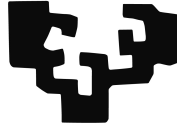


eman ta zabal zazu



UPV EHU

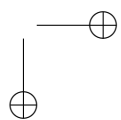
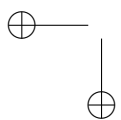
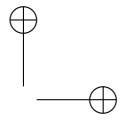
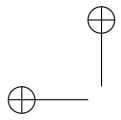
UNIVERSITY OF THE BASQUE COUNTRY UPV/EHU

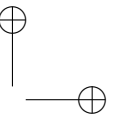
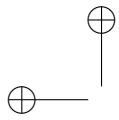
CONTRIBUTIONS ON AGREEMENT IN DYNAMIC DISTRIBUTED SYSTEMS

Department of Computer Architecture and Technology

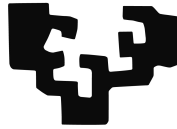
Ph.D. Dissertation presented by Carlos Gómez-Calzado

Advisors: Alberto Lafuente and Mikel Larrea





eman ta zabal zazu



UPV EHU

UNIVERSITY OF THE BASQUE COUNTRY UPV/EHU

CONTRIBUTIONS ON AGREEMENT IN DYNAMIC DISTRIBUTED SYSTEMS

Department of Computer Architecture and Technology

Ph.D. Dissertation presented by Carlos Gómez-Calzado

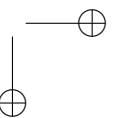
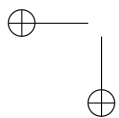
Advisors: Alberto Lafuente and Mikel Larrea

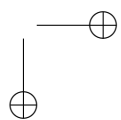
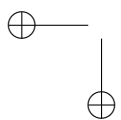
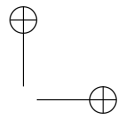
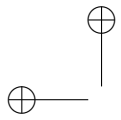
Carlos Gómez-Calzado

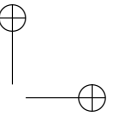
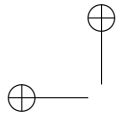
Alberto Lafuente Rojo

Mikel Larrea Álava

Donostia-San Sebastián, June 2015





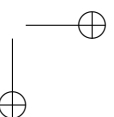
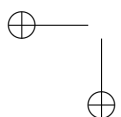


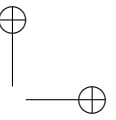
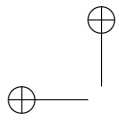
Acknowledgements

Ante todo quiero dar las gracias a mis directores Alberto Lafuente y Mikel Larrea por haberme ayudado en esta tarea tan larga y dura como es una tesis doctoral. Le hemos dedicado mucho trabajo. Tengo clarísimo que no estaría aquí sin vuestra ayuda. Muchas gracias por haber creído en mi.

También quiero agradecer a mis padres (José y Pilar), a mi hermana (Gloria) y a toda la familia en general, por su apoyo en los buenos momentos, pero sobre todo durante los malos momentos. Vosotros me habéis dado la fuerza que necesitaba cuando las mías flaqueaban. Esta tesis es tan vuestra como mía.

Quiero darle las gracias a los profesores del grupo Egokituz (Julio, Luis, Miriam, Nestor) y los que me quedan por mencionar del Grupo de Sistemas Distribuidos (Iratxe y Roberto). Siempre habéis estado dispuestos a ayudar, y se agradece un montón. No puedo olvidarme de la gente de la tercera planta de la Facultad de Informática. Algunos, a día de hoy, seguimos por la universidad (Unai, Borja, Edu, Xabi,...) y otros ya os habéis marchado (Ana, Amaia, Idoia, Raúl y Zigor). Hemos compartido muchas charlas y vivencias durante estos años. Espero que nuestra amistad dure muchos años, independientemente de dónde estemos. Además, quiero agradecer a Antonio Fernandez su apoyo en ciertas partes del trabajo. Ha sido un placer compartir ideas y charlas contigo. Por último, (en español)

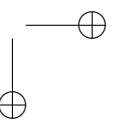
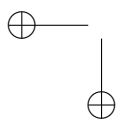


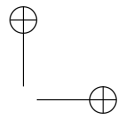
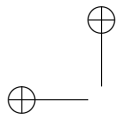


quiero dar las gracias a mis amigos en general. Espero veros más a partir de ahora.

I want to thanks Arnaud Casteigts his hospitality in Bordeaux during my research stay in 2014. It has been a great experience to work with you and your team. I hope it will not be the last time. Thanks also Michel Raynal for the right comments in the precise moment and for contributing in one of the publications of this work.

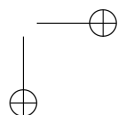
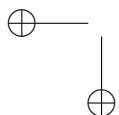
This research has been supported by the Basque Government, under grant IT395-10, the Spanish Research Council, under grants TIN2010-17170 and TIN2013-41123-P, and a doctoral fellowship from the Basque Government (call of 2010). Additionally, part of this work has also been possible thanks to the University of the Basque Country under grant INF11/42, and the support of the Faculty of Informatics of San Sebastián and its the Department of Computer Architecture and Technology.

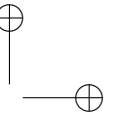
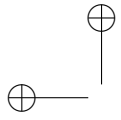




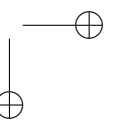
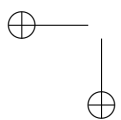
Abstract

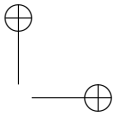
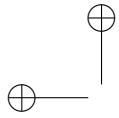
This Ph.D. thesis studies the agreement problem in dynamic distributed systems by integrating both the classical fault-tolerance perspective and the more recent formalism based on evolving graphs. First, we developed a common framework that allows to analyze and compare models of dynamic distributed systems for eventual leader election. The framework extends a previous proposal by Baldoni et al. by including new dimensions and levels of dynamicity. Also, we extend the Time-Varying Graph (TVG) formalism by introducing the necessary timeliness assumptions and the minimal conditions to solve agreement problems. We provide a hierarchy of time-bounded, TVG-based, connectivity classes with increasingly stronger assumptions and specify an implementation of Terminating Reliable Broadcast for each class. Then we define an Omega failure detector, Ω^* , for the eventual leader election in dynamic distributed systems, together with a system model, \mathcal{M}^* , which is compatible with the time-bounded TVG classes. We implement an algorithm that satisfy the properties of Ω^* in \mathcal{M}^* . According to our common framework, \mathcal{M}^* results to be weaker than the previous proposed dynamic distributed system models for eventual leader election. Additionally we use simulations to illustrate this fact and show that our leader election algorithm tolerates more general (i.e., dynamic) behaviors, and hence it is of application in a wider range





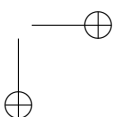
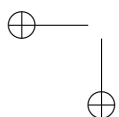
of practical scenarios at the cost of a moderate overhead on stabilization times.

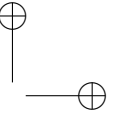
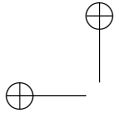




Contents

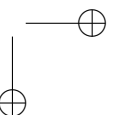
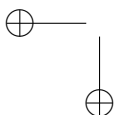
List of Figures	v
List of Tables	ix
1 Introduction	1
1.1 Objectives	4
1.2 Contributions	5
1.3 Roadmap	7
2 Background and Related Work	9
2.1 Distributed Agreement and Related Problems	11
2.1.1 Consensus	11
2.1.2 Eventual Leader Election	12
2.1.3 Terminating Reliable Broadcast	12
2.1.4 Group Membership	13
2.2 Solving Agreement in Distributed Systems	14
2.2.1 Models in Distributed Systems	14
2.2.1.1 Time Models	14
2.2.1.2 Failure Models	18
2.2.2 Solving Agreement with Failure Detectors	22

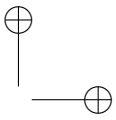
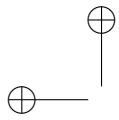




CONTENTS

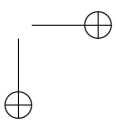
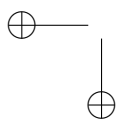
2.3	Solving Agreement in Dynamic Distributed Systems	28
2.3.1	Impossibility Results in Dynamic Distributed Systems	29
2.3.2	Models in Dynamic Distributed Systems	30
2.3.2.1	Time-Varying Graphs	30
2.3.2.2	Connectivity Classes	33
2.3.2.3	A Dynamic Distributed System Categorization	38
2.3.3	Leader Election in Dynamic Distributed Systems	39
2.3.3.1	Implementing Ω in Dynamic Distributed Systems	40
2.3.3.2	Other Dynamic Leader Election Solutions	41
3	Categorizing Dynamic Distributed Systems	45
3.1	A Four-level Categorization	46
3.2	Adding Dimensions	48
3.3	Representing System Models	51
3.4	Conclusions	56
4	Connectivity Models for Solving Agreement	59
4.1	A Timely Model for Dynamic Systems	62
4.1.1	Definitions	63
4.1.2	Terminating Reliable Broadcast in $\mathcal{TC}(\Delta)$	64
4.2	Implementability of TRB	68
4.2.1	(Lower)-bounding the Edge Stability	69
4.2.2	(Upper)-bounding the Edge Appearance	73
4.2.3	Relating Timely Classes	79
4.3	From Δ -TRB to Δ -Consensus in Dynamic Systems	80
4.4	On the Weakest Implementable Timely Connectivity Class	81
4.5	Conclusions	87

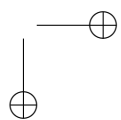
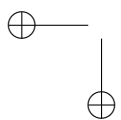
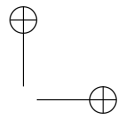
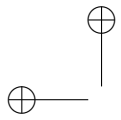


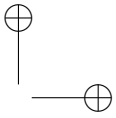
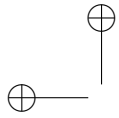


CONTENTS

5	Eventual Leader Election in Dynamic Distributed Systems	91
5.1	Problem Specification and System Model	93
5.1.1	Problem Specification	93
5.1.2	System Model \mathcal{M}^*	96
5.2	A Leader Election Algorithm for \mathcal{M}^*	98
5.2.1	A Reliable Broadcast Primitive for \mathcal{M}^*	98
5.2.2	$\Delta^*\Omega$ Implementation	102
5.2.2.1	Correctness proof of the algorithm	107
5.3	Evaluation	113
5.3.1	Comparing \mathcal{M}^* to Other Models	114
5.3.2	Simulation Examples	116
5.4	Conclusions	119
6	Conclusions	121
6.1	Summary of Contributions	121
6.2	Future Work	124
	Bibliography	127

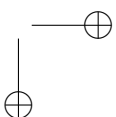
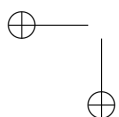


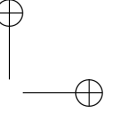
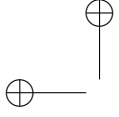




List of Figures

2.1	Relations of inclusion between classes.	36
2.2	The resulting extended relations of inclusion between classes.	38
3.1	An intuition of where (mobile) dynamic distributed systems should be.	47
3.2	Dimensions categorizing mobile dynamic distributed systems.	49
3.3	Graphical representation of a static and synchronous distributed system.	50
3.4	Graphical representation of a totally asynchronous dynamic distributed system.	51
3.5	Representation of the model proposed by Fetzer and Cristian [32].	53
3.6	Representation of the model proposed by Masum et al. [65].	54
3.7	Representation of the model proposed by Ingram et al. [46].	55
3.8	Representation of the model proposed by Melit and Badache [67].	56
3.9	Representation of the model proposed by Arantes et al. [6].	57
4.1	Terminating Reliable Broadcast for $\mathcal{TC}(\Delta)$ executed in a node p	66
4.2	Terminating Reliable Broadcast for $\mathcal{TC}'(\beta)$	70
4.3	Terminating Reliable Broadcast for $\mathcal{TC}''(\alpha, \beta)$	75
4.4	A time-line explaining the Γ upper-bound for the worst case (α, β) -journey from a process p to another process q in the system.	77





LIST OF FIGURES

4.5 Δ -TRB based Δ -Consensus algorithm for $\mathcal{TC}(\Delta)$ 81

4.6 Terminating Reliable Broadcast for $\mathcal{TC}^w(\omega)$ 85

4.7 $\mathcal{TC}''(\alpha, \beta) \subset \mathcal{TC}'(\beta) \subset \mathcal{TC}^w(\omega) \subset \mathcal{TC}(\Delta)$ 88

4.8 The proposed connectivity classes classified in terms of relations of inclusion with respect of class 5 of [16]. 89

5.1 Algorithm implementing Reliable Broadcast by ω -journeys (code for process p_i). 100

5.2 Process states during the algorithm. 102

5.3 Algorithm implementing $\Delta^*\Omega$ in model \mathcal{M}^* (code for process p_i). 103

5.4 Auxiliary functions. 104

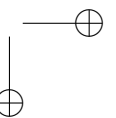
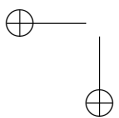
5.5 Figure on the left illustrates the graphical representation of the model \mathcal{M}^* . Figure on the right illustrates the comparison between model \mathcal{M}^* and the model proposed by Fetzer and Cristian [32]. 114

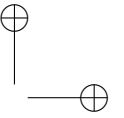
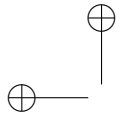
5.6 Figure on the left illustrates the comparison between model \mathcal{M}^* and the model proposed by Masum et al. [65]. Figure on the right illustrates the comparison between model \mathcal{M}^* and the model proposed by Ingram et al. [46]. 115

5.7 Figure on the left illustrates the comparison between model \mathcal{M}^* and the model proposed by Melit and Badache [67]. Figure on the right illustrates the comparison between model \mathcal{M}^* and the model proposed by Arantes et al. [6]. 115

5.8 Screen-shot sequence of a simulation showing how the merge of two graphs leads to a unique leader and how new joins do not affect the leadership. Leaders are represented by white circles. 116

5.9 A continuous joining situation in Melit and Badache's algorithm. The leader is represented in white. 117





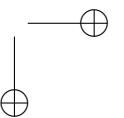
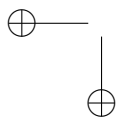
LIST OF FIGURES

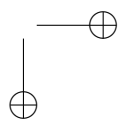
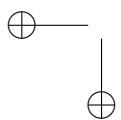
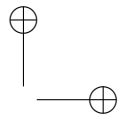
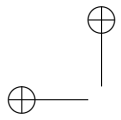
5.10 A periodic mobility pattern in Melit and Badache’s algorithm. The leader is represented in white. 117

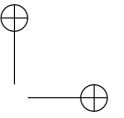
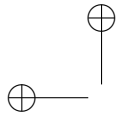
5.11 A varying neighborhood situation in Arante et al.’s algorithm. The leader is represented in white. 118

5.12 The set of initial graphs used for obtaining the convergence metrics of each algorithm studied. 119

5.13 Time to converge a stable leader for the three analyzed algorithms. The simulated environment is composed by 10 processes with the same random topology. 119

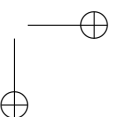
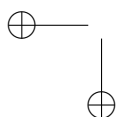


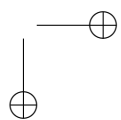
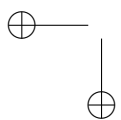
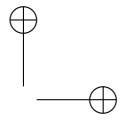
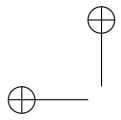


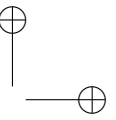
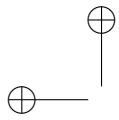


List of Tables

2.1	Classification of the different failure detector classes by their properties.	24
2.2	Classification of dynamic distributed system models by Baldoni et al. [10].	39
3.1	Examples of dynamic system models.	51







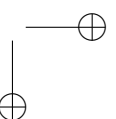
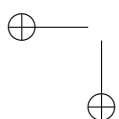
CHAPTER

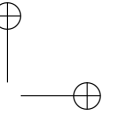
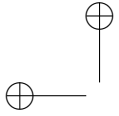
1

Introduction

In nowadays computing applications, devices of any kind, geographically distributed and interconnected by wired or wireless networks, execute pieces of code in a collaborative way. The abstraction of *process* is commonly used to refer to the entity executing the code on a single device. Processes in these *distributed systems* gracefully collaborate to exhibit some kind of useful behavior according to the functionality of the *distributed application*, specified by means of a *distributed algorithm*. Commonly, the collaborative work of processes in a distributed system is boosted by the need of agreement. Researchers in distributed systems have represented agreement problems by the well know paradigm of the *Consensus problem*, which in the last decades has been considered a fundamental problem and its solution has attracted huge attention from theoretical researches and practitioners.

