 0 & 0 & \cdots & 0 & \cdots & 0 \\
\vdots & & \vdots & \ddots & \vdots & & \vdots & \vdots & & \vdots & & \vdots \\
0 & \cdots & 1/2 & \cdots & 1/2 & \cdots & 0 & 0 & \cdots & 1/2 & \cdots & 0 \\
\vdots & & \vdots & & \vdots & \ddots & \vdots & \vdots & & \vdots & & \vdots \\
0 & \cdots & 0 & \cdots & 0 & \cdots & 1 & 0 & \cdots & 0 & \cdots & 0 \\
\hline
0 & \cdots & 0 & \cdots & 0 & \cdots & 0 & 1 & \cdots & 0 & \cdots & 0 \\
\vdots & & \vdots & & \vdots & & \vdots & \vdots & \ddots & \vdots & & \vdots \\
0 & \cdots & -1/2 & \cdots & 1/2 & \cdots & 0 & 0 & \cdots & 1/2 & \cdots & 0 \\
\vdots & & \vdots & & \vdots & & \vdots & \vdots & & \vdots & \ddots & \vdots \\
0 & \cdots & 0 & \cdots & 0 & \cdots & 0 & 0 & \cdots & 0 & \cdots & 1
\end{array}\right]
\begin{array}{c} \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \end{array}
\end{array}
\quad , \tag{5.20}
$$

where q is the position of the error variable ϵ^{ij}.

For the sake of completeness, if the agents involved in the collision are the same but this time node i updates due to expiration of timer at j, the transition matrix is the one in the expression 5.22.

Note that these set of matrices does not have the form of 2.2 since these new matrices combine the separated states of the agents to form states of the communication links, however they indeed represent asynchronous updates of the complete state of the network.

As we have discussed before, from equation 5.19 it is easy to obtain the state as a function of the initial condition, which can be expressed as:

$$x_k = M_{k-1} \cdots M_1 M_0 x_0 \quad , \tag{5.21}$$

$$
\begin{array}{c}
\\
\\
i^{th} \\
\\
j^{th} \\
\\
\\
\\
\\
q^{th} \\
\\
\\
\end{array}
\begin{array}{c}
\begin{array}{ccccccccccccc}
 & & i^{th} & & j^{th} & & & & & & q^{th} & & \\
\end{array} \\
\left[
\begin{array}{ccccccc|ccccc}
1 & \cdots & 0 & \cdots & 0 & \cdots & 0 & 0 & \cdots & 0 & \cdots & 0 \\
\vdots & \ddots & \vdots & & \vdots & & \vdots & \vdots & & \vdots & & \vdots \\
0 & \cdots & 1/2 & \cdots & 1/2 & \cdots & 0 & 0 & \cdots & 1/2 & \cdots & 0 \\
\vdots & & \vdots & \ddots & \vdots & & \vdots & \vdots & & \vdots & & \vdots \\
0 & \cdots & 0 & \cdots & 1 & \cdots & 0 & 0 & \cdots & 0 & \cdots & 0 \\
\vdots & & \vdots & & \vdots & \ddots & \vdots & \vdots & & \vdots & & \vdots \\
0 & \cdots & 0 & \cdots & 0 & \cdots & 1 & 0 & \cdots & 0 & \cdots & 0 \\
\hline
0 & \cdots & 0 & \cdots & 0 & \cdots & 0 & 1 & \cdots & 0 & \cdots & 0 \\
\vdots & & \vdots & & \vdots & & \vdots & \vdots & \ddots & \vdots & & \vdots \\
0 & \cdots & 1/2 & \cdots & -1/2 & \cdots & 0 & 0 & \cdots & 1/2 & \cdots & 0 \\
\vdots & & \vdots & & \vdots & & \vdots & \vdots & & \vdots & \ddots & \vdots \\
0 & \cdots & 0 & \cdots & 0 & \cdots & 0 & 0 & \cdots & 0 & \cdots & 1
\end{array}
\right]
\end{array} . \tag{5.22}
$$

and the state as time tend to infinity is:

$$
x_\infty = M_\infty \cdots M_1 M_0 x_0 \quad , \tag{5.23}
$$

where the matrices that appear in the product are taken from the set of asynchronous matrices induced by the algorithm Σ. Note that since we are assuming that the nodes never failed and that the feasible communication topology is unalterable, the dimension of the matrices in the set is finite, also it is clear that the number of matrices in the set Σ is finite too and furthermore each one of these matrices appears indefinitely in the generated product of the expression 5.21 as time tends to infinity, however the order in which the matrices in Σ appears is unknown and depends on the frequency and phase of the internal timers and if the messages are effectively received.

With the above discussion in mind, we will focus on the behavior of the infinite product of matrices:

$$
(M_\infty \cdots M_1 M_0)
$$

where every matrix that appears in the infinite product is in a finite set Σ of asynchronous matrices induced by the algorithm, and also each matrix in Σ appears infinitely often in such product. It is expected that regardless the order in which the matrices appear, the infinite product should converge to an average consensus matrix. According to the expression 5.23 and considering that the initial state is:

$$x_0 = \left(\begin{array}{c} z_0^{(1)} \\ z_0^{(2)} \\ \vdots \\ z_0^{(N)} \\ \hline \epsilon_0 \end{array}\right) = \left(\begin{array}{c} m_1 \\ m_2 \\ \vdots \\ m_N \\ \hline \mathbb{0} \end{array}\right),$$

the infinite product should converge to a matrix which performs the following update:

$$\left(\begin{array}{c} z_\infty^{(1)} \\ z_\infty^{(2)} \\ \vdots \\ z_\infty^{(N)} \\ \hline \epsilon_\infty \end{array}\right) = \left[\begin{array}{cccc|c} 1/N & 1/N & \cdots & 1/N & ? \\ 1/N & 1/N & \cdots & 1/N & ? \\ \vdots & \vdots & \ddots & \vdots & ? \\ 1/N & 1/N & \cdots & 1/N & ? \\ \hline ? & ? & \vdots & ? & ? \end{array}\right] \left(\begin{array}{c} m_1 \\ m_2 \\ \vdots \\ m_N \\ \hline \mathbb{0} \end{array}\right)$$

in which it is easy to see that the momentums of the agents converge to the average of the initial momentums. Since in the design of the algorithm the error variables represent the quantity of momentum that the communication link retains in order to preserve the momentum, and since all the agents have the total momentum of the system stored in their momentum variables, if they converge to the average the error variables should converge to zero as time tends to infinity regardless the initial state, and therefore we should expect that:

$$\left(\begin{array}{c} z_\infty^{(1)} \\ z_\infty^{(2)} \\ \vdots \\ z_\infty^{(N)} \\ \hline \epsilon_\infty \end{array}\right) = \left[\begin{array}{cccc|c} 1/N & 1/N & \cdots & 1/N & ? \\ 1/N & 1/N & \cdots & 1/N & ? \\ \vdots & \vdots & \ddots & \vdots & ? \\ 1/N & 1/N & \cdots & 1/N & ? \\ \hline \mathbb{0} & \mathbb{0} & \cdots & \mathbb{0} & ? \end{array}\right] \left(\begin{array}{c} m_1 \\ m_2 \\ \vdots \\ m_N \\ \hline \mathbb{0} \end{array}\right)$$

We will discover the unknown entries of the infinite product as long as it converges, however we need to exploit some properties of the structure of the asynchronous matrices induced by the algorithm. Before to do that, we will make explicit the asynchronous matrices induced in the case when there are two and three nodes in the network with the objective of understanding properly the model of the system, which regards asynchronicity and packet losses, but not delays.

5.3.1. Example: Asynchronous Matrices Induced by the Algorithm $N = 2$

The set of nodes is $\mathcal{V} = \{1, 2\}$. The feasible communication graph is straightforward and limited by the assumptions of the formulation, so agent 1 can send information to agent 2 and vice versa. The state of the network is:

$$x = \left(\begin{array}{c} z^{(1)} \\ z^{(2)} \\ \hline \epsilon^{(12)} \end{array} \right).$$

The asynchronous update when agent 2 updates due to node 1 is $x_{k+1} = M^{(21)} x_k$:

$$\left(\begin{array}{c} z_{k+1}^{(1)} \\ z_{k+1}^{(2)} \\ \hline \epsilon_{k+1}^{(12)} \end{array} \right) = \left[\begin{array}{cc|c} 1 & 0 & 0 \\ 1/2 & 1/2 & 1/2 \\ \hline -1/2 & 1/2 & 1/2 \end{array} \right] \left(\begin{array}{c} z_k^{(1)} \\ z_k^{(2)} \\ \hline \epsilon_k^{(12)} \end{array} \right),$$

and the asynchronous update when agent 1 updates due to node 2 is $x_{k+1} = M^{(12)} x_k$:

$$\left(\begin{array}{c} z_{k+1}^{(1)} \\ z_{k+1}^{(2)} \\ \hline \epsilon_{k+1}^{(12)} \end{array} \right) = \left[\begin{array}{cc|c} 1/2 & 1/2 & 1/2 \\ 0 & 1 & 0 \\ \hline 1/2 & -1/2 & 1/2 \end{array} \right] \left(\begin{array}{c} z_k^{(1)} \\ z_k^{(2)} \\ \hline \epsilon_k^{(12)} \end{array} \right),$$

and the finite set of matrices $\Sigma = \{M^{(21)}, M^{(12)}\}$ is:

$$\Sigma = \left\{ M^{(21)} = \left[\begin{array}{cc|c} 1 & 0 & 0 \\ 1/2 & 1/2 & 1/2 \\ \hline -1/2 & 1/2 & 1/2 \end{array} \right], M^{(12)} = \left[\begin{array}{cc|c} 1/2 & 1/2 & 1/2 \\ 0 & 1 & 0 \\ \hline 1/2 & -1/2 & 1/2 \end{array} \right] \right\}$$

Trivially, infinite product $(M_\infty \cdots M_1 M_0)$ in which every matrix in Σ is repeated infinitely often converges to:

$$(M_\infty \cdots M_1 M_0) = \left[\begin{array}{cc|c} 1/2 & 1/2 & 1/2 \\ 1/2 & 1/2 & 1/2 \\ \hline 0 & 0 & 0 \end{array}\right]$$

which happens in finite time whenever the first and the second agent updates at least once, and thus the average consensus is reached.

5.3.2. Example: Asynchronous Matrices Induced by the Algorithm $N = 3$

The set of nodes is $\mathcal{V} = \{1, 2, 3\}$. The feasible communication graph can vary depending on which bidirectional strongly connected graph we choose. If the graph is complete the state of the network is as follows:

$$x = \left(\begin{array}{c} z^{(1)} \\ z^{(2)} \\ z^{(3)} \\ \hline \epsilon^{(12)} \\ \epsilon^{(23)} \\ \epsilon^{(13)} \end{array}\right),$$

and the finite set of matrices $\Sigma = \{M^{(21)},\, M^{(12)},\, M^{(32)},\, M^{(23)},\, M^{(31)},\, M^{(13)}\}$ is:

$$\Sigma = \left\{ M^{(21)} = \left[\begin{array}{ccc|ccc} 1 & 0 & 0 & 0 & 0 & 0 \\ 1/2 & 1/2 & 0 & 1/2 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ \hline -1/2 & 1/2 & 0 & 1/2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{array}\right], \quad M^{(12)} = \left[\begin{array}{ccc|ccc} 1/2 & 1/2 & 0 & 1/2 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ \hline 1/2 & -1/2 & 0 & 1/2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{array}\right], \right.$$

$$\left. M^{(32)} = \left[\begin{array}{ccc|ccc} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1/2 & 1/2 & 0 & 1/2 & 0 \\ \hline 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & -1/2 & 1/2 & 0 & 1/2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{array}\right], \quad M^{(23)} = \left[\begin{array}{ccc|ccc} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1/2 & 1/2 & 0 & 1/2 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1/2 & -1/2 & 0 & 1/2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{array}\right], \right.$$

$$\left. M^{(31)} = \left[\begin{array}{ccc|ccc} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 1/2 & 0 & 1/2 & 0 & 0 & 1/2 \\ \hline 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ -1/2 & 0 & 1/2 & 0 & 0 & 1/2 \end{array}\right], \quad M^{(13)} = \left[\begin{array}{ccc|ccc} 1/2 & 0 & 1/2 & 0 & 0 & 1/2 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 1/2 & 0 & -1/2 & 0 & 0 & 1/2 \end{array}\right] \right\}$$

In the case when the feasible communication topology is only strongly connected but not complete and with bidirectional links, one option would be to chose the links $1 \leftrightarrow 2$ and

$2 \leftrightarrow 3$ and the state would be as follows:

$$x = \left(\begin{array}{c} z^{(1)} \\ z^{(2)} \\ z^{(3)} \\ \hline \epsilon^{(12)} \\ \epsilon^{(23)} \end{array} \right),$$

and the finite set of matrices $\Sigma = \{M^{(21)}, M^{(12)}, M^{(32)}, M^{(23)}\}$ would be:

$$\Sigma = \left\{ \begin{array}{l} M^{(21)} = \left[\begin{array}{ccc|cc} 1 & 0 & 0 & 0 & 0 \\ 1/2 & 1/2 & 0 & 1/2 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ \hline -1/2 & 1/2 & 0 & 1/2 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{array}\right], \quad M^{(12)} = \left[\begin{array}{ccc|cc} 1/2 & 1/2 & 0 & 1/2 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ \hline 1/2 & -1/2 & 0 & 1/2 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{array}\right], \\ \\ M^{(32)} = \left[\begin{array}{ccc|cc} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 1/2 & 1/2 & 0 & 1/2 \\ \hline 0 & 0 & 0 & 1 & 0 \\ 0 & -1/2 & 1/2 & 0 & 1/2 \end{array}\right], \quad M^{(23)} = \left[\begin{array}{ccc|cc} 1 & 0 & 0 & 0 & 0 \\ 0 & 1/2 & 1/2 & 0 & 1/2 \\ 0 & 0 & 1 & 0 & 0 \\ \hline 0 & 0 & 0 & 1 & 0 \\ 0 & 1/2 & -1/2 & 0 & 1/2 \end{array}\right] \end{array} \right\}$$

In both cases it is not clear if the infinite product $(M_\infty \cdots M_1 M_0)$ in which every matrix in Σ repeats infinitely always converges, and only we are analyzing the case $N = 3$. The results presented in the previous chapters can prove convergence in the case when the matrices involved in the infinite product are non-negative and each row of them sum one, and, although consensus is still a reachable subspace (which is $\alpha[\mathbf{1}^T \, \mathbf{0}^T]^T$), in this algorithm the matrices induced have entries with negative signs. Therefore, there are no well developed tools to prove converge of the designed algorithm. In the following we will mention some interesting properties that the matrices in the finite subset Σ satisfies.

5.4. Properties of Σ in the infinite product $(M_\infty \cdots M_1 M_0)$

Recall that Σ is the finite set of asynchronous matrices induced by the algorithm. Every matrix $M \in \Sigma$ satisfies various construction properties that makes them invariant under multiplication and with good convergence properties.

5.4.1. Oblique Projections

Every matrix $M \in \Sigma$ is an oblique projection which means that:

$$M^2 = M \quad , \forall M \in \Sigma$$

or equivalently each eigenvalue of M is $\lambda = 1$ or $\lambda = 0$ and simple. This implies that the state does not change if the same communication happens twice or more times consecutively. Note that the rank of M is $(N + E - 1)$ and the nullity is 1, so there is just one eigenvalue $\lambda = 0$ for all $M \in \Sigma$. Also, thanks to the oblique projection property, the 1-eigenspace is the same as the range of M.

Note that with this in mind, the algorithm is a system that every time that updates:

$$x_{k+1} = M_k x_k$$

is applying an oblique projection over a range with less dimension. Through the time an if the system converges, the infinite product in the expression:

$$x_\infty = (M_\infty \cdots M_1 M_0)\, x_0$$

should converge to a projection matrix of rank 1, i.e., with a unique eigenvalue $\lambda = 1$ and the remaining $(N + E - 1)$ eigenvalues equal to zero.

5.4.2. Conservation Property is Invariant

For every matrix $M \in \Sigma$ there is a left-eigenvalue $\lambda = 1$ corresponding to the left-eigenvector $\mathbb{1}^T$, which means that the momentum is preserved during an update:

$$\mathbb{1}^T M = \mathbb{1}^T \quad , \forall M \in \Sigma$$

Furthermore, for every product of matrices $(M_k \cdots M_1 M_0)$ taken from Σ, it also satisfies the above property, which means that the conservation property is invariant over time:

$$\mathbb{1}^T (M_k \cdots M_1 M_0) = \mathbb{1}^T \quad , \forall M_i \in \Sigma, \forall k \in \mathbb{N}$$

5.4.3. Consensus is a Reachable Subspace

For every matrix $M \in \Sigma$ there is a right-eigenvalue $\lambda = 1$ corresponding to the right-eigenvector $\begin{bmatrix} \mathbb{1}^T & \mathbb{0}^T \end{bmatrix}^T$, where $\mathbb{1}$ has N entries and $\mathbb{0}$ has E entries, which means that the consensus is an equilibrium point:

$$M \begin{bmatrix} \mathbb{1} \\ \mathbb{0} \end{bmatrix} = \begin{bmatrix} \mathbb{1} \\ \mathbb{0} \end{bmatrix} \quad , \forall M \in \Sigma$$

Furthermore, every product of matrices $(M_k \cdots M_1 M_0)$ taken from Σ also satisfies the above property, which means that if consensus is reached then the state cannot be moved from there no matter how many updates happen:

$$(M_k \cdots M_1 M_0) \begin{bmatrix} \mathbb{1} \\ \mathbb{0} \end{bmatrix} = \begin{bmatrix} \mathbb{1} \\ \mathbb{0} \end{bmatrix} \quad , \forall M_i \in \Sigma, \forall k \in \mathbb{N}$$

5.4.4. Consensus is the Unique Reachable Subspace

By construction it is possible to see that the intersection of the 1-eigenspaces of the matrices in Σ, which is the same as the intersection of the ranges, is the consensus subspace:

$$\bigcap_{M \in \Sigma} E_1(M) = \alpha \begin{bmatrix} \mathbb{1} \\ \mathbb{0} \end{bmatrix} \quad , \forall \alpha \in \mathbb{R}$$

This implies that if the infinite matrix product $(M_\infty \cdots M_1 M_0)$ in which every matrix in Σ appears infinitely often converges, then the columns are in the reachable subspace. Since the conservation property is invariant, the sums of the columns of $(M_\infty \cdots M_1 M_0)$ are equal

to one. Therefore, if the infinite product converges, it must converge to a matrix of the form:

$$(M_\infty \cdots M_1 M_0) = \frac{1}{N}\begin{bmatrix} \mathbb{1} \\ \hline \mathbb{0} \end{bmatrix}\left[\begin{array}{c|c} \mathbb{1}^T & \mathbb{1}^T \end{array}\right] = \left[\begin{array}{ccc|ccc} 1/N & \cdots & 1/N & 1/N & \cdots & 1/N \\ \vdots & \vdots & \vdots & \vdots & \cdots & \vdots \\ 1/N & \cdots & 1/N & 1/N & \cdots & 1/N \\ \hline 0 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & & \vdots & \vdots & & \vdots \\ 0 & \cdots & 0 & 0 & \cdots & 0 \end{array}\right]$$

5.4.5. The System Converges

The convergence property could not be proven. As we have discussed before, no many mathematical tools have been developed with infinite matrix products without considering non-negative matrices. However, extensive simulations where performed with excellent results. The simulations consist in multiplying randomly the matrices induced by the algorithm and, as long as every matrix in Σ appears infinitely often, the infinite product converges to the average consensus matrix. Even if just a subset of them appear in the infinite product infinitely often, every entry is bounded by one and apparently the product always converges and therefore the finite set Σ would have the LCP property. Furthermore, the same simulations but performing the right product of matrices always converges, and if the matrices in Σ appear indefinitely the average consensus matrix is achieved, therefore apparently Σ is RCP too. Recall that the LCP property is stronger than the infinite product $(M_\infty \cdots M_1 M_0)$ in which every matrix in Σ appear indefinitely converges, however LCP is desirable in a sense of robustness because this bounds the states of the agents.

Here is shown an example for $N = 4$ agents in which the feasible communication topology is complete where clearly the system converges to the desired average consensus matrix. The code can be found in (Beorostica Github, 2018).

After 10 iterations:

0.5000	0.2500	0.1250	0.1250	0.5000	0	0.2500	0	0	0.1250
0.5000	0.2500	0.1250	0.1250	0.5000	0	0.2500	0	0	0.1250
0	0	0.5000	0.5000	0	0	0	0	0	0.5000
0.3750	0.1250	0.0625	0.4375	0.3750	0	0.1250	0.2500	0.3750	0.0625
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	0	0	0	0
0	0.5000	-0.2500	-0.2500	0	0	0.5000	0	0	-0.2500
-0.2500	0	0	0.2500	-0.2500	0	0	0.5000	0.2500	0.0000
-0.1250	-0.1250	0.0625	0.3125	-0.1250	0	-0.1250	0.2500	0.3750	-0.0625
0	0	0.5000	-0.5000	0	0	0	0	0	0.5000

After 40 iterations:

0.2038	0.1936	0.3476	0.2550	0.2038	0.3225	0.1936	0.2831	0.2168	0.3476
0.2703	0.3015	0.1678	0.2604	0.2703	0.2002	0.3015	0.2704	0.2816	0.1678
0.2174	0.2142	0.3263	0.2421	0.2174	0.3067	0.2142	0.2610	0.2199	0.3263
0.2148	0.1995	0.3383	0.2474	0.2148	0.3120	0.1995	0.2615	0.2180	0.3383
0.0313	-0.0313	-0.0156	0.0156	0.0313	0.0000	-0.0313	-0.0625	0.0313	-0.0156
0.0239	0.0216	-0.0267	-0.0187	0.0239	-0.0245	0.0216	-0.0434	0.0037	-0.0267
0.0137	0.0020	0.0166	-0.0322	0.0137	-0.0313	0.0020	-0.0430	-0.0332	0.0166
-0.0111	-0.0059	0.0094	0.0076	-0.0111	0.0105	-0.0059	0.0215	-0.0012	0.0094
0.0334	0.0901	-0.1517	0.0282	0.0334	-0.0908	0.0901	0.0519	0.0612	-0.1517
0.0026	0.0147	-0.0120	-0.0053	0.0026	-0.0053	0.0147	-0.0005	0.0020	-0.0120

After 100 iterations:

0.2498	0.2503	0.2504	0.2495	0.2498	0.2499	0.2503	0.2500	0.2494	0.2504
0.2502	0.2505	0.2495	0.2498	0.2502	0.2495	0.2505	0.2499	0.2500	0.2495
0.2501	0.2505	0.2499	0.2496	0.2501	0.2496	0.2505	0.2499	0.2497	0.2499
0.2506	0.2499	0.2492	0.2502	0.2506	0.2496	0.2499	0.2491	0.2506	0.2492
-0.0003	-0.0002	0.0009	-0.0003	-0.0003	0.0004	-0.0002	0.0000	-0.0006	0.0009
0.0004	0.0001	-0.0009	0.0003	0.0004	-0.0004	0.0001	-0.0003	0.0006	-0.0009
0.0002	0.0000	-0.0003	0.0000	0.0002	-0.0002	0.0000	-0.0002	0.0002	-0.0003
-0.0005	0.0002	0.0007	-0.0004	-0.0005	0.0002	0.0002	0.0005	-0.0008	0.0007
0.0004	-0.0005	-0.0002	0.0004	0.0004	0.0000	-0.0005	-0.0007	0.0006	-0.0002
-0.0009	-0.0008	0.0009	0.0008	-0.0009	0.0015	-0.0008	0.0017	0.0001	0.0009

And after 10000 iterations:

0.2500	0.2500	0.2500	0.2500	0.2500	0.2500	0.2500	0.2500	0.2500	0.2500
0.2500	0.2500	0.2500	0.2500	0.2500	0.2500	0.2500	0.2500	0.2500	0.2500
0.2500	0.2500	0.2500	0.2500	0.2500	0.2500	0.2500	0.2500	0.2500	0.2500
0.2500	0.2500	0.2500	0.2500	0.2500	0.2500	0.2500	0.2500	0.2500	0.2500
-0.0000	0.0000	0.0000	-0.0000	0.0000	-0.0000	0.0000	0.0000	0.0000	-0.0000
0.0000	0.0000	-0.0000	0.0000	0.0000	0.0000	0.0000	-0.0000	0.0000	0.0000
0.0000	-0.0000	-0.0000	0.0000	-0.0000	-0.0000	0.0000	0.0000	0.0000	-0.0000
0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	-0.0000	-0.0000	0.0000	0.0000
-0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	-0.0000	-0.0000	-0.0000	0.0000
0.0000	0.0000	-0.0000	-0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	-0.0000

Chapter 6. EVALUATION OF THE PROPOSED ALGORITHMS AS PROTOCOLS

As we could see, two versions were developed thanks to the creation of new variables that store the momentum that the bidirectional link retains from the beginning. On one hand, in the uni-cast version a node needs to choose another agent from its neighbors every time it detects a timer event and hopefully to receive a message in response in a short time, also two types of messages must be labeled. On the other hand, in the multi-cast version an agent does not need to choose an agent since it sends the information to all its neighbors when the timer expires, and also it is not necessary to add a second type of message.

In the following, an evaluation over real IoT environments will be exposed where the robustness of the protocol will be tested with all the unfavorable phenomena that a real communication channel presents, in particular not only asynchronous updates or packet losses, but also delays. Two infrastructured communication networks will be tested: a local laboratory with wireless capabilities (22 nodes) and in the public network FiT IoT laboratory (270 nodes). Furthermore, comparisons with the recent push-sum based algorithm (Bof et al., 2017) with the ideas of adding storage variables in case of packet losses will be made and we will show some practical aspects in which our algorithm is superior. Note that, in general, these protocols are rarely implemented over real environments but extensively simulated, therefore the hardware implementation is really a valuable contribution in order to test the performance and robustness of these algorithms. For further details about the code, please refer to (Beorostica Github, 2018) where it is possible to find the necessary files to run the protocol over the open IoT-LAB in France (FIT IoT-LAB, 2018).

6.1. Uni-cast version

6.1.1. Protocol for Implementation

In the following, we will state uni-cast version of the protocol as an algorithm suitable for coding on each agent.

- Initial conditions:

$$z^{(i)} \leftarrow m_i\,, \quad \Sigma\Delta z^{(ij)} \leftarrow 0\,, \quad \hat{\Sigma}\Delta z^{(ji)} \leftarrow 0\,,$$

  for all $i \in \mathcal{V}$ and for all $j \in \mathcal{V}^{(i)}$.

- If agent i generates a timer event:
  (1) Select an agent j in such a way all the neighbor are covered after a finite time.
  (2) Transmit a message to agent j with the actual momentum $z^{(i)}$, $\Sigma\Delta z^{(ij)}$, $\hat{\Sigma}\Delta z^{(ji)}$ and labeled as type 1.

- If agent i detects an event of data reception of type 1 from agent j with the variables $z^{(j)}$, $\Sigma\Delta z^{(ji)}$ and $\hat{\Sigma}\Delta z^{(ij)}$, then it defines:

$$\begin{aligned} z^{(i)}_{corr} &= z^{(i)} + \left(\Sigma\Delta z^{(ji)} - \hat{\Sigma}\Delta z^{(ji)}\right) \\ z^{(j)}_{corr} &= z^{(j)} + \left(\Sigma\Delta z^{(ij)} - \hat{\Sigma}\Delta z^{(ij)}\right) \\ \Delta z^{(ij)} &= \tfrac{1}{2}\left(z^{(i)}_{corr} - z^{(j)}_{corr}\right) \end{aligned}$$

  (1) Update own internal state:

$$\begin{aligned} z^{(i)} &\leftarrow z^{(i)}_{corr} - \Delta z^{(ij)} \\ \Sigma\Delta z^{(ij)} &\leftarrow \Sigma\Delta z^{(ij)} + \Delta z^{(ij)} \\ \hat{\Sigma}\Delta z^{(ji)} &\leftarrow \Sigma\Delta z^{(ji)} \end{aligned}$$

  (2) Send a message to agent j with the total momentum exchanged updated $\Sigma\Delta z^{(ij)}$ and labeled as type 2.