A distributed system is said to be *synchronous* when both the execution speed of processes and the transmission delay of messages are bounded, and the bounds are



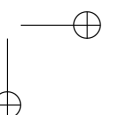
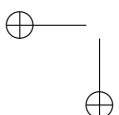


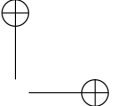
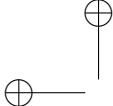
1. INTRODUCTION

known. Solving agreement in such a system is relatively straightforward [23]. On the other hand, in an *asynchronous* system, where there are no time assumptions, it has been shown that it is impossible to solve Consensus in the presence of failures due to the impossibility of distinguishing between a “slow” process and a crashed one (FLP impossibility [34]). For solving agreement and circumvent this impossibility, several models of *partial synchrony* have been proposed where unknown time bounds hold either during the whole execution or eventually [2, 19, 24, 27, 28, 80, 85, 86]. Other proposals, as for example [20, 32], opt for assuming infinitely recurrent good periods where bounds hold followed by bounded periods of asynchrony.

One of the most popular approaches to solve Consensus in models of partial synchrony is based on the abstraction of *unreliable failure detectors* [19], which have been studied for the last two decades. A failure detector encapsulates the required synchrony such that agreement problems that cannot be solved in purely asynchronous systems, e.g., Consensus, become solvable. Among the different classes of failure detectors that have been proposed, Omega (Ω) is of special interest, because it is the weakest failure detector allowing to solve Consensus (assuming a majority of correct processes) [18]. The specification of Ω states that eventually all the correct processes trust the same correct process. In other words, Ω provides an *eventual leader election* functionality. Indeed, a number of leader-based Consensus algorithms have been proposed, e.g., [39, 51, 55, 73, 75].

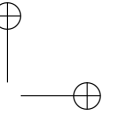
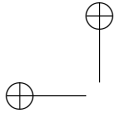
Observe that the models of partial synchrony, as well as unreliable failure detectors, were initially defined for static distributed systems prone to process crash failures. Hence, communication among correct processes was usually assumed to be reliable. Since then, new scenarios have been considered, e.g., process crash-recovery and omission failures [1, 22, 58], unknown membership [7, 47], and more recently system dynamics, including process mobility [59, 78], which makes even the communication among correct processes unpredictable.





Nowadays there are new scenarios where devices include a huge variety of behaviors. Devices like mobile phones, sensors or smart-watches with wireless communication capabilities, lead to new paradigms (e.g., ubiquitous systems [81]) which no longer assume neither a static network nor fully, permanent connectivity among nodes. In general we refer to these scenarios as *dynamic*. Contrary to the well established discipline of static distributed systems, where concepts and terminology are commonly accepted and system models well defined, dynamic distributed systems is a relatively novel field with many faces, which is being boarded from different perspectives, and where much research is necessary to establish concepts, terminology, models and categories. As a result of the mobility, the unplugged power supply and the wireless connection, dynamic distributed systems usually include unavailable and unreliable nodes and links, unknown or unbounded membership and infinite arrival.

Yet, dynamic distributed systems can be modeled using the classical fault-tolerance perspective [6, 59, 67]. For example a system model could include partial synchrony, message omission and crash-recovery failures, and fair lossy channels. However, as pointed by Casteigts et al. [16], in dynamic systems “changes is not anomalies but rather integral part of the nature of the system”. This has encouraged for seeking different approaches to dynamic systems, usually taking the graph theory as a foundation. In this regard the concept of *evolving graphs* [49] or *Time-Varying Graphs* (TVG) [16] has been developed [30, 35, 48, 61, 83]. The TVG framework provides a formalism to describe dynamic networks and introduces the concept of *journey* (a.k.a. temporal path). A journey represents a (multi-hop) communication opportunity between two nodes along time (temporal connectivity). Based on the concept of temporal connectivity, Casteigts et al. define a hierarchy of classes of dynamic networks. One of the classes provides the necessary recurrence to specify agreement problems. However, the TVG approach lacks the necessary mechanisms (i.e. time bounds in communication) to describe the specific assumptions that are required by fault-tolerant agreement



1. INTRODUCTION

algorithms to terminate [40].

The scope of this Ph. D. thesis is agreement in dynamic distributed systems, specifically the models and algorithms that operate in these systems to solve agreement problems. A goal is to integrate different perspectives, on the one hand the classical fault-tolerance approaches, and on the other hand the more recent formalism based on graph dynamics.

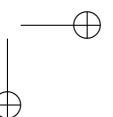
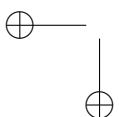
1.1 Objectives

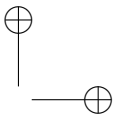
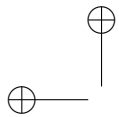
The main objective of this work is the study of agreement in dynamic distributed systems. We mainly focus on leader election and analyze the system model conditions of a distributed dynamic system in which this problem can be solved.

Specific goals of this work are:

- Establishing a framework to provide a reference for analyzing and comparing models of dynamic distributed systems.
- Studying the graph stability conditions required to solve agreement in dynamic distributed systems.
- Defining the properties of eventual leader election and implementing a leader election algorithm for a dynamic distributed system.

To reach these goals, we first study the proposals in the literature from both the classical fault-tolerance theory and the graph theory, specifically the Time-Varying Graph formalism. Furthermore, we search for a common framework to analyze and compare dynamic system models approaches. We take TVG models to find minimal conditions under which agreement problems can be solved in dynamic distributed systems. Then we board eventual leader election in a weak dynamic system model by





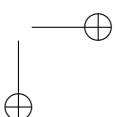
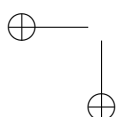
1.2 Contributions

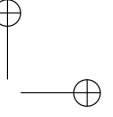
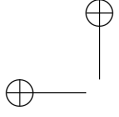
providing formal proofs of the algorithms. Also, we use simulation to compare our approach to other proposals.

1.2 Contributions

The present work contributes to the state-of-the-art of Dynamic Distributed Systems in the following aspects:

1. We present a framework for dynamic distributed systems that allow to compare the system models proposed in the literature for agreement problems, and specifically for leader election [36, 41]. The framework extends the previous categorization proposed by Baldoni et al. [10]. New dimensions are considered to capture all the details of mobility and dynamism. Furthermore, a new level for each dimension is introduced in order to include behaviors which are not considered by the classical proposals of synchrony, asynchrony, and partial synchrony. Models like the one by Cristian and Fetzer [24] can be now compared to the classical proposals and classified in terms of weakness. Interestingly, the four-level scale we propose can be uniformly applied to every dimension in our framework. Furthermore, besides a formal notation, we provide a graphical representation that makes very intuitive to compare system models.
2. We extend the Time-Varying Graph (TVG) formalism by introducing specific timeliness constrains in the recurrent connectivity class proposed by Casteigts et al. We address timeliness from a synchronous point of view, i.e., systems where the transmission delay of messages is bounded and the bound is known a priori by the processes [40]. The set of concepts and mechanisms we provide makes it possible to describe system dynamics at different levels of abstraction and with a gradual set of assumptions, resulting in a hierarchy of time-bounded

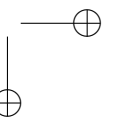
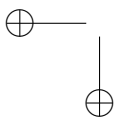


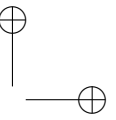
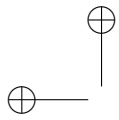


1. INTRODUCTION

TVG classes. We first formulate a very abstract property on the temporal connectivity of the TVG, namely, that the temporal diameter of a component in the TVG is recurrently bounded by Δ . We specify a version of the Terminating Reliable Broadcast problem (TRB) for Δ -components (and their corresponding Δ -journeys), which is related to the ability of solving agreement at component level. Although we prove that Δ -components provide a sufficient concept at the most abstract level to specify a TRB solution, they require oracles that are not directly implementable in message-passing systems. Therefore, we introduce a first constraint to force the existence of journeys whose edges presence duration is lower-bounded by a time β , thereby enabling repetitive communication attempts to succeed eventually. Finally, we introduce a further constrained class of TVG, inspired by the work of Fernández-Anta et al. [5], whereby the local appearance of a link also must be bounded by some duration α , yielding to the concept of (α, β) -journeys. The corresponding algorithms for TRB are specified and proved, and a relation among classes is defined. Furthermore a minimal implementable class based on a lower-bound ω , which is linked to the edge latency, is proposed.

3. We study the leader election problem in mobile and dynamic distributed systems [36]. We define a new Omega failure detector, $\Delta^*\Omega$, together with a system model, \mathcal{M}^* , derived from the eventual leader election model proposed by Larrea et al. for non-mobile dynamic systems [59]. Being compatible with any of the connectivity classes based on Δ -components, \mathcal{M}^* introduces additional assumptions on the graph stability. Specifically, we provide a leader election algorithm that uses the assumptions of w -journeys and introduces the necessary mechanisms to cope with the partial synchrony assumptions of $\Delta^*\Omega$. We provide a correctness proof of the algorithm. Additionally, we have carried out simulations



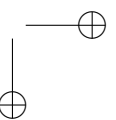
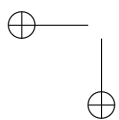


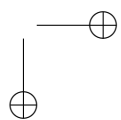
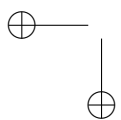
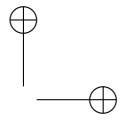
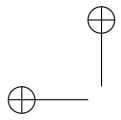
1.3 Roadmap

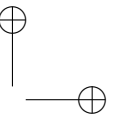
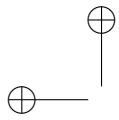
and compare our algorithm to other related leader election algorithms.

1.3 Roadmap

The structure of the rest of this document is the following. In Chapter 2 we define the background on agreement in distributed systems and on dynamic distributed systems. In Chapter 3 we present a common framework for dynamic distributed systems that allow to compare different approaches. Chapter 4 is devoted to extend the TVG formalism to allow to implement agreement algorithms in a model with time bounds. Next, in Chapter 5, we present an eventual leader election algorithm for dynamic distributed systems. Finally, Chapter 6 is devoted to conclude the work and identify the open research lines.







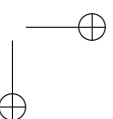
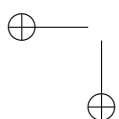
CHAPTER

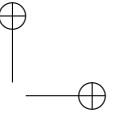
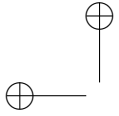
2

Background and Related Work

A distributed system is composed of a set of networked devices such that, in general terms, coordinate their actions in order to achieve a common goal. There exist many different ways for the processes to communicate. However, in this work we only consider message-passing solutions and therefore we leave out of the scope of this work other existing communication paradigms like, for example, shared-memory distributed systems.

The agreement resolution is key in many of today's distributed applications due to their need in any distributed problem at some point. In this sense, the Consensus problem is a central paradigm in distributed systems, as it represents many agreement problems, e.g., leader election, atomic commitment and group membership. Unfortunately the agreement resolution is full of complexities since agreement cannot be





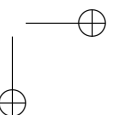
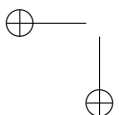
2. BACKGROUND AND RELATED WORK

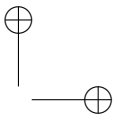
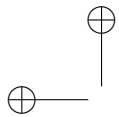
solved in a totally asynchronous systems in presence of failures [34]. Intuitively, in an asynchronous system it is not possible to distinguish between a faulty process and a ‘slow’ process or network. This result, presented by Fischer, Lynch and Patterson, is known as the FLP impossibility. As a consequence of this impossibility, there is no deterministic way to solve Consensus even in presence of only one faulty process in asynchronous systems.

Hence, a fundamental question in distributed computing concerns how processes of the system can agree on a common value despite possible failures. In this regard there exist solutions that circumvent the FLP impossibility providing distributed agreement in presence of faulty processes. One of the purposes of this chapter is to illuminate the issues involved in solving distributed agreement in presence of faulty processes or links.

Most of the research on Consensus has considered a static distributed system with permanent connectivity among processes. In many current distributed systems, however, these assumptions are not valid any more. Instead, these new systems exhibit a dynamic behavior, with process joining the system, leaving it or just moving, which implies uncertain connectivity conditions. Indeed, and unlike in classical static systems, these events are no longer considered incorrect or sporadic behaviors, but rather the natural dynamics of the system. In this chapter we also describe the existing knowledge about dynamic distributed systems, dynamic distributed agreement and evolving networks in general.

The structure of this chapter is the following. In the first section we introduce some of the most important agreement problems in distributed computing. Then, in next section we introduce the concepts of partial synchrony, links and process failure models and we describe some of the existing distributed agreement solutions in static systems. Finally, we introduce the existing background in dynamic systems. More precisely, we describe an existing framework call Time-Varying Graphs, by which we classify the





2.1 Distributed Agreement and Related Problems

existing dynamic models in the literature. We also introduce the dynamic distributed system categorization introduced in [10], which we will extend in the following chapter. As conclusion of this chapter we list some of the existing leader election solutions in dynamic systems.

2.1 Distributed Agreement and Related Problems

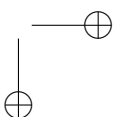
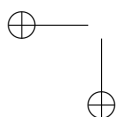
As previously mentioned, the agreement resolution in presence of faulty processes is one of the most studied problems in distributed system problems. More precisely, the Consensus resolution stands out as the most important problem among them, since many distributed problems are equivalent to Consensus. Yet, apart from Consensus, other problems like Group Membership, Leader Election or Terminating Reliable Broadcast are of our interest. In particular, in this work we use the Leader Election problem and the Terminating Reliable Broadcast problem as case studies.

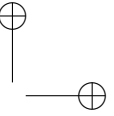
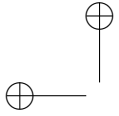
It must be emphasized that some processes of the system can fail at any arbitrary time of the execution. For this reason, processes can be cataloged in terms of how they have behaved during the execution. For one, processes that fail at any time are said to be *incorrect*. For another, processes that never fail are called *correct*.