- If agent i detects an event of data reception of type 2 from agent j with the variable $\Sigma\Delta z^{(ji)}$:
  (1) Update own internal states:

$$\begin{aligned} z^{(i)} &\leftarrow z^{(i)} + \Sigma\Delta z^{(ji)} - \hat{\Sigma}\Delta z^{(ji)} \\ \hat{\Sigma}\Delta z^{(ji)} &\leftarrow \Sigma\Delta z^{(ji)} \end{aligned}$$

REMARK 6.1. *Note that packet labeling as type 1 or type 2 is straightforward in an IoT environment, where agents usually implement full stack communication. Also note that in an IoT environment unique addressing schemes are present.*

6.1.2. Hardware Implementation

6.1.2.1. IoT Local Laboratory Testbed

In order to evaluate the performance of the uni-cast version of proposed protocol, an IoT environment of 22 agents was implemented on our local laboratory. The IoT environment is realistic in the sense that real hardware, heterogeneous in nature, was used, which leads to asynchronism and multi-rate behavior associated with using digital event-triggered devices. The IoT environment also suffers from real phenomena in the communication channel, such as packet losses and delays.

The testbed network consist of a set of 22 heterogeneous devices and a router that serves as network manager, with the potential of implementing infrastructured Local Area Networks (LANs) or ad-hoc (peer-to-peer) networks. We pre-set the feasible communication links or, more precisely, each agent knows its neighborhood of agents. In particular, for all the following experiments, each agent has at most 4 neighbors. Figure 6.1 shows a conceptual cartoon of the IoT environment deployed in an infrastructured LAN configuration. The heterogeneity of the agents is made explicit in the Figure.

The hardware involved are all considered within the class of embedded IoT development platforms (Singh & Kapoor, 2017), except for one laptop that was included in the network to represent the interaction between human-oriented devices and embedded sensors/actuators. It is important to emphasize that another shared characteristic is that every device has a processor which runs a derivative Debian distribution of Linux Operative System (OS) such as Ubuntu, Debian for BeagleBone and Raspbian, that, by the way, has the special feature of being open source. Working with an OS allows to easily set-up and manage a communication network and also allows to execute practically any source code.

The technical specs of the hardware used are the following.

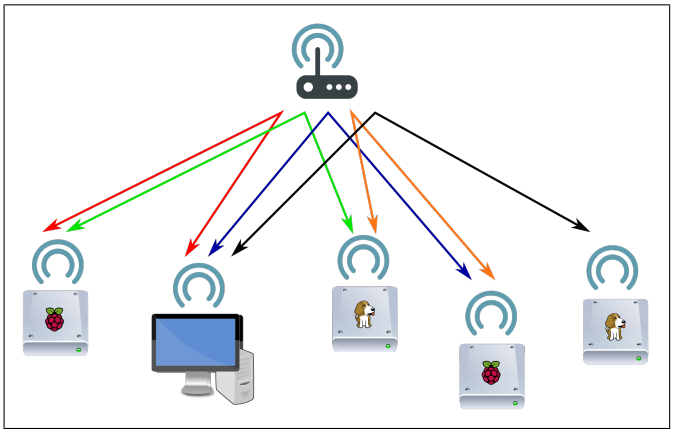

FIGURE 6.1. IoT environment deployed in an infrastructured LAN configuration. Only 5 agents are shown for simplicity, however in the implementation there are 22 agents deployed. Even though the router is a central entity, the processing of the information in the network is totally distributed.

- Router LINKSYS WRT1900AC (Belkin International, Inc., 2017): acts as network manager. Its main features are a 1.3GHz dual-core processor, and that it supports dual band operation at 2.4GHz and 5GHz with transfer rate up to 1.9Gbps.
- Raspberry Pi 3 (Raspberry Pi Foundation, 2017): it is powered by a 1.2-GHz 64-bit quad-core ARM Cortex-A53 CPU, with 1.2GB of RAM and an integrated 802.11n 2.4GHz wireless transceiver. It runs the Raspian operating system. The IoT testbed contains 4 Raspberry Pi 3 boards.
- Laptop HP Mini series 210-3000 (HP Development Company, L.P., 2017): it has a 1.66 GHz Intel Atom N570 processor, 1GB of DDR3 RAM and a 802.11 b/g/n 2.4GHz wireless transceiver. It runs Ubuntu 16.04 LTS as operating system.
- Beagle Bone Green Wireless (BeagleBoard.org Foundation, 2017b): it is based on the Texas Instruments AM335x 1GHz ARM Cortex-A8 processor, it has 512MB of DDR3 RAM and an integrated 802.11 b/g/n 2.4GHz wireless transceiver. It runs the Debian Jessie operating system. The IoT testbed contains 2 Beagle Bone Green Wireless boards.
- Beagle Bone Black (BeagleBoard.org Foundation, 2017a): it is powered by the Texas Instruments AM335x 1GHz ARM Cortex-A8 processor, it has 512MB of DDR3 RAM. The IoT testbed contains 15 Beagle Bone Black boards. Out of the

15 boards, 13 boards run the Debian Jessie operating system and have attached a TP-Link TL-WN722N USB dongle (TP-Lnk Technologies Co., Ltd., 2017), which enables 802.11n connectivity in the 2.4GHz band. The other two boards run the Debian Wheezy operating system and have attached a TP-Link Archer-T2UH USB dongle (TP-Link Technologies Co., Ltd., 2017), which enables 802.11ac connectivity in the 5GHz band.

Since all the agents run a complete operating system, the protocol operates in the agents by making use of a full protocol stack implemented in each of them, to enable transparent communication. Although consensus on a protocol stack for IoT systems has not been reached, we resort to a classical 4-layered stack to represent the communication processes. The layers are, from higher to lower: Application, Transport, Network and Network Interface.

For the Network Interface, the testbed includes agents implementing the 802.11n protocol at 2.4GHz and the 802.11ac protocol at 5GHz. Since the router supports dual band operation, it bridges both subnetworks whenever needed. For the Network layer, all the agents implement the Internet Protocol (IP) version 4. When operating in infrastructure mode, a DHCP server running at the router is in charge of address management, while when operating in ad-hoc mode, pre-set addresses hard coded in the agents are used. At the Transport layer, the User Datagram Protocol (UDP) was chosen instead of others such as the Transmission Control Protocol (TCP) since for control-oriented tasks, speed is more important than quality of service in terms of reliability. Note that since UDP is used, retransmissions for packet not correctly delivered are not present, hence the protocol indeed needs to correct for momentum losses in the following experiments. Finally, the Application layer is not fully determined by the protocol, since we are working in between the application and the transport layers, where each agent opens a socket over in order to transmit and receive UDP datagrams. Figure 6.2 summarizes the protocols used in each layer of the 4-layered protocol stack in our IoT testbed.

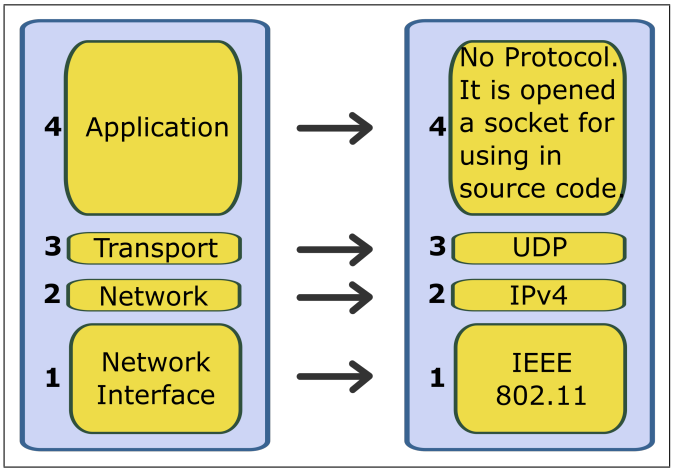

FIGURE 6.2. Protocol stack used to model the communication between agents and protocols chosen for the evaluation in the IoT testbed.

For each agent, the protocol was coded in the user space where timer events and data reception events are handled. Specially suitable for these purposes is Node.js due to its asynchronous even driven nature. Also, the Cloud9 IDE server was made use of in all the agents since it facilitates coding and debugging.

For all the experiments, the initial state of the consensus values or momentums are different for each agent and they are: 12, 55, 25, 70, 100, 65, 35, 75, 45, 78, 15, 85, 57, 95, 13, 0, 5, 10, 60, 88, 90, 22, which give an average of 50. To increase the heterogeneity, a multi-rate feature was added in the frequency of the timers such that 5 agents have a timer with period of $0.5\,[s]$, 6 agents with period $1\,[s]$, 5 agents with period $1.5\,[s]$, and 6 agents with period $2\,[s]$.

At the beginning of the experiment, each device is in idle mode not communicating data with anyone. In order to start the data exchange, an external device sends repeatedly an start signal until all agents are enabled to communicate information. After that, the protocol develops naturally.

The feasible communication topologies used are shown in Figure 6.3 and Figure 6.4. Figure 6.3 shows four different topologies, all are based on a ring structure. The idea behind the chosen topologies is to test the performance of the protocol for the minimum quantity of links, i.e., the ring structure, and then to evaluate its behavior as links are added over the ring

structure. It should be noted that for a ring topology, packet losses are prone to disconnect the network, hence they are more harmful than in other configurations.

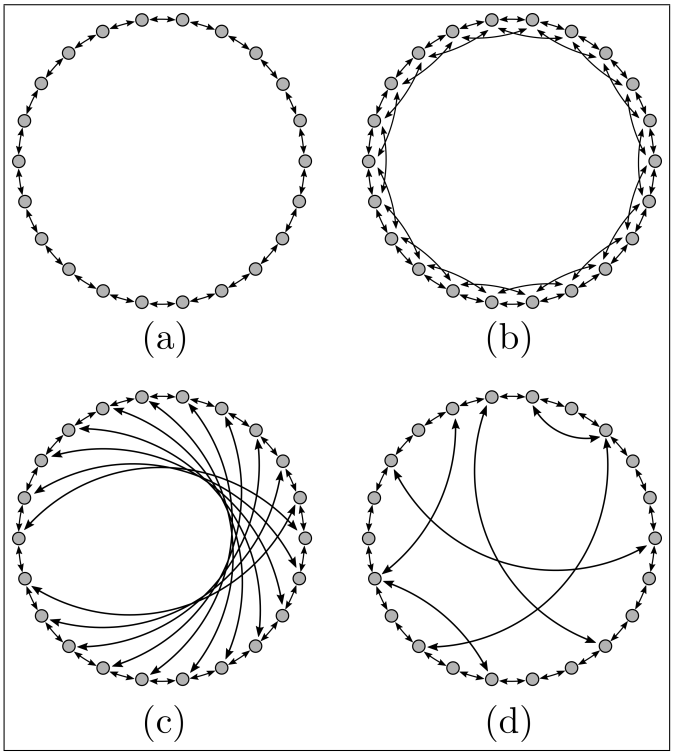

FIGURE 6.3. Topologies used in the first set of experiments, which are based on a ring structure. (a) Pure ring topology. (b) Ring topology with 2 extra links per agent corresponding to the 2 next nearest neighbors. (c) Ring topology with 1 extra link per agent corresponding to the farthest neighbor. (d) Ring topology with 6 extra links, randomly chosen.

Additionally, Figure 6.4 shows two different feasible communication topologies that are time-varying by means of pseudo-switches. These pseudo-switches open and close at different time instants. The idea behind these topologies it to emulate networks with mobility where disconnected clusters are likely to be found from time to time. An also to emulate IoT environments with multiple interfaces, where an agent bridges two subnetworks with different technologies in the physical layer of the protocol stack, e.g., an IEEE802.11 network and an IEEE802.15.4 network.

The dynamics of the communication topology shown in Figure 6.4 (a) is the following. At the beginning, the clusters of 14 and 8 agents does not have any communication link

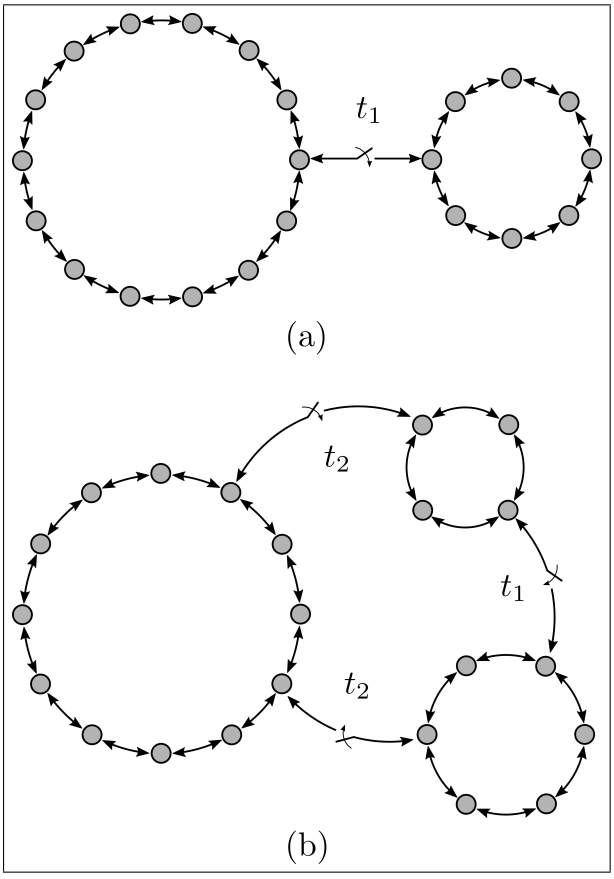

FIGURE 6.4. Communication topologies with time-varying structure. (a) IoT setup where initially there are two isolated ring clusters, which are bridged at time t_1. (b) IoT setup where initially there are three separated ring clusters. At time t_1 two of them are bridged, and at time $t_2 > t_1$ the three clusters are bridged.

between them but the agents of the same cluster can exchange information, after a time t_1 the pseudo-switch is closed by an external device and consequently the final feasible graph is strongly connected.

For the communication topology shown in Figure 6.4 (b), at the beginning there are three isolated clusters where internal agents interact from the beginning, at time t_1 the clusters of 4 and 6 agents are bridged by an external device and form a cluster of 10 agents, then at time t_2 the two remaining clusters are bridged together at two points forming a strongly connected feasible communication graph.

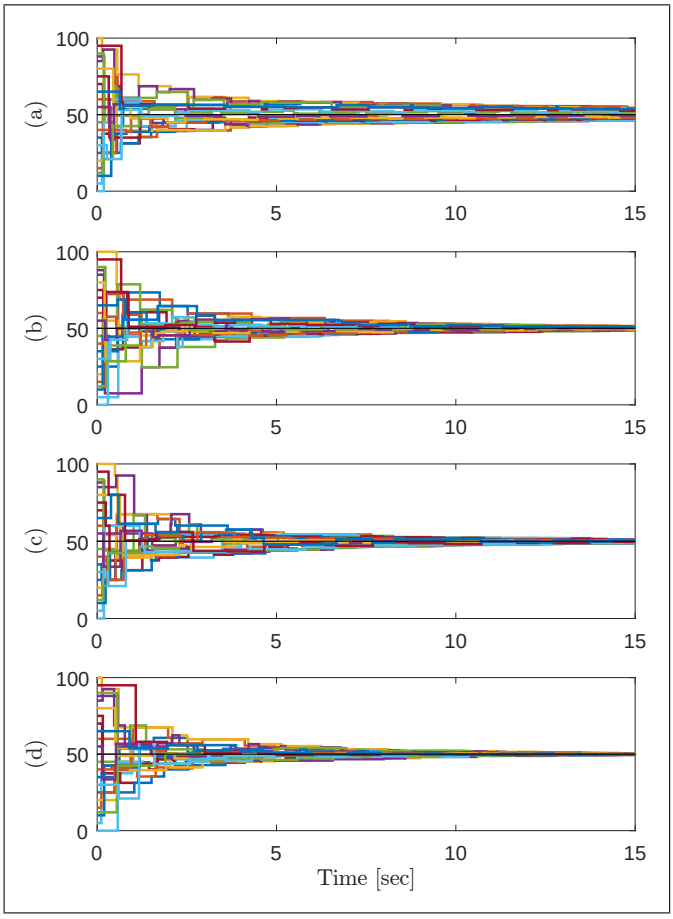

FIGURE 6.5. Results for the communication topologies based on a ring structure presented in Figure 6.3. (a): Results for the ring topology, (b):Results for the ring topology plus two links per agent, (c): Results for the ring topology plus one link per agent, (d): Results for the ring topology plus 6 random links.

Figure 6.5 and Figure 6.6 show the results obtained for the experiments involving the communication topologies in Figure 6.3 and Figure 6.4 respectively. In Figure 6.5 it can be seen that the protocol achieves the average consensus of 50. Also, can be pointed out that the consensus trajectory have an envelope that decreases with an exponential shape, which was expected since this is a convergent linear system, an also the convergence rate is faster for certain topologies than others.

Similar results can be seen in Figure 6.6, where for both topologies the average consensus is achieved with a trajectory bounded by a decaying exponential. However, the most remarkable result in these cases is the robustness of the protocol to deal with time-varying

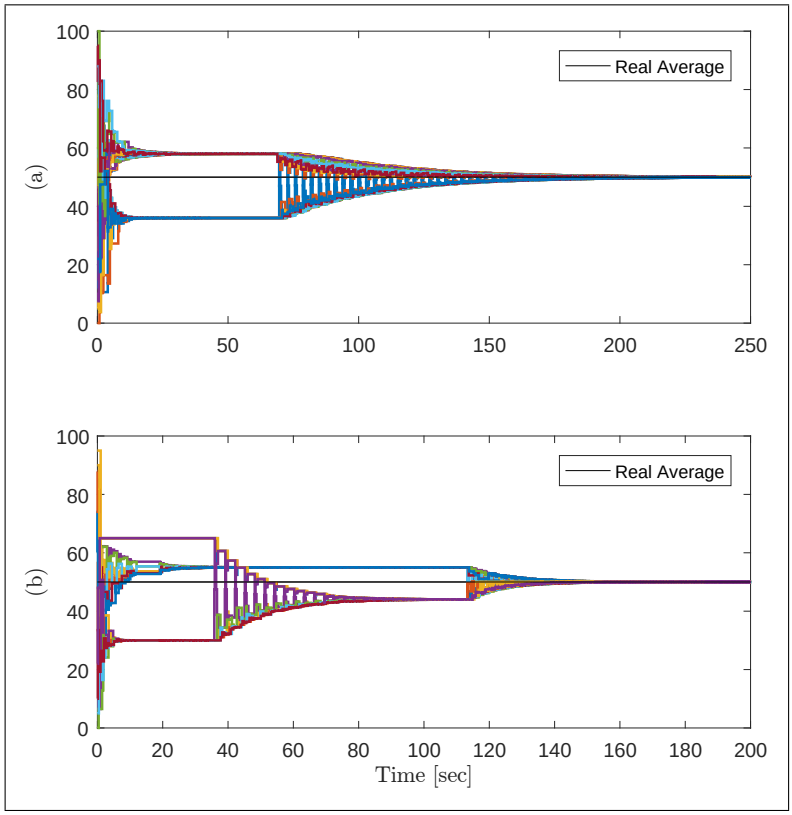

FIGURE 6.6. Results for the time-varying communication topologies shown in Figure 6.4. (a): Results for the time-varying topology with two initial isolated clusters; (b): Results for the time-varying topology with three initial isolated clusters.

interaction topologies. In particular, in Figure 6.6 plot (a) before the pseudo-switch is closed, the agents inside the two clusters achieve the average consensus of its own clusters and then, when interaction among clusters takes place, all the agents achieve the desired global average consensus value. An analogous behavior is shown in Figure 6.6 plot (b), where before time t_1 the agents inside each of the three clusters tend to its local cluster average, between t_1 and t_2 the new formed cluster of 10 agents mix and the agents achieve the combined average value, and finally, after time t_2, the IoT network as a whole is strongly connected and consequently achieves average consensus.

It should be noted that, although the IoT environment used as testbed has several non-ideal phenomena known to deteriorate the performance of traditional average consensus algorithms (Oróstica & Núñez, 2017) , such as packet losses, delays, multi-rate behavior and

asynchronous events, the proposed protocol achieves the desired objective in all cases, even when faced with time-varying communication topologies, as long as the feasible communication topology is strongly connected.

6.1.2.2. FiT IoT LAB Deployment

One of the goals of the proposed protocol is that it should be scalable, so an evaluation over a large network would be ideal. However, we just had a small quantity of nodes in our facilities. Fortunately, there exits the open IoT-LAB in France which provides a very large scale infrastructure suitable for testing heterogeneous communicating devices (FIT IoT-LAB, 2018). In this tested was possible to use more than 270 nodes for all the realizations. It is worth to mention that for each different experiment some nodes could not boot properly at the beginning, so we scheduled a high number of agents but only some of them were in fact running code.

Here we will show a realization in which 285 agents booted properly. All the devices used were A8 nodes which are based on a high-performance ARM Cortex-A8 microprocessor and allows to run high-level OS like Linux. Nodes from Grenoble, Saclay, Strasbourg and Paris, were used. Every agent had a protocol stack similar to the one implemented in our local laboratory, however the physical layer was the standard IEEE 802.3, better known as Ethernet, and in the network layer it was used the public Internet Protocol version 6 (IPv6). In the transport layer the UDP protocol is still used. It is important to emphasize that this network has the typical realistic issues such as packet losses and delays, therefore it really is a good place to test the effectiveness of the proposed protocol.

Several experiments were executed, the initial values, the timer period and the neighbors of each agent were determined by a central computer placed in our installations. The process was automatized and can be summarized as follows: first the manager computer in Chile creates a file with the topology of the network and the initial conditions and parameters of the agents, then the file is sent and read by every node in France, subsequently the manager sends a signal to every agent to run the protocol, meanwhile each node fills a file with the information of their states, after a certain amount of time the central computer sends a stop

signal and each node in France sends back the file with the information of the state, finally the central computer can process the results.

The initial momentum of every agent is a uniform random value between 0 and 100, thus the real average can be calculated by the Chilean central computer in order to be compared with the results of the protocol deployed in France. The period of the timer also was chosen randomly and could have the value of 0.5, 1 or 1.5 seconds. The topology has total number of 285 nodes, the communication graph was random, connected and such that every node had a maximum of 4 neighbors.

The results of one realization is shown in the Figure 6.7. As we can see, the protocol achieves the average consensus as we expected. Several tests have been executed with the same excellent results. The most outstanding feature of this distributed algorithm is its robustness since it has been tested under real and not desirable conditions as asynchronism, packet losses and delays, and in all the cases the average consensus is successfully reached.

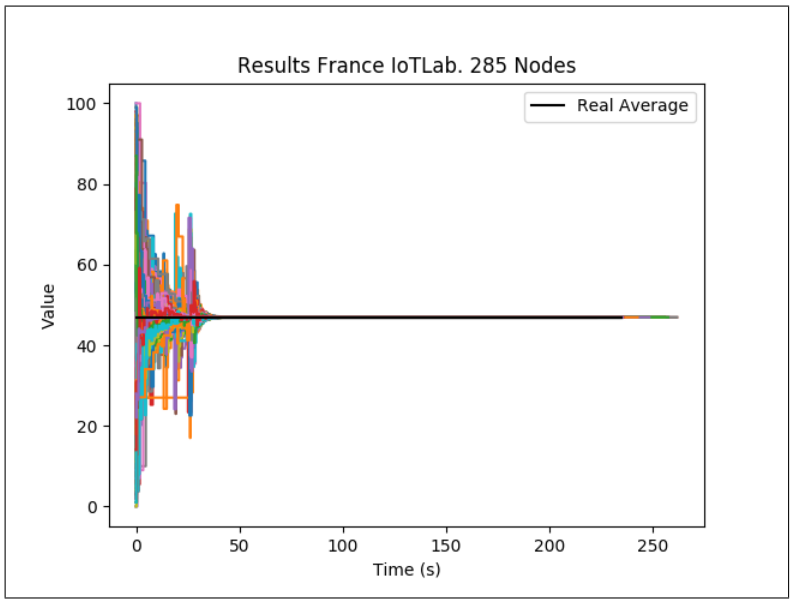

FIGURE 6.7. Results for one realization of the uni-cast algorithm implemented on the IoT-LAB in France with 285 A8 nodes.

6.2. Multi-cast Version

6.2.1. Protocol for Implementation

In the following, we will state the multi-cast version of the protocol as an algorithm suitable for coding on each agent.

- Initial conditions:

$$z^{(i)} \leftarrow m_i\,, \quad \Sigma\Delta z^{(ij)} \leftarrow 0\,, \quad \hat{\Sigma}\Delta z^{(ji)} \leftarrow 0\,,$$

  for all $i \in \mathcal{V}$ and for all $j \in \mathcal{V}^{(i)}$.

- If agent i generates a timer event:

  (1) Transmit a message to each neighbor j with the actual momentum $z^{(i)}$, $\Sigma\Delta z^{(ij)}$, $\hat{\Sigma}\Delta z^{(ji)}$.

- If agent i detects an event of data reception from agent j with the variables $z^{(j)}$, $\Sigma\Delta z^{(ji)}$ and $\hat{\Sigma}\Delta z^{(ij)}$, then it defines:

$$\begin{aligned} z^{(i)}_{corr} &= z^{(i)} + \left(\Sigma\Delta z^{(ji)} - \hat{\Sigma}\Delta z^{(ji)}\right) \\ z^{(j)}_{corr} &= z^{(j)} + \left(\Sigma\Delta z^{(ij)} - \hat{\Sigma}\Delta z^{(ij)}\right) \\ \Delta z^{(ij)} &= \tfrac{1}{2}\left(z^{(i)}_{corr} - z^{(j)}_{corr}\right) \end{aligned}$$

  (1) Update own internal state:

$$\begin{aligned} z^{(i)} &\leftarrow z^{(i)}_{corr} - \Delta z^{(ij)} \\ \Sigma\Delta z^{(ij)} &\leftarrow \Sigma\Delta z^{(ij)} + \Delta z^{(ij)} \\ \hat{\Sigma}\Delta z^{(ji)} &\leftarrow \Sigma\Delta z^{(ji)} \end{aligned}$$

6.2.2. Hardware Implementation over FIT IoTLab Testbed

As in the uni-cast version, the IoT Lab facilities were used in order to test the performance of the multi-cast version of the algorithm. It was used the same communication infrastructure, the same digital devices (A8) and the same protocol stack as in Figure 6.2 in

which the physical layer is ethernet, over the Network layer is the public IPv6 and the Transport layer uses UDP datagrams.

Several experiments were executed in the same way as it was described in the automatized process of the uni-cast version, however it was added an additional change to test the robustness of the protocol to the starting time in which the algorithm begins to run on every node. In order to do so, the central manager computer additionally split into three groups, of approximately the same quantity of agents, from the total number of nodes in the network. Every group had a different average, in particular values of the initial momentums of the first groups was chosen with a uniform distribution in a range between 0 and 33, the second group between 33 and 66 and the third group the initial momentums where taken from the range between 66 and 100. Thus, the random value of the real average was always close to 50. After that, just before the nodes start the protocol, the central manager computer send a start signal for the first group which runs the protocol while the manager awaits for 30 seconds and then send another start signal for the second group, again it awaits for 30 seconds and finally the third group receives the start signal from the chilean manager computer. Finally, after a reasonable quantity of time when the average consensus is achieved, a stop signal is sent, the nodes in France stop the algorithm and return the information to our chilean facilities.