We proceed to describe which are the properties that must be satisfied in order to solve Consensus properly.

2.1.1 Consensus

The Consensus problem describes how all parties of a distributed system must *decide* on a common value. In a distributed system solving Consensus every process p_i proposes a value v_i . Eventually, every *correct* process eventually calls the primitive $decide(d)$, where d is the same value for all correct processes and is chosen among the set of proposed values v_1, v_2, \dots, v_n .





2. BACKGROUND AND RELATED WORK

Formally, a Consensus implementation has to satisfy the following three properties:

- **Validity:** Every correct process has to decide a proposed value.
- **Agreement:** Every correct process has to decide the same value.
- **Termination:** Every correct process has to decide in a bounded amount of time.

2.1.2 Eventual Leader Election

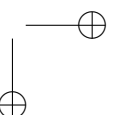
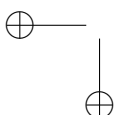
The eventual leader election problem describes how processes *eventually decide* ℓ as their unique, common and *correct* leader. Obviously, the elected leader ℓ is selected among the leaders *proposed* by any of the participants of the system. As a consequence, two sets of processes can be identified in the system: the leader process and the rest of non-leader processes.

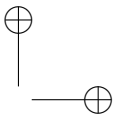
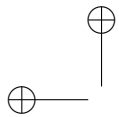
An eventual leader election service implementation must satisfy the following properties:

- **Termination:** Every correct process elects a leader in a bounded time.
- **Integrity:** The elected leader *is a correct* process.
- **Agreement:** Every correct process *elects the same* leader.

2.1.3 Terminating Reliable Broadcast

The Terminating Reliable Broadcast problem (or TRB for short) consists on the broadcasting of a message m to the rest of processes in a system where processes (including the sender) can fail. The resolution of this problem could seem trivial, however, the delivery or not of the message m must be agreed by all correct processes of the system, i.e., either all correct processes deliver m or none of them does it. Instead, if the message m is not delivered, every correct process delivers a special message called SF





2.1 Distributed Agreement and Related Problems

denoting that the sender is faulty. A TRB implementation organizes the system into two subsets of processes: the sender and the receivers.

Formally, a TRB implementation must satisfy the following properties:

- **Termination:** Every correct process delivers some value.
- **Validity:** If the sender is correct and broadcasts a message m , then every correct process delivers m .
- **Integrity:** Every process delivers a message at most once, and if it delivers some message $m \neq \text{SF}$, then m was broadcast by the sender.
- **Agreement:** If a correct process delivers a message m , then all correct processes deliver m .

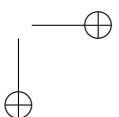
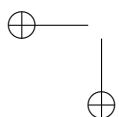
It is important to emphasize that according to [29], Consensus is equivalent to TRB in static synchronous systems.

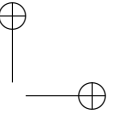
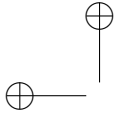
2.1.4 Group Membership

The group membership problem is also equivalent to Consensus [17]. Let us imagine a distributed system with the set of processes where some of them could fail. Eventhough the need of synchrony is not intuitive, for the system to know the active membership at a given time t , first of all, every process in the system must agree on which processes are active at t . Summarizing, a system implementing a group membership service provides a *view* of the membership agreed by every correct process at any time.

Formally, a group membership implementation must fulfill the following properties:

- **Monotonicity:** If a process adopts a view V generated at time t and later adopts another view V' generated at time t' then $t' > t$ and $V' \subseteq V$.





2. BACKGROUND AND RELATED WORK

- **Uniform Agreement:** If some process adopts a view V generated at time t and another different process adopts another view V' generated at time t' then $V = V'$.
- **Completeness:** If a process p fails, then eventually every correct process adopts a view V such that $p \notin V$.
- **Accuracy:** If some process adopts a view V generated at time t with $q \notin V$ such that $q \in \Pi$, then q has failed.

2.2 Solving Agreement in Distributed Systems

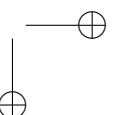
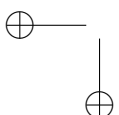
Until now we have described agreement problems in terms of properties. Recall however that the FLP impossibility states that no deterministic agreement can be implemented in an asynchronous and faulty distributed system. In this section we describe partially synchronous distributed systems, which allow circumventing the FLP impossibility.

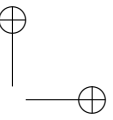
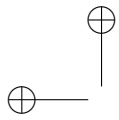
2.2.1 Models in Distributed Systems

We devote this section to describe all concepts surrounding distributed system models.

2.2.1.1 Time Models

The FLP circumvention requires the assumption of a certain degree of synchrony. However, making timeliness assumptions is not the only way of circumventing the FLP impossibility. Moreover, there exist an alternative approach, called failure detectors, that encapsule the partial synchrony required for solving agreement. That way, the agreement protocol focuses only on agreeing on a value relying on the information provided by the failure detector.





2.2 Solving Agreement in Distributed Systems

As the FLP impossibility states, the agreement resolution depends directly on the degree of synchrony provided by the distributed system. Both end points, synchronous and asynchronous distributed systems are defined as follows:

- **Synchronous systems** are those where time bounds exist and are permanently satisfied by processes/links. Moreover, those bounds are a priori known by the system.
- **Asynchronous systems** are those where there exist no timing assumptions. Therefore it cannot be assumed any time bound in the system.

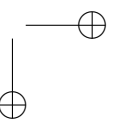
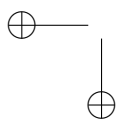
Nevertheless, assuming synchronous distributed systems can be too restrictive in most of real scenarios. This is why alternative synchrony models have been studied.

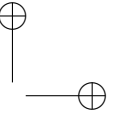
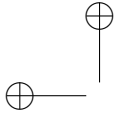
Partial Synchrony Models

A partially synchronous model is weaker than a synchronous model, yet providing enough synchrony to circumvent the FLP impossibility. The concept of partial synchrony is due to Dolev et al. [27]. In this work three kind of asynchrony episodes are identified:

1. Processor asynchrony, that allows processors to "go to sleep" for arbitrarily long time while other processors continue to run.
2. Communication asynchrony, that prevent *a priori* bounds on message transmission time.
3. Message order asynchrony, that results in changing the order in message delivering.

Based on the previous, Dolev et al. state that assuming bounded process execution speed is not enough for solving Consensus, but it is sufficient to assume a bound in





2. BACKGROUND AND RELATED WORK

communication delays or the existence of an order between messages for solving Consensus. Consequently, they propose partial synchrony, which consists on introducing a certain degree of synchrony in the system (without assuming a full synchrony) which allows us to solve agreement in presence of faulty processes. In particular, Dolev et al. introduced 32 models of partial synchrony by the combination of five parameters which can be set to “favorable” or “unfavorable”. Dolev et al. prove that it is necessary the existence of an upper bound on the time for messages to be delivered and an upper bound on the relative speeds of processes to solve Consensus. This discovery was paramount since many researchers started providing new partially synchronous models for solving Consensus.

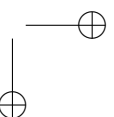
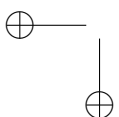
We have classified the existing proposals into three main approaches for providing partial synchrony to a distributed system.

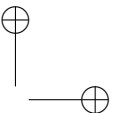
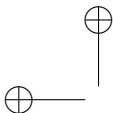
- **Bound-based partial synchrony models:**

- *Dwork, Lynch and Stockmeyer models:* Introduced in [28], the first proposal assumes an unknown bound Δ in the timeliness of the system that holds during the whole execution. The second proposal defines the existence of an a priori known bound in timeliness that is eventually satisfied after a Global Stabilization Time (from now GST).
- *Chandra and Toueg’s model:* Based on the previous definitions of partial synchrony, Chandra and Toueg [19] introduced the combination of both models. The resulting model defines the existence of an unknown bound that holds after GST.

- **Bounded-rate-based partial-synchrony models:**

- *Archimedean model:* Introduced by Vitányi [86], this model assumes a bound in the ratio between the maximum end-to-end delay time and the





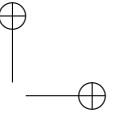
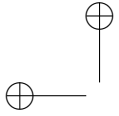
2.2 Solving Agreement in Distributed Systems

minimal computation step time.

- *Θ-model*: Introduced by Le Lann and Schmid [60], this model bounds the ratio between maximal and minimal end-to-end delays of message simultaneously in transit.
- *FAR model*: Introduced by Fetzer et al. [33], it is a model where computing step times and message delays are bounded with a finite average.

- **Behavior-restriction-based partial-synchrony models:**

- *MCM model*: Introduced by Fetzer in [31], this model assumes that all delivered messages are precisely classified as “slow” or “fast”.
- *Timed-Asynchronous model*: Introduced by Cristian and Fetzer [24], this model assumes that the system alternates “stable” periods, where a priori known bound holds, followed by “unstable” periods where bounds can be violated arbitrarily.
- *TCB model*: Introduced by Verissimo et al. [85], this model assumes the existence of a timely subsystem that provides timing failure detection and other time-related services to the asynchronous part of the system.
- *ADD model*: Introduced by Sastry and Pike [80], this model establishes an unknown bound that does not hold perpetually, but is satisfied periodically after an unknown but finite number of events.
- *Weak Timely Link model*: Introduced by Aguilera et al. [2] define a model where only a subset of communications provide a bounded end-to-end delay.
- *Time-Free*: Introduced by Mostefaoui et al. [71, 72], this model allows a total asynchrony assumed the existence of an eventually winning link. An eventually winning link satisfies that eventually a correct process is always



2. BACKGROUND AND RELATED WORK

among the first processes responding in a query-response communication pattern, i.e., there is a process that never responds the last.

As can be observed, partial synchrony has been a well studied field in distributed computing. Nevertheless, a correct distributed system description requires more assumptions than the partial synchrony model itself. In next sections we cover the rest of assumptions required for correctly describe a distributed system model.

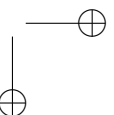
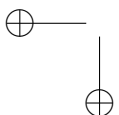
2.2.1.2 Failure Models

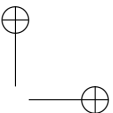
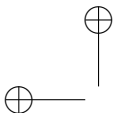
Agreement problems can be solved taking into account one of previous models. Yet, each agreement implementation requires a failure model description. For example, an eventual leader election implementation, for converging to a common leader, could require that faulty processes do not recover during the execution or recover a finite number of times. In the same way, other implementations could allow faulty processes but require that no message is lost. In the following we describe the different failure proposals made in the literature in this regard.

Remember that the FLP impossibility relies on the existence of failures in the system. For this reason, apart from timeliness aspects, a distributed system model has to describe which is the expected behavior of the system. A system is composed of processes and links, therefore, we have to describe which kind of failures can occur at execution time. We proceed to list the some of most relevant failure models regarding processes and links.

Process Failure Models

Apart from determining the synchrony model of the system it is also important to determine the nature of the failures allowed. Introduced by Lamport in [52], a *Byzantine* process, apart from acting asynchronously in terms of time bounds, can also act as an adversary introducing uncertainty in the system. This kind of faulty processes are





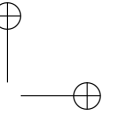
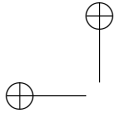
2.2 Solving Agreement in Distributed Systems

cataloged as *adversaries* of the system since they can exhibit a conscious bad behavior in order to explicitly corrupt the system. Among the possible bad behaviors are for example the corruption of messages or the manipulation of the control messages. Lamport showed that in a system composed by n processes where f of them are Byzantine, Consensus cannot be solved if f is $n < 3f + 1$. However, Lamport also stated that, if process failure nature is limited to crashing, the number of faulty processes tolerated in the system increases up to $f \leq (n + 1)/2$.

The basic *Crash-Stop* failure model considers as *correct* processes to those that do not fail by crashing or as *incorrect* processes to those f faulty processes that crash at any time of the execution. Observe that, this basic failure model considers that when an incorrect process crashes it does not recover.

Although the majority of correct processes is still necessary, due to the appearance of new failure models, the meaning of correctness has substantially changed. The *Omission* failure model defines a process as *correct* to the one that neither crashes nor loses messages in its buffers. Besides, *incorrect* processes are those that crash and never recover again, or those that omit messages either in the input buffer or in the output buffer. Observe that processes that omit messages permanently, both in the input buffer and in the output buffer, can be considered as crashed since they cannot communicate.

The *Crash-Recovery* model, introduced in [1, 44], allows four possible behaviors. On the one hand, an *always up* process is the one that never crashes. On the other hand, an *eventually up* process is allowed to crashes and recoveries a finite number of times but there is a time after which it remains behaving correctly for the rest of the execution. On the contrary, an *eventually down* process is also crashes and recovers a finite number of times but there is a time after which it never recovers again. Finally, *unstable* processes can crash and recover as they will. Intuitively, always up processes are considered *correct*. Nevertheless, in some proposals, eventually up processes are



2. BACKGROUND AND RELATED WORK

also considered correct since the solvability of the agreement is assumed to happen eventually.

Summarizing, the existing process failure models are the following:

- Crash-Stop Failure Model
- Omission Failure Model
- Crash-Recovery Failure Model
- Byzantine Failure Model

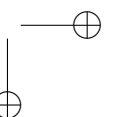
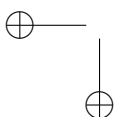
As can be seen, $Crash-Stop \subseteq Omission \subset Crash-Recovery \subset Byzantine$.

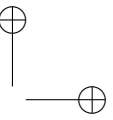
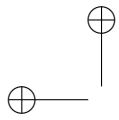
Link Failure Models

In the same way a process can fail, the lose of a message can be considered as a link failure. Even though, this kind of failure was not considered a priori, as time went by researchers realized that the description of this kind of failures is as important as the description of process failures. Unfortunately, some of the link behavior definitions provided in the literature do not clearly divide timeliness assumptions from failure assumptions. This fact makes difficult to identify a pure failure pattern more than a global behavior description. We make an effort in this sense and we describe the existing approaches cataloged by failures and timeliness assumptions.

- **Link failure models:**

- *Reliable links*: Every message sent by this link will be delivered and will not be lost.
- *k-lossy links*: At least one of k message sent is received.
- *ADD links*: There is an unknown number of consecutive message lost after which a message is reliably delivered.

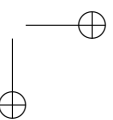
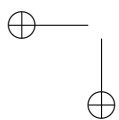


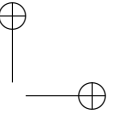
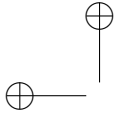


2.2 Solving Agreement in Distributed Systems

- *Fair lossy links*: If it is sent an infinite number of messages then an infinite subset of them will be reliably delivered.
- *Lossy links*: There is no guarantee neither in the delivery time nor in the loss of the message.
- **Link timeliness model:**
 - *Timely delivery*: If a message is delivered is done in a bounded time.
 - *Eventually delivery*: There is a time after which if a message is delivered is done in a bounded time.
- **Combined approaches:**
 - *Timely links*: (Timely delivery + Reliable links) Every message is timely delivered.
 - *Eventually timely links*: (Eventually timely delivery + Reliable links) There is a time after which every message is timely delivered.