Since the experiments were indeed a real implementation of the algorithm, some issues occurred dealing with measuring time. In particular there was not an absolute notion of time of the whole network (not necessarily all nodes has the same universal time), however every node can count locally certain number of milliseconds after a flag signal. Therefore, in order to display the results of the evolution of the momentums of the agents along the time, before any aforementioned start signal was sent, the chilean manager computer transmit a flag signal which indicates to each agent in France a zero reference point to begin to count the time. Note that this problem does not affect the algorithm at all, it just affects the way in which the states of all agents are displayed over the time on the plots that we will show soon. Thus, in the future displayed results the axis of the time in not absolute, it is the local time of each agent instead. Due to the realistic IoT network used, we indeed will see that the duration of the algorithm in each node is different, which theoretically should have same same length, but

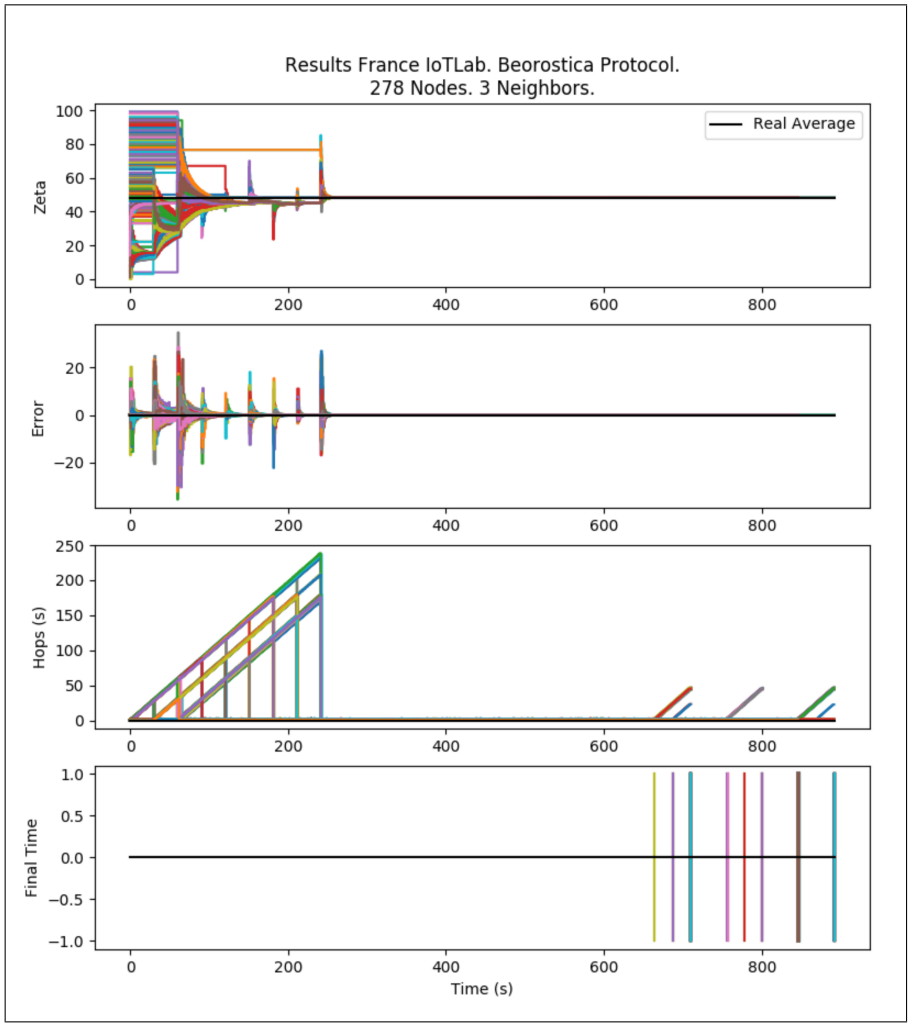

FIGURE 6.8. Results of one realization where every agent had at most three neighbors.

this is not the case due to delays. With this in mind, it is important to note that we do not know the time when an agent starts the protocol and neither when it stop it, however they do not start or stop the algorithm synchronously. Thus, even thought in future figures we will see that all agents starts at time zero, this zero relative to the first received time flag, i.e., the zero is local not absolute.

The results of one realization are shown in the Figure 6.8 in which every agent had at most three neighbors. The first plot shows the evolution of the momentum of every agent $z^{(i)}$. In the second plot are displayed the error variables $\epsilon^{(ij)}$ of every agent for all agents. The

third plot shows all the elapsed times in which an agent do not receive information from a neighbor. Finally, the fourth graph displays the duration time of the algorithm on each node.

Figure 6.8 shows in the first plot that the average consensus is achieved successfully. Note that, since the three groups start at different times, they are trying to converge respectively to local averages at the beginning separated by 30 seconds, thus after the first start signal the first group converges to a number close to 15, after the second start signal the first two groups are running the protocol so they try to converge approximately to a momentum of 33, and finally when the third group receives the start signal all the network try to converge to the average close to 50. However, and due to delays in the messages, the shape of the envelopes of the momentums in the first graph is not a clear exponential as we expected, sometimes the values of the momentums (first plot) and the errors (second plot) change abruptly, which occurs at the times when a node listens to a neighbor that could not listen during a long time according to the third plot. Thus, even though the real implementation has a all the unfavorable phenomena, in this case and in all others experiments conducted the algorithm always achieves average consensus.

6.3. Comparison of the Multi-cast Version with a recent Push-Sum Based Protocol

We wanted to compare the multi-cast version of our algorithm with a recent push-sum based protocol which has the same ideas of adding additional variables to cope with packet losses. Note that the push-sum based protocol only was extensively simulated in (Bof et al., 2017), but here we test this algorithm over a large real network of digital devices and we will see some practical issues of this approach. The push-sum based protocol for implementation is the following:

- Initial conditions:

$$\begin{array}{rclcrcl} y^{(i)} & \leftarrow & m_i & & s^{(i)} & \leftarrow & 1 \\ \Sigma\Delta y^{(ij)} & \leftarrow & 0 & & \Sigma\Delta s^{(ij)} & \leftarrow & 0 \\ \hat{\Sigma}\Delta y^{(ji)} & \leftarrow & 0 & & \hat{\Sigma}\Delta s^{(ji)} & \leftarrow & 0 \end{array} \qquad z^{(i)} = \frac{y^{(i)}}{s^{(i)}}$$

for all $i \in \mathcal{V}$ and for all $j \in \mathcal{V}^{(i)}$.

- If agent i generates a timer event:

  (1) Create auxiliary variables:

$$\begin{aligned} \Delta y &= \tfrac{1}{1+|\mathcal{V}^{(i)}|} y^{(i)} \\ \Delta s &= \tfrac{1}{1+|\mathcal{V}^{(i)}|} s^{(i)} \end{aligned}$$

  (2) Update own state:

$$\begin{aligned} y^{(i)} &\leftarrow \Delta y \\ \Sigma\Delta y^{(ij)} &\leftarrow \Sigma\Delta y^{(ij)} + \Delta y \\ & \qquad\qquad z^{(i)} = \tfrac{y^{(i)}}{s^{(i)}} \\ s^{(i)} &\leftarrow \Delta s \\ \Sigma\Delta s^{(ij)} &\leftarrow \Sigma\Delta s^{(ij)} + \Delta s \end{aligned}$$

  (2) Transmit a message to each neighbor j with the variables $y^{(i)}$, $\Sigma\Delta y^{(ij)}$ and $s^{(i)}$, $\Sigma\Delta s^{(ij)}$.

- If agent i detects an event of data reception from agent j with the variables $y^{(j)}$, $\Sigma\Delta y^{(ji)}$ and $s^{(j)}$, $\Sigma\Delta s^{(ji)}$, then:

(1) Update own internal state:

$$\begin{aligned} y^{(i)} &\leftarrow y^{(i)} + (\Sigma\Delta y^{(ji)} - \hat{\Sigma}\Delta y^{(ji)}) \\ \hat{\Sigma}\Delta y^{(ji)} &\leftarrow \Sigma\Delta y^{(ji)} \\ & \qquad\qquad z^{(i)} = \tfrac{y^{(i)}}{s^{(i)}} \\ s^{(i)} &\leftarrow s^{(i)} + (\Sigma\Delta s^{(ji)} - \hat{\Sigma}\Delta s^{(ji)}) \\ \hat{\Sigma}\Delta s^{(ji)} &\leftarrow \Sigma\Delta s^{(ji)} \end{aligned}$$

Again, the public FiT IoT facilities were used. The exactly automatized process was made as in the previous section. In order to have a fair comparison, both algorithms were executed in parallel. This was possible simply by adding states in the UDP datagram, thus every agent execute the commands of both protocols every time a timer event or an reception event happens and the updates were stored in different variables and when a UDP message needed to be sent, the message contained the states of both protocol in different sections of

the UDP datagram. This implies that every time is executed a realization of both algorithms they can be compared under exactly the same environment, both have the same feasible communication topology, both protocols loss the same packets, they have exactly the same delays and so on.

Figure 6.9 shows the results for a realization in which the feasible communication topology every agent had at most three neighbors. The unique difference between the displayed results of the multi-cast version and the push-sum based algorithm is that in the second graph of the later protocol is plotted the s variables instead of the errors. In general, from the plots of the evolution of the momentums we can see that both protocols achieve the average consensus, however some nodes in the push-sum based present a weird behavior at almost the end of the experiment.

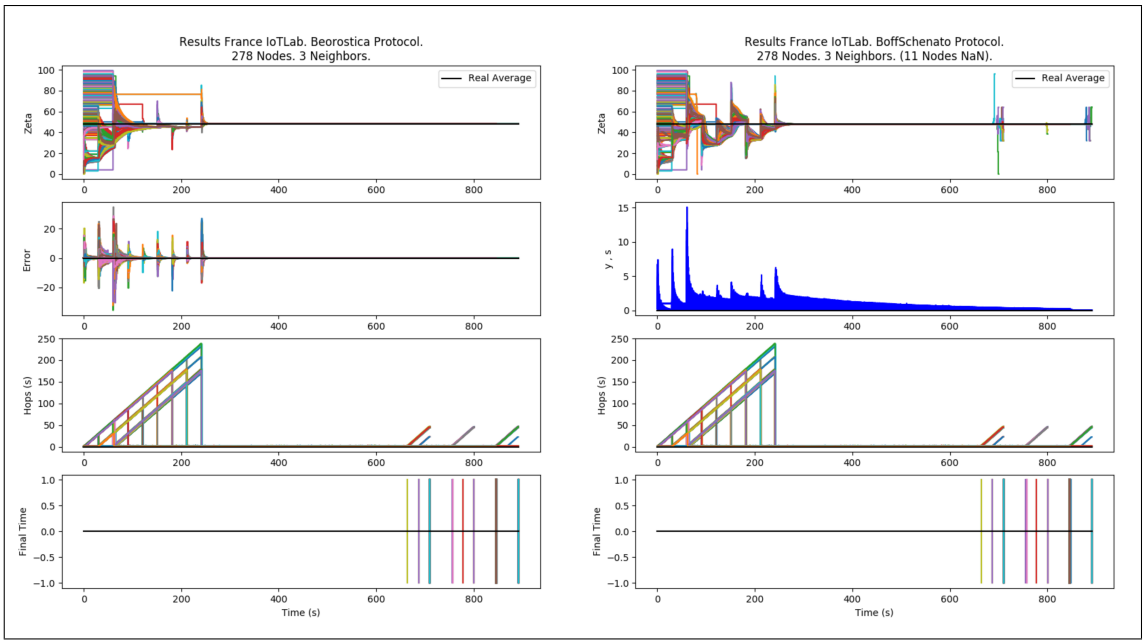

FIGURE 6.9. Comparison between the proposed protocol gossip based and the push-sum based. Every agent had 3 neighbors.

In Figure 6.10 are displayed the results for a realization where every node had at most 15 neighbors. Again the average consensus is achieved successfully in both protocols and the weird behavior still persists in the push-sum based algorithm at almost the end. However here, when every agent has 15 neighbors, there is a clear faster convergence speed than the

in the case when every node had 3 neighbors at least in the intervals where the nodes are at last listened (see third row). Between those intervals of time the convergence has qualitatively an exponential shape which occur repeatedly after all communications are performed regularly and finally after the average consensus is reached there are no abrupt changes of the momentums.

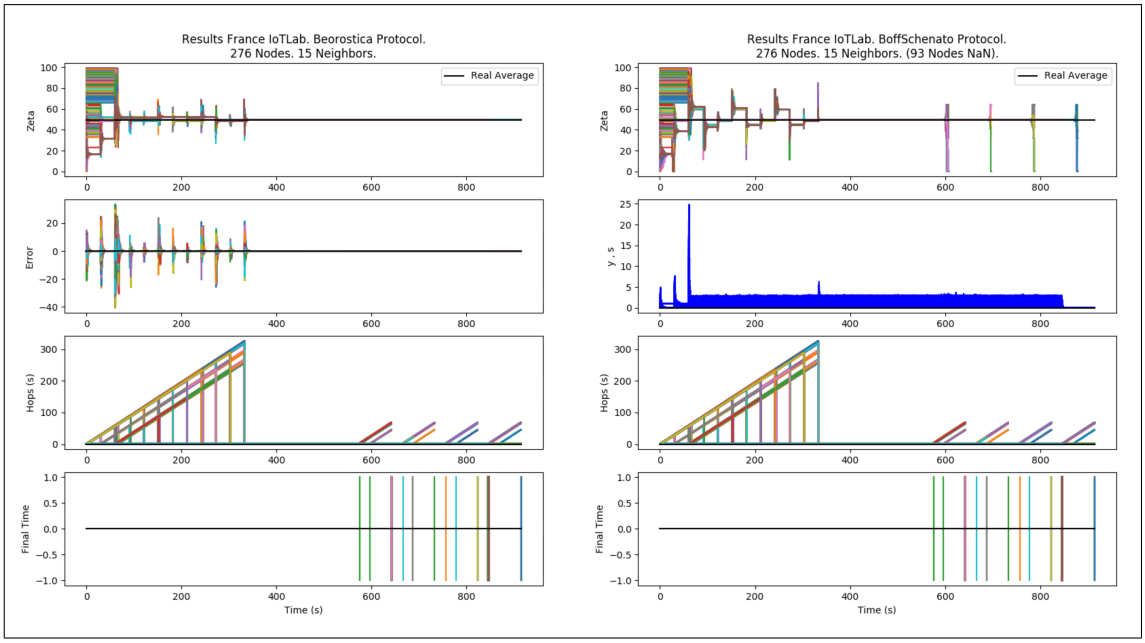

FIGURE 6.10. Comparison between the proposed protocol gossip based and the push-sum based. Every agent had 15 neighbors.

By performing more experiments the weird behavior of the push-sum based protocol always appears. The explanation of this is the following: after the stop signal is sent to every agent, the nodes do not stop immediately the protocol (as it is shown in the fourth row plots), so probably some agents, still operating, update their states due to timer events but they did not receive information from its neighbors which some of them or every one of them were turned off, this implies that the variables y and s tend to zero since both states just decrease their values, indeed by reviewing the states of every agent at the end, the weird nodes have the s value equal to zero which makes the momentum $z = \frac{y}{s}$ undefined. This behavior happens because of the stop signal, however it eventually can occur if a node for any reason can not receive information from its neighbor during a certain finite period of time. Note

that this problem does not happen in the proposed multi-cast gossip based protocol which do not perform updates after a timer event and the changes of the state occur when receives information from its neighbors. Therefore, if a node is isolated our protocol is more robust than the push sum based because the former will not present an undefinition of the state unlike the latter one which decreases the variables y and s. Note that this is a issue related to the quantization of the values on digital devices.

Another practical issue was detected for the push-sum based protocol which was not witnessed in our gossip based algorithm. The situation occurred some times when a node never turned on, i.e., there was a defective node since the very beginning. This anomaly was rather weird but appears more frequently when the public FiT IoT laboratory was under heavy load due to other users using the facilities. Figure 6.11 shows the results for one realization of the experiment in a topology in which every agent had at most 12 neighbors.

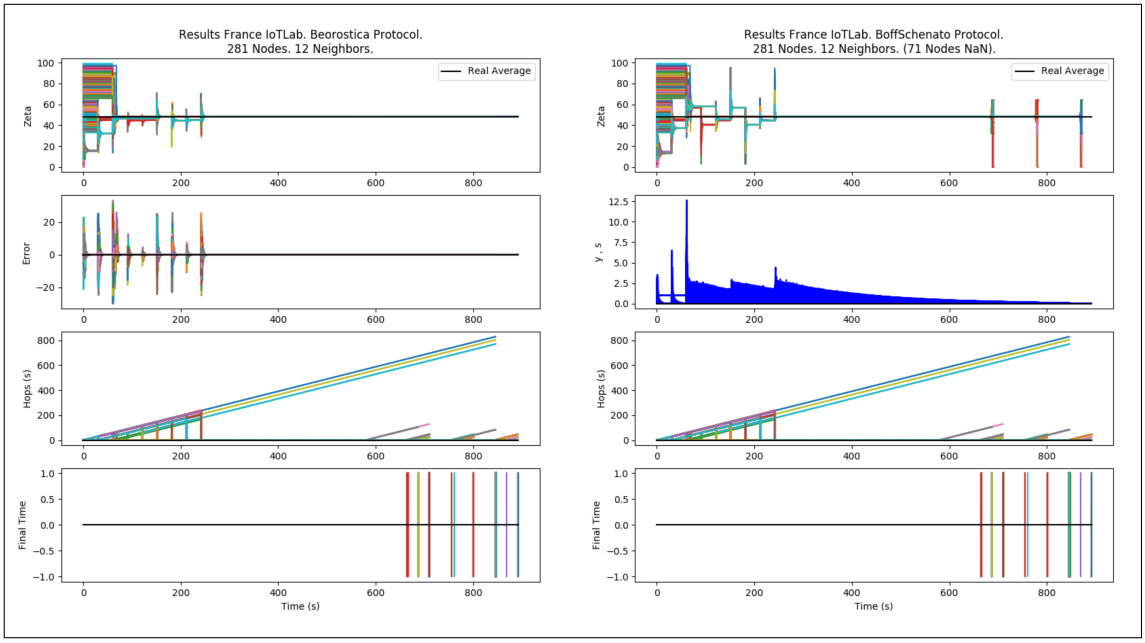

FIGURE 6.11. Comparison between the proposed protocol gossip based and the push-sum based. Every agent had 12 neighbors.

Note that Figure 6.11 shows the same qualitative behavior as the previous results and apparently both protocol achieves the average consensus. However the third row of plots indicates that a some agents never detected the information from a neighbor. By reviewing

the results carefully, the information was never received from one singular node which never runs the algorithm. Furthermore, the values of the consensus of the gossip based and push-sum based where different. However, our gossip based protocol achieved successfully the real average consensus without considering the initial momentum of the defective node, but the push-sum based, even with the same logic of adding storage variable in case of packet losses, did not achieve the average. The explanation is of this is that the push-sum based protocol relies on the topology of the network to preserve the total momentum, in particular when a timer expires it divides theirs variables y and s in equal parts $\frac{1}{1+|\mathcal{V}^{(i)}|}$ among their neighbors, and then it is expected that eventually every neighbor add the same part to its state, however if a neighbor is never present the partition of the state will never be an equal repartition among the neighbors and thus the average is not preserved. This indicates that the push sum based requires information about the topology of the network in order to work properly. Note that our gossip-based algorithm does not have this issue since the momentum conservation property is valid every time there is a change of the state, even if there are packet losses or delays, and do not depend on the number of neighbors of every agent.

Thus, there are some practical issues that makes our gossip-bassed protocol more robust than the push-sum based. In particular, if a node is isolated for any reason since the beginning of the process it achieves successfully the average consensus as long as the feasible communication topology is still strongly connected and also if a node is isolated during a certain period of time there is no quantization issues, unlike the push-sum based protocol which in the former case cannot reach the desired average and in the latter situation the state becomes undefined.

However, it appears the following problem for our gossip based algorithm: what happens if a node at the beginning works properly but suddenly it disappears?. This point was barely treated before because it was assumed that every node is non-defective, but this issue can be fixed by adding some extra steps in the algorithm, the trick can be done thanks that every node can easily determine the momentum without regarding the exchanges performed with

any of their nodes (in our case a defective node) according to the equation:

$$z^{(i)} = m_i - \sum_{j \in \mathcal{V}^{(i)}} \left(\Sigma \Delta z^{(ij)} \right) + \sum_{j \in \mathcal{V}^{(i)}} \left(\hat{\Sigma} \Delta z^{(ji)} \right)$$

in which only internal states of agent i are involved. Thus, if a node detects that a neighbor never sends information, then it can calculate its momentum without exchanging any momentum with the defective node by setting to zero the variables of momentum gained of lost due to the defective node, and even with this change the preservation property is still valid. With this modification in the protocol, the algorithm is robust to a changing feasible communication topology as long as there is still an underlying strongly connected graph. However the convergence analysis with this additional change is rather difficult to treat analytically even if we do not regard delays.

With this last observation pointed out, in order to implement the algorithm as a protocol for either uni-cast version or multi-cast version, the extra steps needed to add on the protocol on each agent are the following:

- If agent i do not receive information from the j agent after a certain T finite time, then updates the own internal state as:

$$\begin{aligned} z^{(i)} &\leftarrow z^{(i)} + \Sigma \Delta z^{(ij)} - \hat{\Sigma} \Delta z^{(ji)} \\ \Sigma \Delta z^{(ij)} &\leftarrow 0 \\ \hat{\Sigma} \Delta z^{(ji)} &\leftarrow 0 \end{aligned}$$

Chapter 7. CONCLUSIONS

In this thesis the distributed average consensus problem was treated for a proper implementation over a real IoT environment. The main two contributions of this manuscript are the design of a new algorithm to solve the distributed average consensus problem and the evaluation of this protocol over a large real IoT environment. Extensive realizations over the testbed indicate that the protocol always achieves the average consensus.

In order to create the algorithm for a real implementation, we always had in mind the non-idealities of the system elements (digital devices and the communication channel) which are asynchronous updates, quantization, packet losses and delays. However, with the objective of making a comprehensive analysis the theoretical framework only considered the asynchronous and the packet losses issues and also assumes that the feasible communication topology was invariant over time.

We saw that, without delays or quantization problems, the change of the state of the network can be analyzed by using a linear time varying model, whose convergence can be analyzed equivalently by an infinite product of matrices from a finite set Σ.

The analytical results presented show that a product of a finite set of row-stochastic matrices with positive diagonal entries such that in the product always are being generated sequences of rooted underlying graphs, converges to a matrix of the form $\mathbb{1}\nu^T$ which is useful for the gossip based algorithm. Analogously, if the finite set of matrices are column-stochastic with positive diagonal entries, then the infinite product of them such that always are being generated sequences of sinked underlying graphs converges to a matrix of the form $\nu\mathbb{1}^T$, which is useful for the push-sum based algorithm. Finally, if the matrices involved are double-stochastic with positive diagonal entries and always are being generating sequences of rooted or sinked or complete underlying graph then the infinite product converges to a matrix of the form $\frac{1}{N}\mathbb{1}\mathbb{1}^T$.

In the context of asynchronous updates with a unique real state for each agent, only the convergence to a matrix of the form $\mathbb{1}\nu^T$ is possible, which means that the consensus problem can be solved. However it is not possible to generate a column-stochastic matrix with positive

diagonal entries that generates sinked underlying graphs (the only matrix that satisfies this is the identity) unless further assumptions are made and therefore convergence to a matrix of the form $\nu \mathbb{1}^T$ would not be reachable, which means that is not possible to conserve the sum of the initial conditions over time, and thus the average consensus could not be solved using this approach.

In order to only reach consensus to an undefined value over a network of digital devices, only considering asynchronous updates and packet losses with one real state per node, it is sufficient that the asynchronous row-stochastic matrices with positive diagonal entries induced by the algorithm Σ are repeated infinitely often over a feasible communication topology strongly connected, i.e., every agent is always updating its state due to its neighbors, which is incidentally a mild assumption. This makes that the infinite product of matrices is always generating sequences of complete graphs, which implies rooted graphs, and therefore the consensus problem is solved.

With the aforementioned analysis of convergence of matrix products is possible to analyze some classical problems like the Vicsek's Problem, the gossip algorithm, the broadcast-gossip and the push-sum. However, the analysis and extensive simulation indicates that just the consensus can be reached but not the average which is lost due to packet losses. Some recent findings have treated the problem of packet losses by adding some extra variables to the state of each agent that stores all the exchanges performed with its neighbors in order to compensate the lost information when necessary.

Thus, with all the learned on the non-idealities of the network, the converge properties and the new ideas from recent state-of-the-art algorithms, a new protocol was designed based on gossips. This new algorithm tries to induce collisions between two agents such that after the collision the agents are close and the momentum is invariant, however due to the asynchronous communication process and the packet losses, the momentum must be preserved on every agent individually when makes an update of its momentum which happens only if the node receives an induced collision. With this logic every agent can calculate its momentum every time it receives information from other neighbor but also can compute its momentum

regarding just the initial momentum and the actual variables that stored the total exchange of momentum with its neighbors. This preservation property is valid even if there are delays since the agent that receives information conserve a quantity invariant whatever is the information received, however analyze convergence is a not clear by using a qualitative analysis.

In order to try to prove convergence, the delay phenomena was neglected, as in the most theoretical findings, but the asynchronous and packet losses phenomenons are still regarded and therefore the analysis can be reduced to the convergence of an infinite matrix product of asynchronous matrices induced by the algorithm Σ in which every matrix is appears indefinitely in the infinite product. The set of matrices Σ is finite and the feasible communication topology is assumed to be unchanged over the time with the objective that the dimensions of the matrices involved do not change.

Some interesting construction properties have the asynchronous matrices induced by the algorithm, one of them is that they are oblique projections which implies that a node do not change its momentum due to a neighbor if it already had the same information from that neighbor, this makes the algorithm in a sense robust to multirate and packet losses. Other property is that the vector $[\mathbb{1}^T \mathbb{0}^T]^T$ is a right-eigenvector which means that the consensus is a reachable subspace with zero momentums stored in the links. Also the matrices in Σ satisfies that $[\mathbb{1}^T \mathbb{1}^T]$ is a left-eigenvector which implies that the sum of the initial condition is preserved all over the time. The final showed property is that since in the infinite product the matrices of the protocol are repeated infinitely often the consensus is the unique reachable subspace as long as the infinite product converges, this property is clear by intersecting the images of the matrices in Σ. However the convergence property could not be proved, even when there is theory on converge of infinite matrix products, the theoretical tools are well developed mainly when the matrices involved are non-negative, however the asynchronous matrices induced by our the protocol have some negative entries. Notwithstanding the aforementioned, simulations of random products of the matrices in Σ shows that the average consensus is always reached, so at least is expected that the infinite product converges to a matrix of the form $\frac{1}{N}[\mathbb{1}^T \mathbb{0}^T]^T[\mathbb{1}^T \mathbb{1}^T]$.

Due to the unravel convergence property, two versions of the algorithm could be developed: a uni-cast version in which pair of agents tries to make "gossip collisions" in a best effort way and a multi-cast version in which a node induces changes of momentums to all its neighbors and eventually each of them induces the momentum back to the the first node when the respective timers expires. It is important to highlight that this gossip based algorithm does not have the problem of dead locks in order to perform an successful "gossip collision" since that the protocol relies on that eventually there is an bidirectional exchange of information. Since the simulation results of the infinite product of the asynchronous matrices induced by the protocols, which are the same in both algorithm, always converged to the average consensus matrix no matter what the order was as long as all the matrices were repeated infinitely often, the protocols were implemented over a local low-scale testbed and over a public large-scale network with more than 270 nodes by using the facilities of the public FiT IoT Laboratory.