Observe that reliable or eventually timely links provide a finite number of message losses while the rest of links allow an infinite number of losses. Among those that allow an infinite number of losses the differences rely on the way that successful messages are delivered. While k -lossy or ADD links provide a reliable communication in a bounded (maybe unknown) number of attempts, fair lossy links and lossy links do not have a priori a successful communication pattern. In essence, for having timely communications, it is important to provide a bounded successful communication pattern. That is the reason why lossy links cannot be used to implement agreement solutions since they are not assumed even to deliver any message.





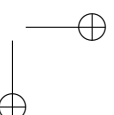
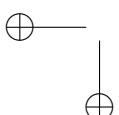
2. BACKGROUND AND RELATED WORK

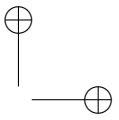
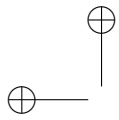
2.2.2 Solving Agreement with Failure Detectors

Regarding the concepts introduced in previous sections researchers were able to provide different solutions to agreement problems in distributed systems. One of the first proposal was the one introduced by Dolev et al. in [27] that showed how Consensus becomes possible with up to $n/2$ faulty processes in partially synchronous systems under the crash-stop failure model.

In general, there exist three main approaches to implement agreement problems in the literature: randomized Consensus, timing assumption based Consensus and failure detectors. Randomized Consensus algorithms are based on probabilistic techniques and provide that a value is decided with probability equal to 1 [8, 11, 14]. In [9] it is provided a survey of some of the most important existing randomized algorithms. Additionally, there exist Consensus solutions in dynamic networks based on randomization [69, 70] (concretely in the wireless networks field). Timing assumptions based Consensus solutions are executed in systems with weak synchrony assumptions. Solutions like those proposed by Dwork et al. in [28] or the one proposed by Dolev et al. in [27] are examples of this kind of solutions. These approaches are explicit in the protocol. Finally, failure detector based agreement solutions solve this problem abstracting from the synchrony matters encapsulating this requirements into failure detectors implementations more than in the implementation of the main agreement solution. In this work we focus on failure detectors, being the rest of approaches out of our scope.

Since their appearance in 1996, *unreliable failure detectors* have attracted the attention of many researchers of the area. Proposed by Chandra and Toueg in their seminal paper [19], an unreliable failure detector is a module executed at each process of a distributed system. The goal of this module is reporting the status for each process composing the distributed system. Unfortunately, occasionally the information can be *unreliable* resulting in false suspicions.



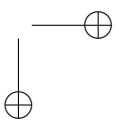
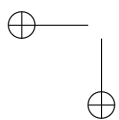


2.2 Solving Agreement in Distributed Systems

In general, detectors assume that eventually an unknown time bound holds not only in processing time but also in message delivery time. This is important since failure detectors operate usually exchanging heartbeat messages with the rest of processes in order to denote their correctness. Consequently the reception of those messages becomes a key element in the implementation. The reception of a heartbeat messages before a threshold expiration denotes the correct behavior of the sender process. Otherwise, if the timer associated to a process p_i expires then p_i is suspected. However, the timeout value associated to p_i is increased in order to converge to a real value in case of a premature false suspicion.

Recall that the information provided may not be reliable. Indeed Chandra and Toueg were aware of it and proposed a family of unreliable failure detectors classified in terms of the precision of the information provided. Failure detectors can be classified in terms of two properties:

- **Completeness:** This property denotes the ability of a failure detector to detect crashed processes.
 - *Strong completeness:* Eventually every process that crashes is permanently suspected by *every* correct process.
 - *Weak completeness:* Eventually every process that crashes is permanently suspected by *some* correct process.
- **Accuracy:** This property denotes the ability of a failure detector to not suspect processes that are still alive.
 - *Strong accuracy:* No process is suspected before it crashes.
 - *Weak accuracy:* Some correct process is never suspected.
 - *Eventual strong accuracy:* There is a time after which correct processes are not suspected by any correct process.



2. BACKGROUND AND RELATED WORK

- *Eventual weak accuracy*: There is a time after which some correct process is never suspected by any correct process.

A failure detector must satisfy both properties in a certain degree to be useful, since the independent satisfaction of a single property is trivial. For example, a “complete” failure detector is the one that suspect every process. On the other hand, an accurate failure detector is the one that does not suspect any process.

Failure detectors are shown in at the Table 2.1 according to the possible properties combinations:

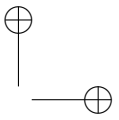
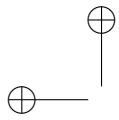
Accuracy	Completeness	
	Strong	Weak
Strong	P	Q
Weak	S	W
Eventual strong	$\diamond P$	$\diamond Q$
Eventual weak	$\diamond S$	$\diamond W$

Table 2.1: Classification of the different failure detector classes by their properties.

There are Consensus solution based on $\diamond P$ like [45, 82].

We want to highlight the Consensus protocol proposed by Chandra and Toueg that it is based on a $\diamond S$ failure detector implementation. The system model adopted for the $\diamond S$ failure detector assumes eventually reliable links and the crash-stop failure model. Chandra et al. [18] proposed another failure detector class, called Omega (Ω). The Ω failure detector class provides eventual agreement on a common and correct leader among all correct processes in a distributed system. An Ω failure detector must satisfy that:

- There is a time after which all the correct processes always trust the same correct process.



2.2 Solving Agreement in Distributed Systems

Chandra et al. proved that Ω is the weakest failure detector for solving Consensus. In the same work, Chandra et al. also proved that Ω and $\diamond S$ are equally strong.

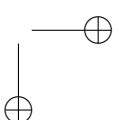
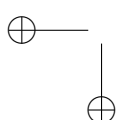
We are interested on the Ω failure detector because is one of our case study. It is important to remark that there exist several Consensus protocol like for example the one proposed by Mostefaoui and Raynal in [73] that explicitly make use of an eventual leader election service. Moreover, one of the most popular Consensus algorithm, Paxos, is a leader-based Consensus protocol. Paxos is presented as a fictional story which describes how the parliament of the Greek island Paxos used to agree their decisions. Yet, the Paxos protocol was published as a journal article [51] by Lamport in 1998, this contribution was written in 1989 . One of the most attractive properties of Paxos is that it can sporadically solve Consensus in asynchronous systems without Byzantine processes. However, Paxos does not contradict the FLP impossibility since for deterministically solving agreement it requires the existence of a stable leader.

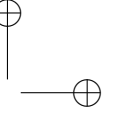
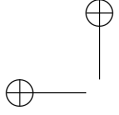
Until now we can conclude that for circumventing the FLP impossibility it is necessary to implement a monitorization mechanism. Since we place this work in the message-passing world, we consider only those monitorization mechanisms implemented using network communication. Thus, we proceed to study which are the communication patterns typically used in this kind of mechanisms.

We can observe two different kind of messages that are *control messages* and *event-messages*. Control messages are sporadic and ensure the fulfillment of a certain invariant in the protocol. Alternatively, event messages have a *periodical* nature and they are used to implement the ratification of process status.

Periodical messages are implemented by one of these two main communication patterns:

- **Polling:** A polling communication based mechanism sends a query message from each process p to another process q . After that p waits for an answer from q . The





2. BACKGROUND AND RELATED WORK

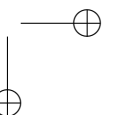
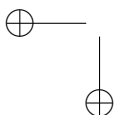
non reception of an answer after a given time makes p suspecting q . Examples of failure detectors based on polling are [53, 54].

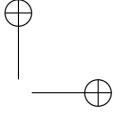
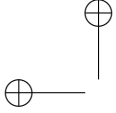
- **Heartbeat:** A heartbeat-based mechanism only requires the periodic send of an alive message from a process q to the rest of processes in the system. If the message is not received after a given time by any process p , p automatically suspects q .

The election of one of this communication pattern has its impact in the scalability of the resulting failure detectors. In fact, it has been recurrently observed that the more complex is the protocol and more message requires, the less scalable is the system. In this regard, Aguilera et al. [2, 3], Larrea et al. [56], and recently Lafuente et al. [50] study the communication efficiency and optimality of failure detector implementations (in particular, failure detector classes Ω and $\diamond P$), defined as follows:

- **Communication efficiency:** A failure detector implementation is communication-efficient if only n links are used forever, n being the number of processes in the system.
- **Communication optimality:** A failure detector implementation is communication-optimal if only c links are used forever, $c \leq n$ being the number of *correct* processes in the system.

These definitions classify failure detector implementations in terms of how good a failure detector uses the network resource, i.e., the number of links used forever. In order to empirically evaluate failure detectors, Chen et al. in [21] proposed a set of metrics that describe the quality of service (QoS) of failure detectors. The proposed metrics are the following:





2.2 Solving Agreement in Distributed Systems

- **Primary metrics:**

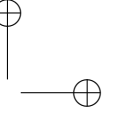
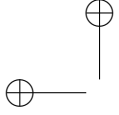
- *Detection time (T_D):* T_D is the time between a process p crashes and another process q starts suspecting p permanently.
- *Mistake recurrent time (T_{MR}):* T_{MR} is the time between two consecutive mistakes.
- *Mistake duration (T_M):* T_M is the time needed to correct a false suspicion.

- **Derived metrics:**

- *Average mistake rate (λ_M):* λ_M is the mistake rate of a failure detector.
- *Query accuracy probability (P_A):* P_A is the probability of having a correct response if the failure detector is queried at an arbitrary time.
- *Good period duration (T_G):* T_G denotes the length of a good period.
- *Forward good period duration (T_{FG}):* T_{FG} represents the time that elapses from a random time where a process q trusts p to the time when q starts trusting again p after a previous suspicion.

These metrics provide us an easy way of comparing different implementations from an empirical view point. This comparison framework is very useful since the obtained results clearly show which failure detector implementation fits better in certain real scenarios.

Regarding process failures, classical implementations of $\diamond P$ and $\diamond S$ usually assume the crash-stop process failure model. However, we believe that, apart from Byzantine process behaviors, the crash-recovery failure model is the weakest dynamic membership behavior that we can expect in a distributed system. A remarkable crash-recovery solution was proposed by Aguilera et al. in [1] where it was presented an adaptation



2. BACKGROUND AND RELATED WORK

of $\diamond S$ called $\diamond S_e$ in the crash-recovery failure model. One of the featured characteristics of $\diamond S_e$ is that it allows solving agreement with a majority of always up processes without stable storage. Additionally, another failure detector called $\diamond S_u$ is presented in the same paper, which allows solving Consensus even with a lower number of always up process compared to the number of incorrect processes. Instead, $\diamond S_u$ requires a number of correct processes higher than the number of incorrect processes and stable storage.¹

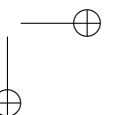
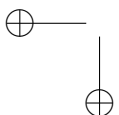
After the proposal of Aguilera et al., several new proposals appeared, almost all of them implementing the Ω failure detector . There exist many Ω implementations that assume failure models like crash-recovery or omission failure models [1, 22, 57, 58, 63, 64] differently to the classic crash-stop failure model. Some of these algorithms are analyzed in terms of QoS in [37].

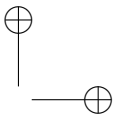
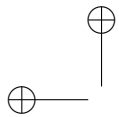
Recently some models with unknown membership [7, 47] and dynamic distributed systems [59, 78] have been proposed. Nevertheless, these proposals and the classical failure models describe a key characteristic of distributed systems that is the dynamicity of the system. In this work we want to adopt these new arising dynamic models as well as the process crash-recovery model. We believe that it is interesting to reach agreement in a totally dynamic distributed system. To do so, we consider that a totally dynamic distributed system should satisfy all the previous dynamicity properties all together. In the following sections we present the needed theoretical concepts in order to achieve the objective of this work.

2.3 Solving Agreement in Dynamic Distributed Systems

So far we have studied how agreement is solved in the presence of failures. However, all this knowledge have been deployed assuming permanent connectivity. As we have

¹In this case, eventually up processes are also considered correct.





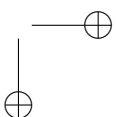
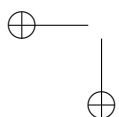
2.3 Solving Agreement in Dynamic Distributed Systems

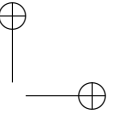
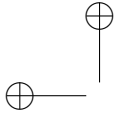
quoted before, a process p isolation because of mobility is equivalent to p 's failure or to the simultaneous failure of the links connecting p with the rest of the processes of the system. Note that, for example, a process connected by a lossy link could be considered as a well-connected process but continuously moving inside and outside of the network graph. Consequently, if links and process failures prevent agreements to converge, intuitively, evolving systems will have the same problem.

There exist several proposals that implement agreement in evolving systems, but they rely on “ad-hoc” connectivity models that sometimes could seem unrealistic. One problem is the lack of a common framework to be able to compare different connectivity models. This is essential, since the basis of knowledge generation starts from having a common language. In this work we propose for this task the use of the time-varying graph framework [16]. We study, using the time-varying graph notation, which are the minimum conditions that a dynamic distributed system has to satisfy in order to allow solving agreement. More concretely, in this section we introduce the necessary concepts needed for understanding this part of the work. We also describe the existing dynamic agreement solutions.

2.3.1 Impossibility Results in Dynamic Distributed Systems

In [13] Biely et al. study the agreement problem in synchronous and directed dynamic graphs. The system model assumes a slotted time approach where messages and graph changes are performed in the beginning of each time slot. Additionally, it is assumed that the whole system compose permanently a strongly connected component, i.e., there is no isolated processes in the system and the system cannot partition at any time. However edge directions and even the presence of them can change between rounds. For example, in round 1 the graph can conform a star with out-coming links from the center to the rest of nodes, and in round 2 the graph can conform a ring with links oriented in the same direction of the clockwise.





2. BACKGROUND AND RELATED WORK

Biely et al. prove that under directional graph assumptions, Consensus cannot be solved if it is not guaranteed an eventual bidirectional connectivity between nodes of the system.

In [5], Fernandez Anta et al. assume opportunistic communication where the changes in the communication topology are created online. In essence, processes are not required to be directly connected at the same time while exist a temporal path that connects them. In [5] the case study is place in the field of Mobile Ad-hoc Networks (MANETs). They assume a slotted time model where communications are sent in the beginning of each time slot, and the time slot is enough for their correct deliverance. Additionally, the graphs evolves synchronously in the sense that graph changes are also performed at the beginning of each slot of time. Under this assumptions, Fernandez Anta et al. study a fundamental problem as it is the information dissemination.

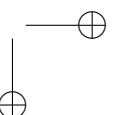
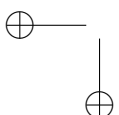
2.3.2 Models in Dynamic Distributed Systems

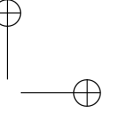
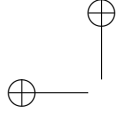
System models are in charge of describing how the system is assumed to behave. Nevertheless, as we have highlighted before, there is no unanimity in the way how a dynamic distributed system must be described. A recent framework called time-varying graphs [16] (TVG, for short), provides the notation needed for describing evolving graphs. Indeed, the TVG framework extends the classical graph theory introducing the required notation to describe evolving graphs.