The testing over real hardware has all the unfavorable phenomena present in a real IoT environment, in particular our model of infinite products of matrices in Σ only is valid when there are no delays, however in the testbed the algorithms face this unmodeled phenomenon. Implementation results show that average consensus is always reached as long as the nodes work properly during the process. In the case a node is defective, the feasible communication topology changes and the exchanges of momentum due to defective agents would not be regarded. We saw that an additional modification to the algorithms can be made in order to recover convergence to the correct average, which is possible since every agent can compute its momentum considering only the initial momentum and the variables that stored the total exchange of momentum with neighbors. Notwithstanding the aforementioned, in all our experiments was detected that once a node runs the protocol, it never stopped suddenly so the last modification never was performed and average consensus always was reached in all the realizations.

Finally, the new designed multi-cast version of the gossip based protocol was compared against a state-of-the-art push-sum based protocol. For every realization, both algorithms were executed in parallel taking advantage from the fact that the states of both protocols can

be sent in the same UDP datagram and, therefore, the comparison was fair in the sense that both algorithms have exactly the same topology, the same communication over time, the same packet losses and delays. We could see that our gossip-based algorithm is more robust in some practical scenarios, in particular the push-sum based algorithm has a problem when a node is isolated from its neighbors for a long time in which due to quantization some states become zero and the momentum is undefined, and also has the problem that the conservation property depends on the number of neighbors to divide the information, thus if a node fails, the average is not preserved. Thus, our gossip based protocol shows certain degree of superiority.

In summary, this work faced the distributed average consensus problem by means of the design of a new protocol. Extensive real hardware experiments, with all the unfavorable phenomena as asynchronicity, packet losses and delays, show that our protocol always achieves average consensus. Numerical simulations of infinite matrix products, which model the system when delays are neglected, also shows that average consensus is achieved. Even though a lot of effort was put on the properties of the matrices to prove a convergence property, it was not possible to find a way to prove the converge of the infinite matrix product. The most that we could show was that if the product converges, then it converges to the average consensus matrix. One of the main constraints was that most of the theory on convergence of non-homogeneus matrix products is developed for non-negative matrices, which is not satisfied by our asynchronous matrices induced by our algorithm.

Therefore, one of the first aspects to focus on for future research would be the analysis of convergence of the algorithm, in particular simulations indicate that not only the infinite product of matrices from the finite set Σ in which all the matrices are repeated infinitely often in the product converges to the average consensus matrix of the form $\frac{1}{N}[\mathbb{1}^T\ \mathbb{0}^T]^T[\mathbb{1}^T\ \mathbb{1}^T]$, but also they show that the set Σ has a stronger property, LCP, and moreover, Σ is RCP too. The work in (Vladimirov, 2016) and the references there in could be a good starting point for developing on this aspect.

Another point for future research would be the development of a new model that regards the unfavorable phenomenon of delays, however beforehand this is a really difficult task.

Dynamical hybrid systems (Goebel et al., 2009) could be a suitable framework in order to include delays. Note that the practical implementation over a real IoT environment suggests that even with delays the convergence property still persists, therefore, it would be interesting to verify this behavior with a theoretical model.

Further experiments are needed to verify the behavior of the proposed protocol over a dynamically changing feasible communication topology or with defective nodes. The analysis indicates that the conservation property is invariant with the modification of the algorithm by not considering the total exchanges of momentums with the defective agents, however there were not tests performed during the development of this thesis to verify this behavior. These experiments would require that a central manager dynamically changes the communications links of the agents involved during the process in which the distributed algorithm tries to converge, and the development of these experiments would require a long design time in order to make the experiment scalable with the number of agents.

Even though the proposed solution to solve the average consensus problem is distributed, it still requires a certain level of central management in order to generate the feasible communication topology. More research about the a distributed strategy to generate strongly connected graphs over a network would be a good topic in order to complement the proposed protocol and thus the average consensus problem would be solved totally in a distributed way.

REFERENCES

Aysal, T. C., Oreshkin, B. N., & Coates, M. J. (2009, April). Accelerated distributed average consensus via localized node state prediction. *IEEE Transactions on Signal Processing*, *57*(4), 1563-1576.

Aysal, T. C., Yildiz, M. E., Sarwate, A. D., & Scaglione, A. (2009, July). Broadcast gossip algorithms for consensus. *IEEE Transactions on Signal Processing*, *57*(7), 2748-2761.

Aysal, T. C., Yildiz, M. E., & Scaglione, A. (2008, May). Broadcast gossip algorithms. In *2008 ieee information theory workshop* (p. 343-347).

BeagleBoard.org Foundation. (2017a, July). *Beaglebone black.* Retrieved from `https://beagleboard.org/black`

BeagleBoard.org Foundation. (2017b, July). *Seeedstudio beaglebone green wireless.* Retrieved from `https://beagleboard.org/green-wireless`

Belkin International, Inc. (2017, July). *Linksys wrt1900ac ac1900 dual-band smart wi-fi wireless router.* Retrieved from `http://www.linksys.com/us/p/P-WRT1900AC/`

Beorostica Github. (2018, September). *Beorostica thesis: Distributed average consensus.* Retrieved from `https://github.com/beorostica/Beorostica_Thesis_Distributed_Average_Consensus`

Berger, M. A., & Wang, Y. (1992). Bounded semigroups of matrices. *Linear Algebra and its Applications*, *166*, 21 - 27.

Bessis, N., & Dobre, C. (2014). *Big data and internet of things: a roadmap for smart environments*. Springer International Publishing.

Biggs, N. (1993). *Algebraic graph theory*. Cambridge University Press.

Bliman, P.-A., & Ferrari-Trecate, G. (2008). Average consensus problems in networks of agents with delayed communications. *Automatica*, *44*(8), 1985 - 1995.

Blondel, V. D., Hendrickx, J. M., Olshevsky, A., & Tsitsiklis, J. N. (2005, Dec). Convergence in multiagent coordination, consensus, and flocking. In *Proceedings of the 44th ieee conference on decision and control* (p. 2996-3000).

Blondel, V. D., & Tsitsiklis, J. N. (2000). The boundedness of all products of a pair of matrices is undecidable. *Systems & Control Letters*, *41*(2), 135 - 140.

Bnzit, F., Blondel, V., Thiran, P., Tsitsiklis, J., & Vetterli, M. (2010, June). Weighted gossip: Distributed averaging using non-doubly stochastic matrices. In *2010 ieee international symposium on information theory* (p. 1753-1757).

Bof, N., Carli, R., & Schenato, L. (2017). Average consensus with asynchronous updates and unreliable communication. *IFAC-PapersOnLine*, *50*(1), 601 - 606. (20th IFAC World Congress)

Borgia, E. (2014). The internet of things vision: Key features, applications and open issues. *Computer Communications*, *54*, 1-31.

Boyd, S., Ghosh, A., Prabhakar, B., & Shah, D. (2005, March). Gossip algorithms: design, analysis and applications. In *Proceedings ieee 24th annual joint conference of the ieee computer and communications societies.* (Vol. 3, p. 1653-1664 vol. 3).

Boyd, S., Ghosh, A., Prabhakar, B., & Shah, D. (2006, June). Randomized gossip algorithms. *IEEE Transactions on Information Theory*, *52*(6), 2508-2530.

Bullo, F., Cortés, J., & Martínez, S. (2009). *Distributed control of robotic networks: A mathematical approach to motion coordination algorithms*. Princeton University Press.

Cao, M., Morse, A., & Anderson, B. (2005). Coordination of an asynchronous multi-agents system via averaging. *IFAC Proceedings Volumes*, *38*(1), 17 - 22. (16th IFAC World Congress)

Cao, M., Morse, A., & Anderson, B. (2008a). Reaching a consensus in a dynamically changing environment: A graphical approach. *SIAM Journal on Control and Optimization*, *47*(2), 575-600.

Cao, M., Morse, A., & Anderson, B. (2008b). Reaching a consensus in a dynamically changing environment: Convergence rates, measurement delays, and asynchronous events. *SIAM Journal on Control and Optimization*, *47*(2), 601-623.

Cao, M., Morse, A. S., Yu, C., Anderson, B. D. O., & Dasguvta, S. (2007, Dec). Controlling a triangular formation of mobile autonomous agents. In *2007 46th ieee conference on decision and control* (p. 3603-3608).

Carli, R., Chiuso, A., Schenato, L., & Zampieri, S. (2008, May). Distributed kalman filtering based on consensus strategies. *IEEE Journal on Selected Areas in Communications*, *26*(4), 622-633.

Cavalcante, R. L. G., & Mulgrew, B. (2010, March). Adaptive filter algorithms for accelerated discrete-time consensus. *IEEE Transactions on Signal Processing*, *58*(3), 1049-1058.

Chen, Y., Tron, R., Terzis, A., & Vidal, R. (2010, Dec). Corrective consensus: Converging to the exact average. In *49th ieee conference on decision and control (cdc)* (p. 1221-1228).

Chen, Y., Tron, R., Terzis, A., & Vidal, R. (2011, June). Accelerated corrective consensus: Converge to the exact average at a faster rate. In *Proceedings of the 2011 american control conference* (p. 3417-3422).

Cortes, J., Martinez, S., & Bullo, F. (2006, Aug). Robust rendezvous for mobile autonomous agents via proximity graphs in arbitrary dimensions. *IEEE Transactions on Automatic Control*, *51*(8), 1289-1298.

Cucker, F., & Smale, S. (2007, May). Emergent behavior in flocks. *IEEE Transactions on Automatic Control*, *52*(5), 852-862.

Daubechies, I., & Lagarias, J. C. (1992). Sets of matrices all infinite products of which converge. *Linear Algebra and its Applications*, *161*, 227 - 263.

Daubechies, I., & Lagarias, J. C. (2001). Corrigendum/addendum to: Sets of matrices all infinite products of which converge. *Linear Algebra and its Applications*, *327*(1), 69 - 83.

Denantes, P., Benezit, F., Thiran, P., & Vetterli, M. (2008, April). Which distributed averaging algorithm should i choose for my sensor network? In *Ieee infocom 2008 - the 27th conference on computer communications* (p. 986-994).

Dimakis, A. G., Kar, S., Moura, J. M. F., Rabbat, M. G., & Scaglione, A. (2010, Nov). Gossip algorithms for distributed signal processing. *Proceedings of the IEEE*, *98*(11), 1847-1864.

Eren, T., Belhumeur, P. N., & Morse, A. S. (2002, Dec). Closing ranks in vehicle formations based on rigidity. In *Proceedings of the 41st ieee conference on decision and control, 2002.* (Vol. 3, p. 2959-2964 vol.3).

Fagnani, F., & Zampieri, S. (2006, Dec). Average consensus with packet drop communication. In *Proceedings of the 45th ieee conference on decision and control* (p. 1007-1012).

Fax, J. A., & Murray, R. M. (2002). Graph laplacians and stabilization of vehicle formations. *IFAC Proceedings Volumes*, *35*(1), 55 - 60. (15th IFAC World Congress)

Fax, J. A., & Murray, R. M. (2004, Sept). Information flow and cooperative control of vehicle formations. *IEEE Transactions on Automatic Control*, *49*(9), 1465-1476.

FIT IoT-LAB. (2018, August). *Iot-lab: a very large scale open testbed.* Retrieved from `https://www.iot-lab.info/`

Fortino, G., & Trunfio, P. (2014). *Internet of things based on smart objects: technology, middleware and applications*. Springer International Publishing.

Frasca, P., Carli, R., Fagnani, F., & Zampieri, S. (2008, Dec). Average consensus by gossip algorithms with quantized communication. In *2008 47th ieee conference on decision and control* (p. 4831-4836).

Frommer, A., & Szyld, D. B. (2000). On asynchronous iterations. *Journal of Computational and Applied Mathematics*, *123*(1-2), 201-216.

Ghosh, B., Muthukrishnan, S., & Schultz, M. H. (1996). First and second order diffusive methods for rapid, coarse, distributed load balancing (extended abstract). In *Proceedings of the eighth annual acm symposium on parallel algorithms and architectures* (pp. 72–81). New York, NY, USA: ACM.

Godsil, C., & Royle, G. (2013). *Algebraic graph theory*. Springer New York.

Goebel, R., Sanfelice, R. G., & Teel, A. R. (2009). Hybrid dynamical systems. *IEEE Control Systems*, *29*(2), 28-93.

Gubbi, J., Buyya, R., Marusic, S., & Palaniswami, M. (2013). Internet of things (IoT): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, *29*(7), 1645-1660.

Gupta, V., Hassibi, B., & Murray, R. M. (2005, Dec). On sensor fusion in the presence of packet-dropping communication channels. In *Proceedings of the 44th ieee conference on decision and control* (p. 3547-3552).

Gurvits, L. (1995). Stability of discrete linear inclusion. *Linear Algebra and its Applications*, *231*, 47 - 85.

Hartfiel, D. (2002). *Nonhomogeneous matrix products*. World Scientific.

He, J., Duan, L., Hou, F., Cheng, P., & Chen, J. (2015). Multiperiod scheduling for wireless sensor networks: A distributed consensus approach. *IEEE Transactions on Signal Processing*, *63*(7), 1651-1663.

Horn, R., Horn, R., & Johnson, C. (1990). *Matrix analysis*. Cambridge University Press.

HP Development Company, L.P. (2017, July). *Hp mini 210-3000 pc series - product information.* Retrieved from `https://support.hp.com/us-en/product/hp-mini-210-3000-pc-series/5082191/product-info`

Iutzeler, F., Ciblat, P., & Hachem, W. (2013, June). Analysis of sum-weight-like algorithms for averaging in wireless sensor networks. *IEEE Transactions on Signal Processing*, *61*(11), 2802-2814.

Iutzeler, F., Ciblat, P., Hachem, W., & Jakubowicz, J. (2012, March). New broadcast based distributed averaging algorithm over wireless sensor networks. In *2012 ieee international conference on acoustics, speech and signal processing (icassp)* (p. 3117-3120).

Iutzeler, F., Ciblat, P., & Jakubowicz, J. (2012, Nov). Analysis of max-consensus algorithms in wireless channels. *IEEE Transactions on Signal Processing*, *60*(11), 6103-6107.