In this section, first of all we present the TVG framework for describing evolving systems. Next we describe and classify the existing dynamic models in terms of TVG. Finally we introduce an existing dynamic distributed systems classification.

2.3.2.1 Time-Varying Graphs

A recent framework called *time-varying graphs*, proposed by Casteigts et al. [16], aims at providing a precise formalism for describing dynamic networks. As usual, the en-



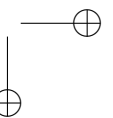
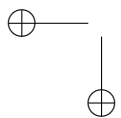


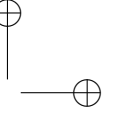
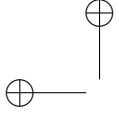
2.3 Solving Agreement in Dynamic Distributed Systems

tities of the system and the communication links between them are represented as a graph. More specifically, a time-varying graph (TVG, for short) is defined as a tuple $\mathcal{G} = (V, E, \mathcal{T}, \rho, \zeta, \psi)$, where:

- V is the set of communicating entities (or nodes, or processes, interchangeably).
- E is the set of edges (or links, interchangeably) that interconnect the nodes in V . In this work, all edges are undirected, i.e., unidirectional links.
- \mathcal{T} is the *lifetime* of \mathcal{G} , i.e., the interval of time over which the graph is defined. It is a subset of the temporal domain \mathbb{T} , itself being \mathbb{N} or \mathbb{R}^+ depending on whether time is discrete or continuous (in this work, it is continuous). For convenience, both endpoints of \mathcal{T} are referred to as \mathcal{T}^- and \mathcal{T}^+ , the latter being possibly $+\infty$.
- $\rho : E \times \mathcal{T} \rightarrow \{true, false\}$, called the *presence* function, indicates whether a given edge is present at a given time (i.e., $\rho(e, t) = true$ if and only if edge e is present at time t).
- $\zeta : E \times \mathcal{T} \rightarrow \mathbb{T}$, called the *latency* function, indicates how long it takes to send a message across a given edge for a given emission time (assuming the edge is present at that time).
- (Optional) $\psi : V \times \mathcal{T} \rightarrow \{true, false\}$, called the *node presence* function, indicates whether a given node is present at a given time (i.e., $\psi(p, t) = true$ if and only if node p is present at time t).

The kind of network we are addressing is possibly disconnected at every instant. Still, a form of communication can be achieved over time by means of *journeys* (a.k.a. *temporal path*). Formally, a journey $\mathcal{J} = \{(e_1, t_1), (e_2, t_2), \dots, (e_k, t_k)\}$ is a sequence such that (e_1, e_2, \dots, e_k) is a valid path in the underlying graph (V, E) , and:





2. BACKGROUND AND RELATED WORK

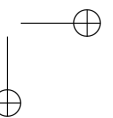
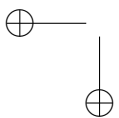
1. for every $i \in [1, k]$, edge e_i is present at time t_i long enough to send a message across (formally, $\rho(e_i, t_i + \delta) = \text{true}$ for all $\delta \in [0, \zeta(e_i, t_i))$).
2. the times when edges are crossed (we also say *activated*) and the corresponding latencies allow a sequential traversal (formally, $t_{i+1} \geq t_i + \zeta(e_i, t_i)$ for all $i \in [1, k)$). What makes this form of connectivity *temporal* is the fact that a journey can pause in between hops, e.g., if the next link is not yet available.

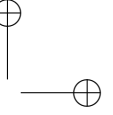
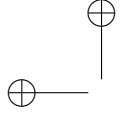
Given a journey \mathcal{J} , $\text{departure}(\mathcal{J})$ and $\text{arrival}(\mathcal{J})$ denote respectively its starting time t_1 and its ending time $t_k + \zeta(e_k, t_k)$. Journeys can be thought of as paths over time, having both a *topological length* k (i.e., the number of *hops*) and a *temporal length* (i.e., a duration) $\text{arrival}(\mathcal{J}) - \text{departure}(\mathcal{J}) = t_k + \zeta(e_k, t_k) - t_1$. Note that journeys describe *opportunities* of communication between an emitter and a receiver. $\mathcal{J}_{\mathcal{G}}^*$ is the set of all such opportunities over \mathcal{G} 's lifetime, while $\mathcal{J}_{(p,q)}^* \subseteq \mathcal{J}_{\mathcal{G}}^*$ are those journeys from p to q . A simplified way of denoting the existence of a journey between a process p and a process q , when the context of \mathcal{G} is clear, is $p \rightsquigarrow q$. Finally, the graph is said to be *temporally connected* if for every $p, q \in V, p \rightsquigarrow q$.

An induced sub-TVG $\mathcal{G}' \subseteq \mathcal{G}$ is obtained by restricting either the set of nodes $V' \subseteq V$ or the lifetime $\mathcal{T}' \subseteq \mathcal{T}$, resulting in the tuple $(V', E', \mathcal{T}', \rho', \zeta', \psi')$ such that:

- (V', E') is the subgraph of (V, E) induced (in the usual sense) by V'
- $\rho' : E' \times \mathcal{T}' \rightarrow \{\text{true}, \text{false}\}$ where $\rho'(e, t) = \rho(e, t)$
- $\zeta' : E' \times \mathcal{T}' \rightarrow \mathbb{T}$ where $\zeta'(e, t) = \zeta(e, t)$
- $\psi' : V' \times \mathcal{T}' \rightarrow \{\text{true}, \text{false}\}$ where $\psi'(e, t) = \rho(e, t)$

If only the lifetime is restricted, say to some interval $[t_a, t_b)$, then the resulting graph \mathcal{G}' is called a *temporal subgraph* of \mathcal{G} and denoted $\mathcal{G}_{[t_a, t_b)}$. The *temporal diameter* of a graph \mathcal{G} at time t is the smallest duration d such that $\mathcal{G}_{[t, t+d)}$ is temporally connected.





2.3 Solving Agreement in Dynamic Distributed Systems

Finally, following Bhadra and Ferreira in [12], we consider a temporal variant of connected components (hereafter, simply called *components*), which are maximal sets of nodes $V' \subseteq V$ such that $\forall p, q \in V', p \rightsquigarrow q$. Two variants are actually considered, whether the corresponding journeys can also use nodes that are in $V \setminus V'$ (*open components*) or not (*closed components*). Observe that a closed component is equivalent to an induced sub-TVG being temporally connected.

2.3.2.2 Connectivity Classes

The study of evolving graphs is a mature research area in computation science. Paradoxically, dynamic distributed agreement has recently started to be considered. A possible reason is the target consumers of these kind of solutions. While graph theory is applicable in many theoretical research topics, the use of distributed agreement is more related to practical implementations (distributed applications). Nevertheless, any connectivity model approach (theoretical or practical) can be useful if it is able to solve dynamic distributed agreement.

In [16] a hierarchy of *classes* of TVG is provided. This hierarchy classifies some of the existing connectivity models into a class dependence-tree. Note that the first five are the bases for describing any existing connectivity model. The TVG classes are the following:

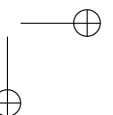
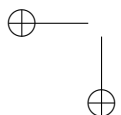
- **Class 1:** At least one node can reach once all the others.

Formally,

$$\exists u \in V : \forall v \in V, u \rightsquigarrow v$$

- **Class 2:** At least one node can be reached once by all the others.

Formally,



2. BACKGROUND AND RELATED WORK

$$\exists u \in V : \forall v \in V, v \rightsquigarrow u$$

- **Class 3:** Connectivity over time. Every node can reach all the others onces.

Formally,

$$\forall u, v \in V, u \rightsquigarrow v$$

- **Class 4:** Round Connectivity. Every node can reach all the others and be reached back afterwards onces.

Formally,

$$\forall u, v \in V, \exists \mathcal{J}_1 \in \mathcal{J}_{(u,v)}^*, \exists \mathcal{J}_2 \in \mathcal{J}_{(v,u)}^* : \text{arrival}(\mathcal{J}_1) \leq \text{departure}(\mathcal{J}_2)$$

- **Class 5:** Recurrent connectivity. Formally,

$$\forall u, v \in V, \forall t \in \mathcal{T}, \exists \mathcal{J} \in \mathcal{J}_{(u,v)}^* : \text{departure}(\mathcal{J}) > t$$

- **Class 6:** Recurrence of edges. Formally,

$$\forall e \in E, \forall t \in \mathcal{T}, \exists t' > t : \rho(e, t') = \text{true} \text{ and } G_{t'} \text{ is connected}$$

- **Class 7:** Time-bounded recurrence of edges.

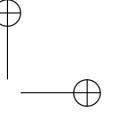
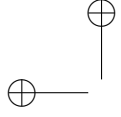
Formally,

$$\forall e \in E, \forall t \in \mathcal{T}, \exists t' \in [t, t + \Delta), \rho(e, t') = \text{true}, \text{ for some } \Delta \in \mathbb{T} \text{ and } G \text{ is connected}$$

- **Class 8:** Periodicity of edges.

Formally,

$$\forall e \in E, \forall t \in \mathcal{T}, \forall k \in \mathbb{N}, \rho(e, t) = \text{true} \rightarrow \rho(e, t + kp) = \text{true}, \text{ for some } p \in \mathbb{T} \text{ and } G \text{ is connected}$$



2.3 Solving Agreement in Dynamic Distributed Systems

- **Class 9:** Constant connectivity.

Formally,

$$\forall t \in \mathcal{T}, G_t \text{ is connected}$$

Under the assumption of a slotted time, it is proved by Biely et al. in [13] that is sufficient to solve Consensus.

- **Class 10:** A *T-interval connectivity model* is introduced in [49]. A graph is *T-interval connected* if and only if for any *T* consecutive time slots exist a common spanning subgraph.

Formally,

$$\forall i \in \mathcal{T}, T \in \mathbb{N}, \exists G' \subseteq G : V_{G'} = V_G, G' \text{ is connected, and } \forall j \in [i, i+T-1], G' \subseteq G_j$$

- **Class 11:** Eventual connectivity.

Formally,

$$\forall i \in \mathbb{N}, \exists j \in \mathbb{N} : j \geq i, G_j \text{ is connected}$$

- **Class 12:** Eventual routability.

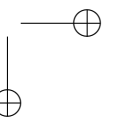
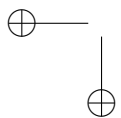
Formally,

$$\forall u, v \in V, \forall i \in \mathbb{N}, \exists j \in \mathbb{N} : j \geq i \text{ and there exists a path from } u \text{ to } v \text{ in } G_j$$

- **Class 13:** Complete graph of interaction.

Formally,

$$G = (V, E) \text{ is complete and } \forall e \in E, \forall t \in \mathcal{T}, \exists t' > t : \rho(e, t') = \text{true}$$



2. BACKGROUND AND RELATED WORK

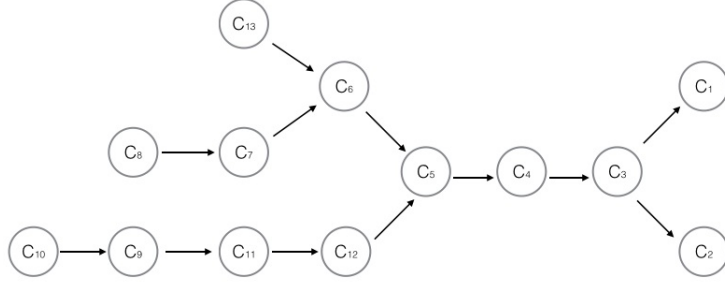


Figure 2.1: Relations of inclusion between classes.

Figure 2.1 shows the relations of inclusion among these classes.

This hierarchy of classes was proposed in 2012 and meanwhile new connectivity models have appeared. As one of the contributions of this work, we extend this hierarchy with the following connectivity models:

- **Class 14:** A model called α, β -connectivity model is proposed in [5]. The (α, β) -connectivity model is defined as the existence of an edge e in at most α time such that e connects two processes p, p' belonging to two different subsets. Additionally, the edge $e = (p, p')$ exists during a time interval of size β . Formally,

$$\begin{aligned}
 & (\forall t \in \mathcal{T}, \forall p, q \in V, \forall (p, q) \in E, \max(\zeta((p, q), t)) \leq \beta) \text{ and} \\
 & (\forall t \in \mathcal{T}, \forall S \subset V, \bar{S} = V \setminus S, p \in S, p' \in \bar{S}, \exists (p, p') \in E, \exists t' \in [t, t + \alpha) : \forall t'' \in \\
 & \quad [t', t' + \beta), \rho((p, p'), t'') = \text{true})
 \end{aligned}$$

- **Class 15:** In [6] it is proposed a novel connectivity model based on the Class 5 but with a stability assumption. The stability condition requires that at least the majority of neighbors of each process remains permanently in the neighborhood. Formally, if we define \mathcal{N}_p^t as the set of neighbors of a process $p \in V$ at time t , then

2.3 Solving Agreement in Dynamic Distributed Systems

$$(\exists p_i \in V : \forall t \in \mathcal{T}, t' \geq t, \exists S \subset \mathcal{N}_{p_i}^t : |S| \geq |\mathcal{N}_{p_i}^t|/2 + 1 \wedge S \subseteq \mathcal{N}_{p_i}^{t'}) \text{ and}$$

$$(\forall t \in \mathcal{T}, \forall p, q \in S \cup p_i, \exists \mathcal{J} \in \mathcal{J}_{p,q}^* : \text{departure}(\mathcal{J}) \geq t)$$

- **Class 16:** In [79] it is proposed a variant of Class 6 which introduces the following assumption: if a link appears its lifetime is lower-bounded by δ , where δ is the maximum latency allowed in a one-hop communication in the system. Formally,

$$\forall e \in E, \forall t \in \mathcal{T}, \exists t' > t : \rho(e, t') = \text{true} \text{ and } G \text{ is connected and}$$

$$(\nexists t'' \in [t', t' + \delta) : \rho(e, t'') = \text{false})$$

- **Class 17:** Also introduced in [79], it is another variant of Class 6 where whenever a recurrent link e appears, e is allowed to be active an arbitrary short period. In exchange, it is assumed that e is present infinitely often during more than 2δ , where δ is the maximum latency allowed in a one-hop communication in the system.

$$\forall e \in E, \forall t \in \mathcal{T}, \exists t' > t : \forall t'' \in [t', t' + 2\delta) \rho(e, t'') = \text{true} \text{ and } G \text{ is connected}$$

- **Class 18:** Recently, Michail et al. [68], have proposed a novel connectivity class that allows the network to be disconnected. It is assumed a finite number of nodes and slotted time, i.e., $\mathcal{T} = \mathbb{N}$. Additionally, if the graph changes it does it at the beginning of each round. Intuitively, the number of changes that can occur in an execution is finite, thus there is a time after which edges reappear, even being the graph disconnected. Observe that this model is equivalent to a (α, β) -connectivity model instanced to $\alpha = n - 1$ and $\beta = 1$.

$$\forall e \in E, \forall t \in \mathbb{N}, \exists t' > t : \rho(e, t') = \text{true} \wedge t' < t + n$$

Figure 2.2 represents relations of inclusion among classes resulting from the extension of Figure 2.1.

2. BACKGROUND AND RELATED WORK

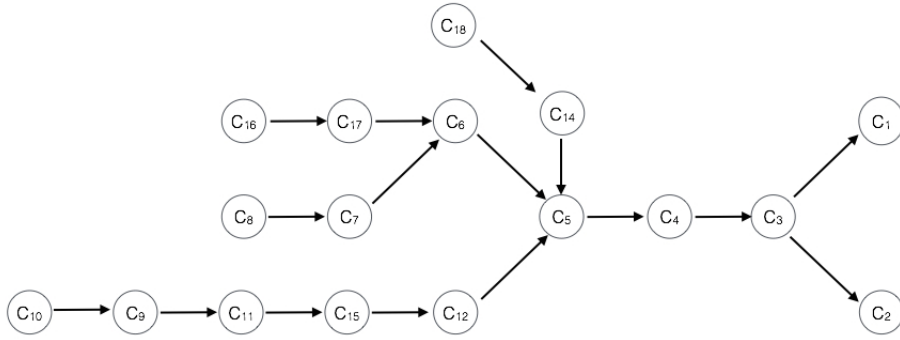


Figure 2.2: The resulting extended relations of inclusion between classes.

2.3.2.3 A Dynamic Distributed System Categorization

Baldoni et al. characterize in [10] dynamic distributed systems in terms of two complementary dimensions:

1. the number of concurrent processes in the system, and
2. the network diameter

Each dimension can be cataloged into three levels depending on the existence and the knowledge of an upper bound. If a bound exists and is known the level is denoted as b , if a bound exists but is unknown the dimension is denoted by n , and finally, if there is no bound the dimension is denoted by ∞ . The combination of dimensions and levels results into six possible dynamic models. For more details see [10]. The Table 2.2 illustrates all possible models regarding the combination of dimensions with levels, excluding non logical ones.

b : a bound exists and it is known a priori by processes in all runs.

n : a bound exists in each run, but it is not known a priori by processes.

2.3 Solving Agreement in Dynamic Distributed Systems

∞ : there is no bound, i.e., the value can grow indefinitely.

For example, a fully-connected synchronous system corresponds to a network diameter of level b whose bound value is 1.

The combination of process concurrency level with network diameter level results in a particular category of dynamic distributed system model. From the nine possible combinations, $M^{b,n}$, $M^{b,\infty}$ and $M^{n,\infty}$ are not considered since the number of concurrent processes bounds the network diameter. The possible six models of interest are shown in Table 2.2.

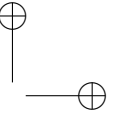
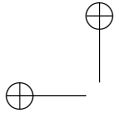
Processes \ Diameter	b	n	∞
b	$M^{b,b}$		
n	$M^{n,b}$	$M^{n,n}$	
∞	$M^{\infty,b}$	$M^{\infty,n}$	$M^{\infty,\infty}$

Table 2.2: Classification of dynamic distributed system models by Baldoni et al. [10].

We consider that dynamic distributed systems are not totally described with the set of dimensions and levels proposed by Baldoni et al. In the following chapter we will extend the set of dimensions and levels with new values that we consider that describe better dynamic distributed systems.

2.3.3 Leader Election in Dynamic Distributed Systems

As a case study of this work we want to provide a way of solving agreement in dynamic systems. In this work we focus on eventual leader election and TRB problems for providing agreement in dynamic distributed systems. On the one hand, from a synchronous view point, TRB has been proven to be equivalent to Consensus. On the other hand, from a partial synchrony view point, there exist several Consensus protocols that rely on an eventual leader election service.



2. BACKGROUND AND RELATED WORK

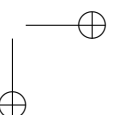
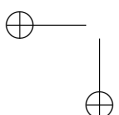
In previous sections we have addressed the leader election problem from a static point of view. However, such static solutions do not handle the possible scenarios that can arise from process movements, infinite arrivals, or system partitions. In the next we list the existing leader election approaches that tolerate a certain degree of dynamicity.

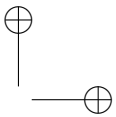
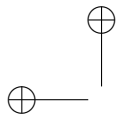
2.3.3.1 Implementing Ω in Dynamic Distributed Systems

In [59] Larrea et al. study the eventual leader election problem from a dynamic membership point of view. In this regard, they redefine the leader election property for adapting it to this kind of scenarios. As a result, they introduce the $\Delta\Omega$ failure detector class that has the following properties:

- **EL-NI (Eventual Leadership in Non-Increasing systems):** If after some time the system does not increase (i.e., no new process joins the system), then a correct leader must eventually be elected.
- **EL-ND (Eventual Leadership in Non-Decreasing systems):** If after some time the system does not decrease (i.e., no process leaves the system or crashes), then (1) a leader must eventually be elected, and (2) the new processes that join the system after a leader has been elected have to eventually consider this process as their leader.

Additionally, they provide two different $\Delta\Omega$ implementations. The first one assumes the existence of a global time service by which processes can take a time stamp and decide whether the actual leader is older or not. The second implementation is based on the sing of an special control message by which processes are able to establish a chronological order in the system. In both cases, the oldest process with the lower identifier becomes the leader of the system. Observe that $\Delta\Omega$ provides the properties for tolerating infinite arrivals, but it is not the only proposal which allows infinite





2.3 Solving Agreement in Dynamic Distributed Systems

arrival. Tucci-Piergiovanni and Baldoni, in [78], also provide an Ω implementation that tolerates infinite arrival.

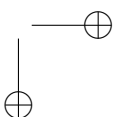
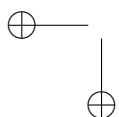
Melit and Badache in [66, 67] propose a classic Ω -based solution for eventually electing a leader per connected partition of a dynamic distributed system. The approach followed by the algorithm is the flooding of the network with a heartbeat message containing the identifier and a priority value chosen aleatory. Melit and Badache's solution is the closest a communication-efficient implementation in multi-hop scenarios, i.e., eventually only the elected leader continues creating new messages while the rest of processes only limit their communication helping the leader messages reaching to the whole system.

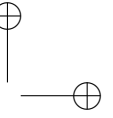
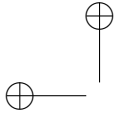
In a recent paper [6], Arantes et al. redefine the proposal made by Greve et al. in [38] in order to provide a classic Ω failure detector implementation able to deal with mobile processes. For the algorithm to converge they assume a *Stabilization Responsiveness Property*, which requires that every neighbor of the leader ℓ to eventually stop moving outside ℓ 's neighborhood.

2.3.3.2 Other Dynamic Leader Election Solutions

The appearance of scenarios that exhibit behaviors not considered by the classical theory, as for example dynamic partitioning or multi-hop communications, resulted in a new family of leader election solutions for this kind of scenarios. These new solutions are not based on the implementation of an Ω failure detector, since the assumptions made do not fit with the ones made in the classical distributed computing literature.

One of the first solutions was proposed by Fetzer and Cristian. In [32], they propose a *local leader election service* useful for partitionable networks. At the time of this proposal, wireless networks were not as common as today. In fact, in this work system partition were considered not by process movements but by link failures. However, as we have previously mentioned, a process movement could be somehow equi-





2. BACKGROUND AND RELATED WORK

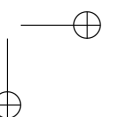
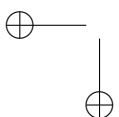
valent to a link failure. The goal of a local leader election service is to elect a local leader per partition. Fetzer and Cristian introduce two properties called Δ -partitions and stable partitions. A Δ -partition is defined as the set of processes that can communicate with each other in a bounded time lower than Δ ¹. Moreover, two processes are Δ -disconnected if the link used to communicate between them does not deliver any message in a time lower than Δ . Based on the Δ -partition definition, it is also provided a definition of stable partition of the system. A Δ -partition is stable if and only if all processes in the partition are mutually connected and are Δ -disconnected from other processes in the system. The leadership criteria proposed in this solution elects a leader per logical partition, assuming that there is a stable partition included in each logical partition. Haddar et al. in [43], following the same approach consider mobile agents to provide a leader election solution in several network topologies.

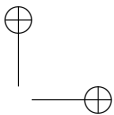
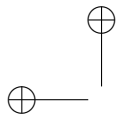
On the other hand, there exist several leader election solutions that take as start point the routing protocol TORA [76]. For example, Malpani et al. in [62] proposed two algorithms based on TORA. This algorithm restrict cycles in their connectivity graph (by means of a Directed Acyclic Graph). One of the algorithms proposed in this paper assumes a single change in the graph at a time, while the second one supports concurrent changes in the communication graph. However, no one of the algorithms has been formally proved, what makes them not as trustworthy as other solutions.

Another TORA-based solution was proposed by Ingram et al. in [46]. Based on [62], they adapt their solution for providing a good election in presence of concurrent process movements.

At the same time Derhab and Badache in [25] proposed another TORA-based solution. As leadership criteria they consider the oldest process as the leader of the system. Nevertheless, they introduce an upper-bound in the number of merges of components.

¹Observe that this definition leaves open the possibility of overlapping between different Δ -partitions.





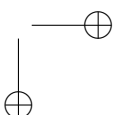
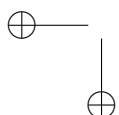
2.3 Solving Agreement in Dynamic Distributed Systems

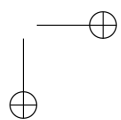
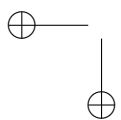
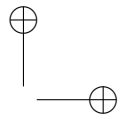
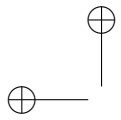
TORA has not been the unique communication protocol adapted for solving leader election. As another example, the Zone Routing Protocol (ZRP) proposed in [42], has been modified by Parvathipuram et al. [77] for solving leader election in dynamic networks.

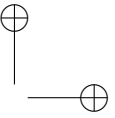
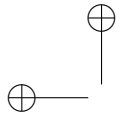
Apart from the adaptations of other protocols, Masum et al. [65] provided an algorithm that elects a local process in the system as the leader. The solution uses as leadership criteria the information provided by the context of the device like battery level or computational capability.

Alslaity and Alwidian in [4] propose a solution that also takes into account the energy consumption. The proposal relies on electing the nodes with best performance value, where the criteria is the density, velocity, battery power, or signal strength. Whenever a process detects a leader crash, it sends a message for starting a new election to a subset of K processes from its neighborhood. The leader eventually is elected from the set of processes that received that message either directly or indirectly.

Another interesting work by Vasudevan et al. [84], propose an algorithm that elects a unique leader in the system despite topological changes. The leader election algorithm is based on the classical termination-detection algorithm by Dijkstra and Scholten [26]. A flooding process p that starts the algorithm and creates a route to the rest of processes. Every process sends back to p a proposal value for being elected as leader. Once p receives all the messages, it decides which of the proposals will be the leader. Then p communicates its decision to the rest of processes by flooding the identity of the new leader.







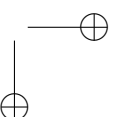
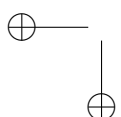
CHAPTER

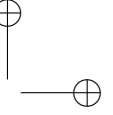
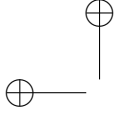
3

Categorizing Dynamic Distributed Systems

In the previous chapter we have described the background concepts on Consensus solvability in both static and dynamic distributed systems. Since the dynamic distributed discipline is not mature, there does not exist a common framework that would allow to extract conclusions in a comparison of dynamic system models. Our goal in this chapter is to identify a set of relevant dimensions, including a common set of levels, which allow to define a framework that should facilitate a systematic system model analysis and comparison.

As shown in Chapter 2, in [10], Baldoni et al. describes a framework that categorize dynamic distributed systems in terms of two complementary dimensions in one of a set of three possible behaviors levels. In essence, the mentioned levels denote the degree of a priori knowledge and the existence of bounds in each dimension.





3. CATEGORIZING DYNAMIC DISTRIBUTED SYSTEMS

In the rest of this chapter we extend the categorization proposed by Baldoni et al. by providing a descriptive framework that will allow to compare dynamic system models. We consider that the framework proposed by Baldoni et al. does not describe all the characteristic aspects inherent to dynamic distributed systems. As we will see, new dimensions should be added in order to capture all the details of dynamic system models, for example, graph stability, graph partitioning or process and link failures.

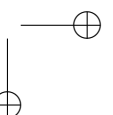
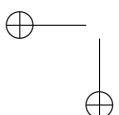
On the other hand, the three-levels framework proposed by Baldoni et al. (bounded and known, unknown but bounded, and unbounded), does not consider other alternative models which have been proposed in the literature. For example, the one by Cristian and Fetzer [24] considers alternate periods of stability (bounds hold) and instability (bounds do not hold). In our framework we will take as a basis this proposal to introduce a new level in each dimension.

To proceed, we first extend the set of levels for the dimensions provided by Baldoni et al. and then we complete the set of dimensions. As we will see, the four-level description can be applied systematically to all the dimensions. Additionally, we use a graphical representation that enable a first-look comparison of system models and we illustrate it by the analysis of a set of example extracted from the literature.

3.1 A Four-level Categorization

Recall that in Baldoni et al.'s categorization each dimension is tagged into three different levels depending on the existence and the knowledge of an upper bound. If a bound exists and it is known, then the level is denoted by b . If a bound exists but it is unknown, then the level is denoted by n . Finally, if there is no bound the level is denoted by ∞ .

In order to characterize (mobile) dynamic distributed systems more precisely, we propose to extend the previous formalism by introducing an additional level, inspired



3.1 A Four-level Categorization

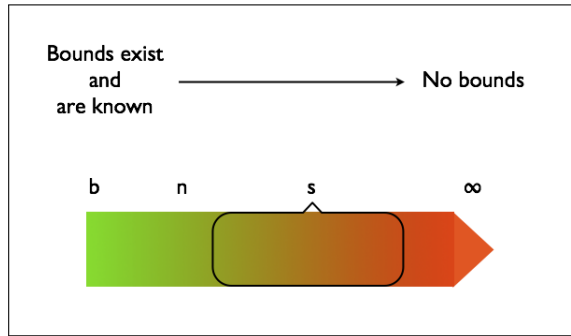


Figure 3.1: An intuition of where (mobile) dynamic distributed systems should be.

by the good/bad period alternation of the timed asynchronous model proposed by Cristian and Fetzer [24]. We denote the new level by s , denoting that the system alternates periods of stability where bounds hold and finite periods of instability where no bound holds. Thus, the new four-level categorization remains as follows:

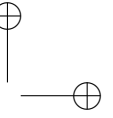
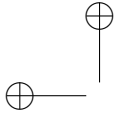
b : a bound exists and it is known a priori by processes in all runs.

n : a bound exists in each run, but it is not known a priori by processes.

s : the system alternates good periods, where a bound exists but it is not known a priori by processes, and bad periods, where there is no bound.

∞ : there is no bound, i.e., the value can grow indefinitely.

Note that s is weaker than n and stronger than ∞ . Observe first that level s is stronger than level ∞ , since existence with dimensions of level ∞ are not required to provide stability periods. On the other hand, observe that level s is weaker than level n . Consider a system model S_1 with an n dimension d_i , where bounds hold after GST. Observe that there exist a system model S_2 equal to S_1 except that in S_2 the dimension d_i is s where d_i bounds could not hold after GST for a finite period of time.