Jadbabaie, A., Lin, J., & Morse, A. S. (2003, June). Coordination of groups of mobile autonomous agents using nearest neighbor rules. *IEEE Transactions on Automatic Control*, *48*(6), 988-1001.

Johansson, B., & Johansson, M. (2008). Faster linear iterations for distributed averaging. *IFAC Proceedings Volumes*, *41*(2), 2861 - 2866. (17th IFAC World Congress)

Jungers, R. (2009). *The joint spectral radius: Theory and applications*. Springer Berlin Heidelberg.

Kar, S., & Moura, J. M. F. (2007a, April). Distributed average consensus in sensor networks with random link failures. In *2007 ieee international conference on acoustics, speech and signal processing - icassp '07* (Vol. 2, p. II-1013-II-1016).

Kar, S., & Moura, J. M. F. (2007b). Distributed consensus algorithms in sensor networks: Quantized data. *CoRR*, *abs/0712.1609*. Retrieved from `http://arxiv.org/abs/0712.1609`

Kar, S., & Moura, J. M. F. (2009, Jan). Distributed consensus algorithms in sensor networks with imperfect communication: Link failures and channel noise. *IEEE Transactions on Signal Processing*, *57*(1), 355-369.

Kashyap, A., Basar, T., & Srikant, R. (2006, July). Quantized consensus. In *2006 ieee international symposium on information theory* (p. 635-639).

Kempe, D., Dobra, A., & Gehrke, J. (2003, Oct). Gossip-based computation of aggregate information. In *44th annual ieee symposium on foundations of computer science, 2003. proceedings.* (p. 482-491).

Kim, Y., & Mesbahi, M. (2006, Jan). On maximizing the second smallest eigenvalue of a state-dependent graph laplacian. *IEEE Transactions on Automatic Control*, *51*(1), 116-120.

Kokiopoulou, E., & Frossard, P. (2007, Oct). Accelerating distributed consensus using extrapolation. *IEEE Signal Processing Letters*, *14*(10), 665-668.

Kokiopoulou, E., & Frossard, P. (2009, Jan). Polynomial filtering for fast convergence in distributed consensus. *IEEE Transactions on Signal Processing*, *57*(1), 342-354.

Lavaei, J., & Murray, R. M. (2012, Jan). Quantized consensus by means of gossip algorithm. *IEEE Transactions on Automatic Control*, *57*(1), 19-32.

Lee, K. K., & Chanson, S. T. (2002). Packet loss probability for real-time wireless communications. *IEEE Transactions on Vehicular Technology*, *51*(6), 1569-1575.

Lin, J., Morse, A., & Anderson, B. (2007a). The multi-agent rendezvous problem. part 1: The synchronous case. *SIAM Journal on Control and Optimization*, *46*(6), 2096-2119.

Lin, J., Morse, A., & Anderson, B. (2007b). The multi-agent rendezvous problem. part 2: The asynchronous case. *SIAM Journal on Control and Optimization*, *46*(6), 2120-2147.

Lin, J., Morse, A. S., & Anderson, B. D. O. (2003, Dec). The multi-agent rendezvous problem. In *42nd ieee international conference on decision and control (ieee cat. no.03ch37475)* (Vol. 2, p. 1508-1513 Vol.2).

Liu, J., Anderson, B. D., Cao, M., & Morse, A. S. (2013). Analysis of accelerated gossip algorithms. *Automatica*, *49*(4), 873 - 883.

Liu, J., & Morse, A. S. (2012, June). Asynchronous distributed averaging using double linear iterations. In *2012 american control conference (acc)* (p. 6620-6625).

Liu, J., Mou, S., Morse, A. S., Anderson, B. D., & Yu, C. B. (2018). Request-based gossiping without deadlocks. *Automatica*, *93*, 454 - 461.

Liu, J., Mou, S., Morse, A. S., Anderson, B. D. O., & Yu, C. (2011, Sept). Deterministic gossiping. *Proceedings of the IEEE*, *99*(9), 1505-1524.

Longo, S., Su, T., Herrmann, G., & Barber, P. (2013). *Optimal and robust scheduling for networked control systems*. CRC Press.

Mehyar, M., Spanos, D., Pongsajapan, J., Low, S. H., & Murray, R. M. (2005, Dec). Distributed averaging on asynchronous communication networks. In *Proceedings of the 44th ieee conference on decision and control* (p. 7446-7451).

Mehyar, M., Spanos, D., Pongsajapan, J., Low, S. H., & Murray, R. M. (2007, June). Asynchronous distributed averaging on communication networks. *IEEE/ACM Transactions on Networking*, *15*(3), 512-520.

Moreau, L. (2005, Feb). Stability of multiagent systems with time-dependent communication links. *IEEE Transactions on Automatic Control*, *50*(2), 169-182.

Mukhopadhyay, S. C., & Suryadevara, N. (2014). *Internet of things: Challenges and opportunities*. Springer International Publishing.

Nedic, A., Olshevsky, A., Ozdaglar, A., & Tsitsiklis, J. N. (2009, Nov). On distributed averaging algorithms and quantization effects. *IEEE Transactions on Automatic Control*, *54*(11), 2506-2517.

Núñez, F., Wang, Y., & Doyle III, F. J. (2012). Bio-inspired hybrid control of pulse-coupled oscillators and application to synchronization of a wireless network. In *2012 american control conference (acc)* (p. 2818-2823).

Nunez, F., Wang, Y., Grasing, D., Desai, S., Cakiades, G., & III, F. J. D. (2017). Pulse-coupled time synchronization for distributed acoustic event detection using wireless sensor networks. *Control Engineering Practice*, *60*, 106 - 117.

Olfati-Saber, R. (2005, June). Ultrafast consensus in small-world networks. In *Proceedings of the 2005, american control conference, 2005.* (p. 2371-2378 vol. 4).

Olfati-Saber, R. (2006, March). Flocking for multi-agent dynamic systems: algorithms and theory. *IEEE Transactions on Automatic Control*, *51*(3), 401-420.

Olfati-Saber, R. (2007, Dec). Distributed kalman filtering for sensor networks. In *2007 46th ieee conference on decision and control* (p. 5492-5498).

Olfati-Saber, R., Fax, J. A., & Murray, R. M. (2007, Jan). Consensus and cooperation in networked multi-agent systems. *Proceedings of the IEEE*, *95*(1), 215-233.

Olfati-Saber, R., & Murray, R. M. (2004, Sept). Consensus problems in networks of agents with switching topology and time-delays. *IEEE Transactions on Automatic Control*, *49*(9), 1520-1533.

Olfati-Saber, R., & Shamma, J. S. (2005, Dec). Consensus filters for sensor networks and distributed sensor fusion. In *Proceedings of the 44th ieee conference on decision and control* (p. 6698-6703).

Olshevsky, A. (2015). Linear time average consensus on fixed graphs. *IFAC-PapersOnLine*, *48*(22), 94 - 99. (5th IFAC Workshop on Distributed Estimation and Control in Networked Systems NecSys 2015)

Olshevsky, A., & Tsitsiklis, J. (2011). Convergence speed in distributed consensus and averaging. *SIAM Review*, *53*(4), 747-772.

Olshevsky, A., & Tsitsiklis, J. N. (2006, Dec). Convergence rates in distributed consensus and averaging. In *Proceedings of the 45th ieee conference on decision and control* (p. 3387-3392).

Oreshkin, B. N., Coates, M. J., & Rabbat, M. G. (2010, May). Optimization and analysis of distributed averaging with short node memory. *IEEE Transactions on Signal Processing*, *58*(5), 2850-2865.

Oróstica, B., & Núñez, F. (2017, Aug). Evaluation of asynchronous average consensus algorithms in pure broadcasting infrastructure-free networks. In *2017 ieee conference on control technology and applications (ccta)* (p. 61-66).

Patterson, S., Bamieh, B., & Abbadi, A. E. (2007, Dec). Distributed average consensus with stochastic communication failures. In *2007 46th ieee conference on decision and control* (p. 4215-4220).

Raspberry Pi Foundation. (2017, July). *Raspberry pi 3 model b.* Retrieved from https://www.raspberrypi.org/products/raspberry-pi-3-model-b/

Ren, W., & Beard, R. W. (2005, May). Consensus seeking in multiagent systems under dynamically changing interaction topologies. *IEEE Transactions on Automatic Control*, *50*(5), 655-661.

Rhodius, A. (1997). On the maximum of ergodicity coefficients, the dobrushin ergodicity coefficient, and products of stochastic matrices. *Linear Algebra and its Applications*, *253*(1), 141 - 154.

Ricardo Sanfelice. (2017, March). *Hybrid equations toolbox v2.03.* Retrieved from https://www.mathworks.com/matlabcentral/fileexchange/41372-hybrid-equations-toolbox-v2-03

Saber, R. O., & Murray, R. M. (2003, June). Consensus protocols for networks of dynamic agents. In *Proceedings of the 2003 american control conference, 2003.* (Vol. 2, p. 951-956).

Sarkar, C., Nambi, A., Prasad, R., Rahim, A., Neisse, R., & Baldini, G. (2015). DIAT: A Scalable Distributed Architecture for IoT. *IEEE Internet of Things Journal*, *2*(3), 230-239.

Schenato, L., & Fiorentin, F. (2011). Average TimeSynch: A consensus-based protocol for clock synchronization in wireless sensor networks. *Automatica*, *47*(9), 1878-1886.

Schenato, L., & Gamba, G. (2007, Dec). A distributed consensus protocol for clock synchronization in wireless sensor network. In *2007 46th ieee conference on decision and control* (p. 2289-2294).

Seneta, E. (2006). *Non-negative matrices and markov chains*. Springer New York.

Singh, K. J., & Kapoor, D. S. (2017). Create your own internet of things: A survey of IoT platforms. *IEEE Consumer Electronics Magazine*, *6*(2), 57-68.

Spanos, D., & Murray, R. M. (2005). Distributed sensor fusion using dynamic consensus..

Strikwerda, J. C. (2002). A probabilistic analysis of asynchronous iteration. *Linear Algebra and its Applications*, *349*(1), 125 - 154.

Sun, Y. G., Wang, L., & Xie, G. (2008). Average consensus in networks of dynamic agents with switching topologies and multiple time-varying delays. *Systems & Control Letters*, *57*(2), 175 - 183.

Tanner, H. G., Jadbabaie, A., & Pappas, G. J. (2003). *Stability of flocking motion* (Tech. Rep.). The GRASP Laboratory, Univ. Pennsylvania.

Tanner, H. G., Jadbabaie, A., & Pappas, G. J. (2007, May). Flocking in fixed and switching networks. *IEEE Transactions on Automatic Control*, *52*(5), 863-868.

TP-Link Technologies Co., Ltd. (2017, July). *Ac600 high gain wireless dual band usb adapter archer t2uh.* Retrieved from `http://www.tp-link.com/us/products/details/cat-5520_Archer-T2UH.html`

TP-Lnk Technologies Co., Ltd. (2017, July). *150mbps high gain wireless usb adapter tl-wn722n.* Retrieved from `https://www.tp-link.com/us/products/details/cat-5520_TL-WN722N.html`

Tsianos, K. I., Lawlor, S., & Rabbat, M. G. (2012a, Oct). Consensus-based distributed optimization: Practical issues and applications in large-scale machine learning. In *2012 50th annual allerton conference on communication, control, and computing (allerton)* (p. 1543-1550).

Tsianos, K. I., Lawlor, S., & Rabbat, M. G. (2012b, Dec). Push-sum distributed dual averaging for convex optimization. In *2012 ieee 51st ieee conference on decision and control (cdc)* (p. 5453-5458).

Tsianos, K. I., & Rabbat, M. G. (2011, Sept). Distributed consensus and optimization under communication delays. In *2011 49th annual allerton conference on communication, control, and computing (allerton)* (p. 974-982).

Tsitsiklis, J., & Athans, M. (1984, January). Convergence and asymptotic agreement in distributed decision problems. *IEEE Transactions on Automatic Control*, *29*(1), 42-50.

Tsitsiklis, J. N., & Blondel, V. D. (1997, Mar 01). The lyapunov exponent and joint spectral radius of pairs of matrices are hard—when not impossible—to compute and to approximate. *Mathematics of Control, Signals and Systems*, *10*(1), 31–40.

Varagnolo, D., Zanella, F., Cenedese, A., Pillonetto, G., & Schenato, L. (2016, April). Newton-raphson consensus for distributed convex optimization. *IEEE Transactions on Automatic Control*, *61*(4), 994-1009.

Vicsek, T., Czirók, A., Ben-Jacob, E., Cohen, I., & Shochet, O. (1995, Aug). Novel type of phase transition in a system of self-driven particles. *Phys. Rev. Lett.*, *75*, 1226–1229.

Vladimirov, A. (2016, March). Continuous products of matrices. *ArXiv e-prints*.

Vladimirov, A., Elsner, L., & Beyn, W.-J. (2000). Stability and paracontractivity of discrete linear inclusions. *Linear Algebra and its Applications*, *312*(1), 125 - 134.

Wang, Y., Núñez, F., & Doyle III, F. J. (2012). Energy-efficient pulse-coupled synchronization strategy design for wireless sensor networks through reduced idle listening. *IEEE Transactions on Signal Processing*, *60*(10), 5293-5306.

Xiao, L., & Boyd, S. (2004). Fast linear iterations for distributed averaging. *Systems & Control Letters*, *53*(1), 65 - 78.

Xiao, L., Boyd, S., & Lall, S. (2005, April). A scheme for robust distributed sensor fusion based on average consensus. In *Ipsn 2005. fourth international symposium on information processing in sensor networks, 2005.* (p. 63-70).

Xiong, G., & Kishore, S. (2009, Aug). Linear high-order distributed average consensus algorithm in wireless sensor networks. In *2009 ieee/sp 15th workshop on statistical signal processing* (p. 529-532).

Zhang, R., & Kwok, J. T. (2014). Asynchronous distributed admm for consensus optimization. In *Proceedings of the 31st international conference on international conference on machine learning - volume 32* (pp. II-1701–II-1709). JMLR.org.