3. CATEGORIZING DYNAMIC DISTRIBUTED SYSTEMS

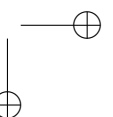
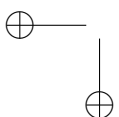
As represented in Figure 3.1 that the proposed level s is weaker than level n but stronger than level ∞ . In fact, level s combines levels n and ∞ in good and bad periods, respectively. Observe also that the new level s can be naturally applied to the process concurrency and network diameter dimensions defined by Baldoni et al. in [10].

3.2 Adding Dimensions

As introduced in Chapter 2, Baldoni et al. characterize dynamic distributed systems in terms of two complementary dimensions, namely the number of concurrent processes in the system and the diameter of the network. For example, consider two systems with known membership N , (i.e., level b), and a known diameter $D < N/2$, (also level b). Consider that S_1 , is static in the sense that processes do not move nor fail. However, consider that in S_2 processes can fail and move. Observe that, in S_2 , since more than half of the process can fail or became disconnected, Consensus could not be solved, contrary to in S_1 . Thus, it is apparent that new dimensions should be added as we will describe next.

As defined in our previous work [36], the dimensions that characterize mobile dynamic distributed systems are:

- **The timeliness of processes and links.** For simplicity, and without loss of generality, processing times are usually considered as negligible with respect to message transmission times.
- **The number of process failures**, which allows addressing crash-recovery scenarios.
- **The number of link failures**, which allows addressing message loses.
- **Graph partitioning**, defined as the number of partitions (i.e., disconnected graphs) allowed during the execution of the system. Graph partitioning is denoted by b if



3.2 Adding Dimensions

the number of partitions is bounded and the bound is known, by n if the number of partitions is bounded but the bound is unknown, by s if the number of partitions alternates between bounded but unknown (good) periods and (bad) periods where the number of partitions is unbounded, and by ∞ if there is no bound on the number of partitions.

- **The number of processes in a graph** (similar to the process concurrency dimension of [10]).
- **The diameter of the graph** (similar to the network diameter dimension of [10]).
- **Graph stability**, modeled in terms of a bound in the number transitions in the link status (the link exists or the link does not exist). Similarly, for this dimension b denotes having a bounded and known number transitions, n denotes having a bounded but unknown number link transitions, s denotes the alternation of good periods with a bounded but unknown number of link transitions and bad periods with unbounded number of link transitions, and ∞ denotes no bound on the number of link transitions.

Dimension ($L \in \{b, n, s, \infty\}$)	Bound corresponds to...
Timeliness (T^L)	... processing and message transmission time
Process Failures (\mathcal{P}_F^L)	... the number of failures a process can suffer
Link Failures (\mathcal{C}_F^L)	... the number of message losses on any link
Graph Partitioning ($G_{\#}^L$)	... the number of graphs in the system
Graph Membership (G_{Π}^L)	... the number of processes in any graph
Graph Diameter (G_D^L)	... the diameter of any graph
Graph Stability (G_S^L)	... the graph stability bound or link transitions

Figure 3.2: Dimensions categorizing mobile dynamic distributed systems.

3. CATEGORIZING DYNAMIC DISTRIBUTED SYSTEMS

In [36] we introduce a formalism to denote the aforementioned dimensions as summarized in Figure 3.2. Following our approach, the compositions of those dimensions with a particular level per dimension denotes a dynamic distributed system model, formally:

$$\mathcal{M}(T^L, \mathcal{P}_F^L, \mathcal{C}_F^L, G_{\#}^L, G_{\Pi}^L, G_D^L, G_S^L)$$

We introduce now a graphical representation that provides a first-look comparison between models. This representation use a radar chart in which each dimension is represented as a radio of the chart where levels are represented from the center to the outside in increasing degree of weakness. Observe that, the higher the area covered in the radar chart, the weaker the system model. To illustrate this, we represent the two extreme trivial cases: a totally synchronous and static distributed system (Figure 3.3) and a totally asynchronous dynamic distributed system (Figure 3.4).

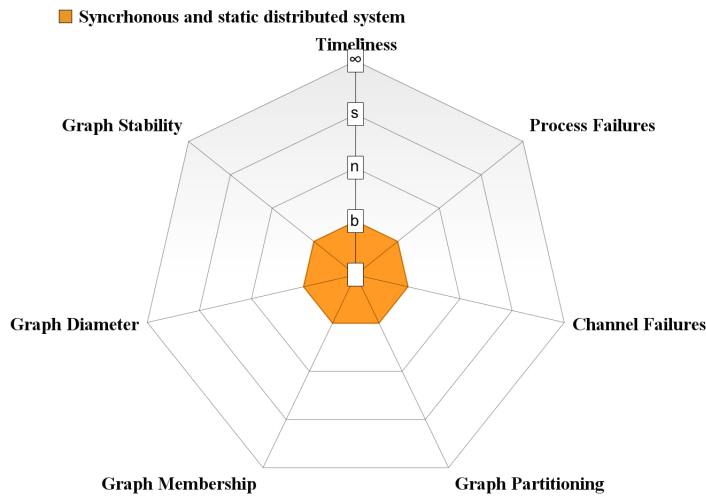


Figure 3.3: Graphical representation of a static and synchronous distributed system.

3.3 Representing System Models

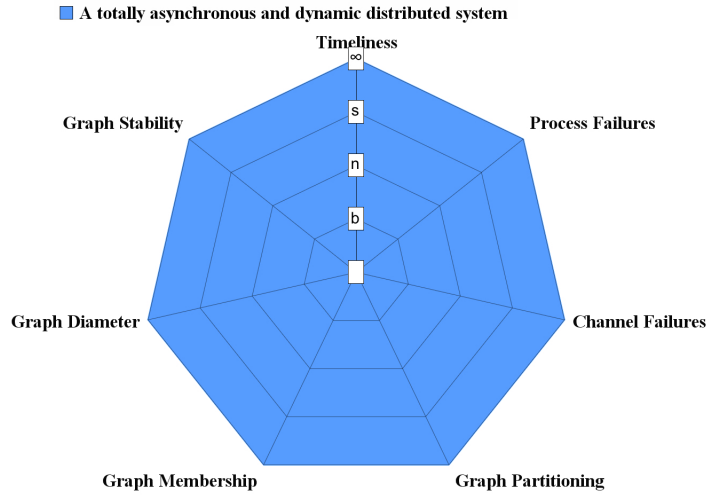


Figure 3.4: Graphical representation of a totally asynchronous dynamic distributed system.

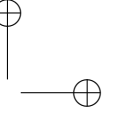
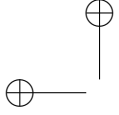
3.3 Representing System Models

In this section we use our framework to categorize some of the system models proposed for leader election algorithms that we have studied in Chapter 2. In Table 3.1 summarizes the models that we will describe next.

Model	Reference
$\mathcal{M}(T^b, \mathcal{P}_F^b, C_F^\infty, G_\#^n, G_\Pi^n, G_D^b, G_S^b)$	Fetzer and Cristian [32]
$\mathcal{M}(T^\infty, \mathcal{P}_F^b, C_F^b, G_\#^n, G_\Pi^n, G_D^n, G_S^n)$	Ingram et al. [46]
$\mathcal{M}(T^b, \mathcal{P}_F^b, C_F^b, G_\#^n, G_\Pi^n, G_D^b, G_S^n)$	Masum et al. [65]
$\mathcal{M}(T^n, \mathcal{P}_F^n, C_F^n, G_\#^n, G_\Pi^n, G_D^n, G_S^n)$	Melit and Badache [67]
$\mathcal{M}(T^\infty, \mathcal{P}_F^b, C_F^\infty, G_\#^n, G_\Pi^n, G_D^n, G_S^s)$	Arante et al. [6]

Table 3.1: Examples of dynamic system models.

For the models analyzed from the literature, we have interpreted the system model definition in terms of our formalism and extracted the parameter values accordingly. In



3. CATEGORIZING DYNAMIC DISTRIBUTED SYSTEMS

some cases, the system model specification were incomplete. In this cases we extract the parameters for the categorization by the inspection of the algorithm. On the other hand, some system models assume an ∞ level in some of their dimensions. As we will see, in these cases, in general there are implicit extra assumptions in the system model or in the algorithm.

Fetzer and Cristian [32]

Fetzer and Cristian do not assume a mobile scenario, however we consider this model as dynamic because they assume network partitions due to network traffic delays. In this work, Fetzer and Cristian propose a model based on a global approach including good and bad periods, and encapsulates link and process failures. We had categorized separately process and link failure models by inspection of the algorithm.

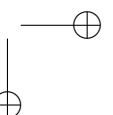
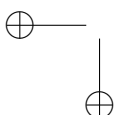
Regarding the rest of dimensions, on the one hand, we can clearly specify that the membership is finite but unknown, being the membership dimension identified as a n level. On the other hand, they assumes a fully-connected network which is translated into a b level for the diameter dimension.

Figure 3.5 illustrates the model proposed by Fetzer and Cristian according the framework presented in this Chapter.

Masum et al. [65]

Another interesting proposal is presented in [65] by Masum et al. which is similar to [32]. The system described by Masum et al. shares with [32] the categorization of processes failures, graph membership and graph diameter. However, they differ in the assumption of totally reliable and timely links (T^b and C^b), and in the assumption of a dynamic topology which eventually stops moving, i.e., G_S^n .

Figure 3.6 represents the model proposed by Masum et al.



3.3 Representing System Models

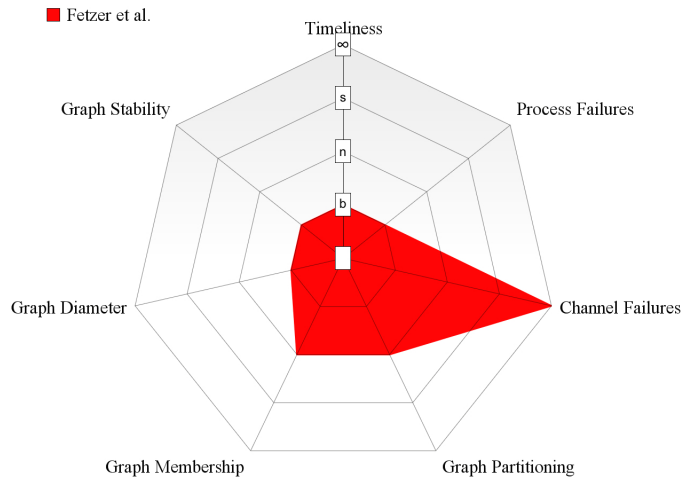


Figure 3.5: Representation of the model proposed by Fetzer and Cristian [32].

Ingram et al. [46]

Ingram et al. in [46] present a weaker timeliness link model where there is no bound in message transmission time, but all messages are eventually delivered, i.e., links are eventually reliable (level n). In this proposal it is assumed a crash failure model and a finite but unknown membership. Observe that since the membership is finite, the graph diameter will be also bounded.

Finally, Ingram et al. assume that eventually the topology stops evolving. Consequently, the number of partitions and the stability are denoted by the n level in both graph partitioning and graph stability dimensions. Figure 3.7 illustrates the model proposed by Ingram et al. using our formalism.

For the ∞ timeliness dimension, they assume synchrony between process clocks. Moreover, a common feature of the aforementioned proposal is the existence of an

3. CATEGORIZING DYNAMIC DISTRIBUTED SYSTEMS

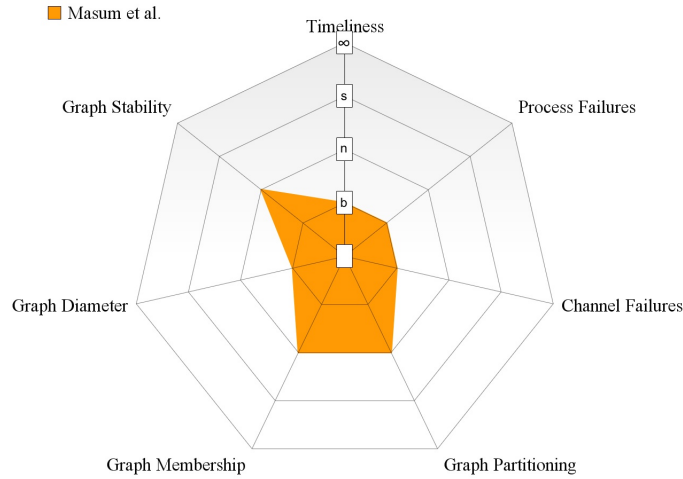


Figure 3.6: Representation of the model proposed by Masum et al. [65].

oracle that provides a list of correct neighbors. This oracle encapsulates the additional synchrony requirements to solve leader election.

Melit and Badache [67]

Melit and Badache propose a dynamic eventual leader election algorithm based on the Ω failure detector. They assume a system model which follows an eventual approach in every dimension, i.e., their proposal assumes that after a time GST, messages are reliably and timely delivered, and the topology does not change. Observe that, even they assume a crash-recovery failure model, the requirement of an eventually static topology prevent process failures after GST. In essence, the eventually static graph assumption makes the system unable to cope with unstable processes and prevents the inclusion of new processes. Consequently, the membership will be unknown but finite, and thus, the number of failures are arbitrary high but finite. In fact, due to the

3.3 Representing System Models

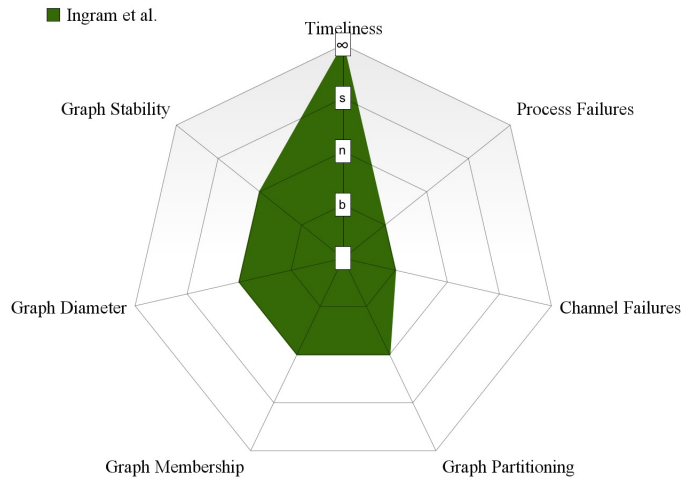


Figure 3.7: Representation of the model proposed by Ingram et al. [46].

finite membership, the graph diameter will be also unknown but bounded. Figure 3.8 illustrates the model proposed by Melit and Badache.

Arantes et al. [6]

Based on a time-free approach, Arantes et al. assume a system model where processes can crash and links behave as fair-lossy. Additionally, they assume that the system eventually converge to a unique partition and the membership is unknown but finite. For the system to converge in [6], the neighborhood of the leader must not change, i.e., once a process becomes a neighbor of the leader, it remains neighbor of leader forever. This introduces a bound in the mobility of nodes (graph stability), and we represent this dimension by level s . Note that, although the number of link changes is unbounded, some good behavior is expected for the transitions of the links with the leader.

3. CATEGORIZING DYNAMIC DISTRIBUTED SYSTEMS

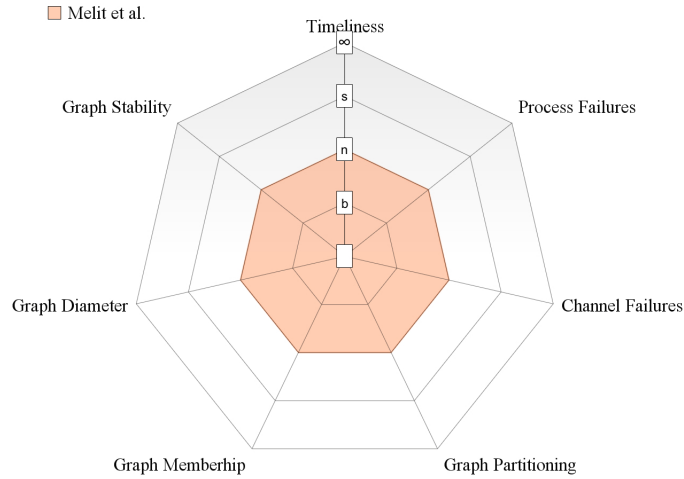


Figure 3.8: Representation of the model proposed by Melit and Badache [67].

Figure 3.9 illustrate the model proposed by Arantes et al.

3.4 Conclusions

In this chapter we have extended the formalism introduced by Baldoni et al. [10] in order to provide a framework and a notation which has been revealed very useful to compare our system to other proposals in the literature. Basically, the formalism defines a range of levels on the bound of several system dimensions (e.g., network diameter or process synchrony). A feature of the formalism is its uniformity: the same range of levels is defined for all dimensions of the system. As a new level, we have included the behavior representing the alternation of bounded and unbounded periods, inspired on the timed asynchronous model of Cristian and Fetzer [24].

An eventually stable model defines unknown bounds under which good things fi-

3.4 Conclusions

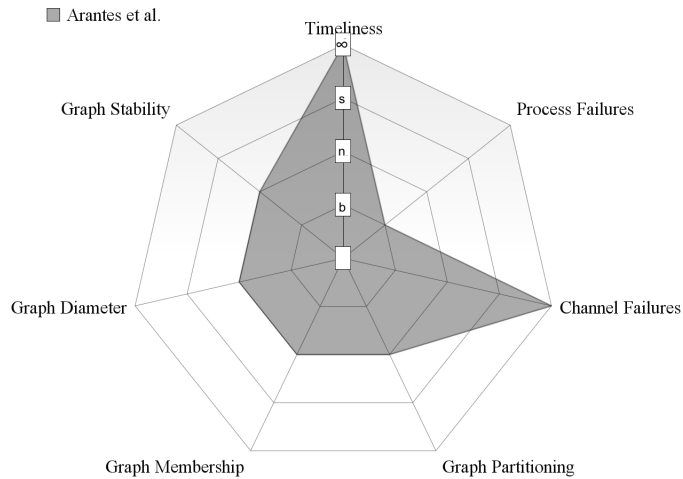
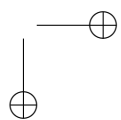
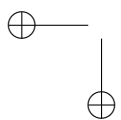
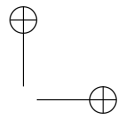
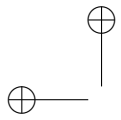
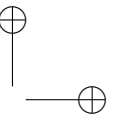
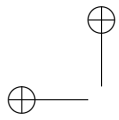


Figure 3.9: Representation of the model proposed by Arantes et al. [6].

nally happen. Those bounds hold forever after a Global Stabilization Time (GST). Observe that eventually stable systems do not guarantee anything before GST and do not allow unexpected behaviors after GST. Eventually stable models are more restrictive than “periodically” stable systems. Indeed, a periodically stable approach can provide a dynamic system with the flexibility required for dealing with continuously changing scenarios. Said this, in periodically stable approaches it is necessary to identify which properties should be defined to provide some minimal stability and how long those properties must hold. In other words, an answer to the question of what *stable enough* and *long enough* mean, for each dimension of the system, should be given. We will board this questions in the next chapters.





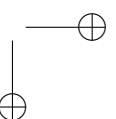
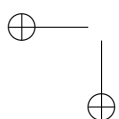
CHAPTER

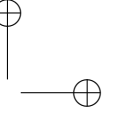
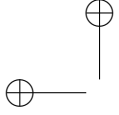
4

Connectivity Models for Solving Agreement

As we have mentioned, most of the research on Consensus has considered a static distributed system with permanent connectivity among processes. In many current distributed systems, however, these assumptions are not valid any more. Instead, these new systems exhibit a dynamic behavior, with processes joining the system, leaving it or just moving, which implies uncertain connectivity conditions. Indeed, and unlike in classical static systems, these events are no longer considered incorrect or sporadic behaviors, but rather the natural dynamics of the system.

Clearly, even the synchrony assumptions of classical (static) models of distributed systems are not enough to solve agreement problems in dynamic systems. For example, having an upper bound on link latencies is pointless if the link is not available at the time of transmission of the message. Note however that the processes could





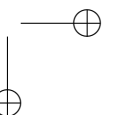
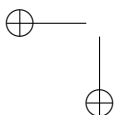
4. CONNECTIVITY MODELS FOR SOLVING AGREEMENT

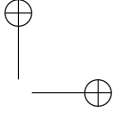
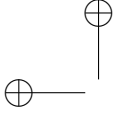
still communicate using an alternative path in the network. Thus, assumptions should consider the overall system connectivity, which encourages for a holistic approach to model dynamic distributed systems.

As we have seen in Chapter 2, in recent years there was a rising interest in modeling dynamic distributed systems from the perspective of graph theory. However, regarding Consensus, none of them lowers the assumptions to the realm of *temporal* connectivity, i.e., not requiring that the graph be connected at every instant, but only that paths exist over time and space (temporal path, aka *journeys*). We believe that the Time-Varying Graph formalism [16] (TVG, for short) provides a useful qualitative framework to model dynamic distributed systems. Among the TVG classes provided by [16], the *recurrent connectivity* class requires that a journey exists between any two processes infinitely often (that is, recurrently). Nevertheless, this class lacks the necessary timeliness (i.e., time bounds in communication) to describe the specific assumptions that are required by synchronous agreement algorithms, such as TRB, to terminate.

In this chapter we extend the recurrent connectivity class by introducing timeliness constraints, together with practical considerations, and analyze the impact of these new constraints on solving Consensus. Thus, we address timeliness in evolving systems (i.e., TVG) from a synchronous point of view, i.e., systems where the transmission delay of messages is bounded and the bound is known a priori by the processes. The resulting set of concepts and mechanisms make it possible to describe system dynamics at different levels of abstraction and with a gradual set of assumptions.

We first formulate a very abstract property on the temporal connectivity of the TVG, namely, that the temporal diameter (i.e., maximum duration of a journey) of a component in the TVG is recurrently bounded by Δ . We refer to such a component as a Δ -*component*, and define the concept of correct process in terms of this component.

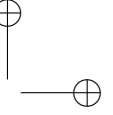
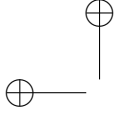




We then specify a version of the Terminating Reliable Broadcast problem (TRB) for Δ -components, which we relate to the ability of solving agreement at component level.

Although Δ -components are proven to be a sufficient concept at the most abstract level, they rely on very inefficient (and hardly implementable) communication patterns in message-passing systems. Indeed, the solution to TRB proposed in this abstract model requires the existence of an oracle that provides the algorithm an instantaneous knowledge of the appearance of an edge. Unfortunately, this oracle does not have a straightforward implementation in terms of real processes and communication links. Therefore, we introduce a first constraint to force the existence of journeys whose edges presence duration is lower-bounded by some duration β (which holds a relation to the maximal latency of a link), thereby enabling repetitive communication attempts to succeed eventually. These journeys are called β -journeys and their existence makes it possible to implement the TRB algorithm without an oracle. We then look at a further constrained class of TVG, inspired by the work of Fernández-Anta et al [5], whereby the local appearance of the edge used by every next hop of (at least one of the possibly many) β -journeys also must be bounded by some duration α , yielding to the concept of (α, β) -journeys. The existence of recurrent (α, β) -journeys allows the processes to stop sending a message α time after they receive it, which is much more efficient. Note that, as we will discuss in this chapter, the introduction of new restrictions decrease the number of potential behaviors achieved by the connectivity model. Intuitively, the number of different timely graphs based on β -journeys is lower than the number of graphs based on Δ -journeys since Δ -journeys have not the extra restriction introduced in β -journeys. Similarly, number of graph based on (α, β) -journey is lower than the number of graph based on β -journeys.

Recall that one of the goals of this work is study how the Consensus solvability is affected by evolving networks. Even the described connectivity models are weak, we consider that exists a weaker connectivity class that we call ω -journeys. ω -journeys



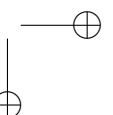
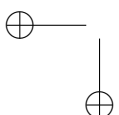
4. CONNECTIVITY MODELS FOR SOLVING AGREEMENT

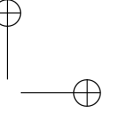
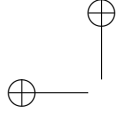
introduce a common lower-bound ω in the active time of every edge e of the system. This new bound makes e be active during a time strictly higher than the latency of e at any moment. We prove that \mathcal{TC}^ω , a ω -journey connectivity class, is the weakest Δ -component based connectivity class that allows solving agreement.

In this chapter we define the abstract timely connectivity model based in a new timely component definition called Δ -components. Additionally, we show how the TRB problem can be solved with respect to Δ -components. However, this TRB implementation requires the assumption of an “oracle” which implementation is far from the reality. Then, we introduce β -journeys (and the corresponding β -components), which we show to be sufficient to implement an effective (i.e., oracle-free) version of the algorithm. We then define (α, β) -journeys and components, and discuss their trade-offs compared to β -journeys based components. After the presentation of these set of timely behaving temporal components, we describe how Consensus can be solved by using their corresponding TRB implementations. We also consider important to analyze which are the trade-offs between the election of the proposed timely dynamic models in terms of the number of possible graphs that can be achieved with respect to others. We highlight that there exist no implementable Consensus solution for some timely behaving evolving graph. In the same time we also prove that β -components and (α, β) -components can achieve the same set of timely evolving graphs. In farther reading, on the weakest connectivity model, we present a weaker connectivity model called ω -components that allows implementable TRB solutions, and consequently is the weakest timely component based connectivity model solving Consensus.

4.1 A Timely Model for Dynamic Systems

This section focuses on the analysis of timeliness in dynamic systems at the most abstract point of view, i.e., considering only a general communication bound Δ for





4.1 A Timely Model for Dynamic Systems

end-to-end communication. We first provide a set of definitions related to this bound, which leads to the formulation of a new class of TVGs that is a strict subclass of Class 5 (*recurrent connectivity*) in [16]. We then specify a solution to the problem of Terminating Reliable Broadcast (TRB) in the corresponding context.

4.1.1 Definitions

We define the concept of bounded-time journey as follows:

Definition 4.1.1. A journey \mathcal{J} is said to be a Δ -journey if and only if $\text{arrival}(\mathcal{J}) - \text{departure}(\mathcal{J}) \leq \Delta$.

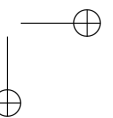
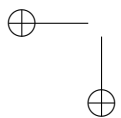
Based on Δ -journeys we define the concept of bounded-time component. Unlike components, we require here that connectivity be also recurrent by definition.

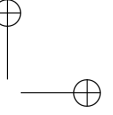
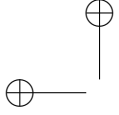
Definition 4.1.2. A Δ -component in $\mathcal{G} = (V, E, \mathcal{T}, \rho, \zeta)$ is a set $V' \subseteq V$ such that for every t in $[\mathcal{T}^-, \mathcal{T}^+ - \Delta]$, for every p, q in V' , there exists a Δ -journey from p to q in $\mathcal{G}_{[t, t+\Delta]}$.

Similarly to components, Δ -components can be open or closed, depending on whether the Δ -journeys use nodes in $V \setminus V'$. Observe that, a graph behaving in an open way provides flexibility in mobility, and therefore, a model allowing open Δ -components is weaker (in the sense that it requires less assumptions) than a model strictly based on closed Δ -components. Henceforth we assume that in our system model Δ -components are open.

Informally, Δ -components allow us to think about subsets of nodes behaving timely with each other. Hence, nodes in a Δ -component are also said to be *timely connected*. We define the (parametrized) class of timely (and recurrently) connected TVGs $\mathcal{TC}(\Delta)$ as follows:

Definition 4.1.3. $\mathcal{G} \in \mathcal{TC}(\Delta) \iff V$ is a Δ -component.





4. CONNECTIVITY MODELS FOR SOLVING AGREEMENT

Observe that $\mathcal{TC}(\Delta)$ does not rely on any kind of slotted synchronization, since \mathcal{T} could perfectly be the set of real values \mathbb{R} . In fact, this is one of the contributions of this model, since trivially, this is the weakest way of introducing time bounds in dynamic systems with no slotted time synchrony assumption.

4.1.2 Terminating Reliable Broadcast in $\mathcal{TC}(\Delta)$

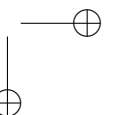
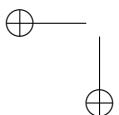
It has been shown in [29] that Consensus is equivalent to Terminating Reliable Broadcast in static synchronous systems. We take this as a starting point and describe here a solution for TRB in the scope of a Δ -component.

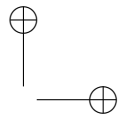
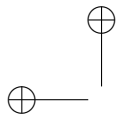
Basic System Model

First of all, processing times are assumed to be negligible with respect to communication time. The system is composed of processes that can crash and recover, and leave and join the system. Processes that crash or leave the system, even if they recover or join again later, are by definition excluded from any Δ -component, however since we assume the existence of open Δ -components, they can punctually take part on journeys.

Recall that a distributed system may have several Δ -components. There may exist values of Δ for which a same process belongs to different components, which are thus overlapping. However, since every component is *recurrently* connected, then overlapping components become naturally merged as the value for Δ increases, and transitively, there must exist a sufficiently large value of Δ such that all remaining components are disjoint. Henceforth, we consider Δ to be (an upper bound on) such a value.

We define now which processes are *correct* in terms of the classical terminology. In a classical partitioned system it can be considered that a process p behaves correctly in its partition, and incorrectly with respect to the other partitions in the system. Similarly, in our Δ -component based system a process p behaves correctly with respect to the Δ -component p belongs to, denoted by C . However p could still sporadically





4.1 A Timely Model for Dynamic Systems

communicate *timely with some* processes in other Δ -component, say C' . We consider p incorrect with respect to C' , but a message m from p received by some process in C' should either be delivered by all processes in C' , or by none of them in order to hold the agreement property of reliable broadcast.

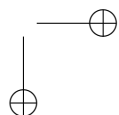
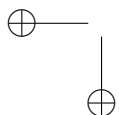
Thus, in $\mathcal{TC}(\Delta)$ a set of properties should be satisfied in order to solve Δ -TRB:

- Δ -Termination: Every process in the same Δ -component eventually delivers some message.
- Δ -Validity: If a process in a Δ -component broadcasts a message m , then all processes in the same Δ -component eventually deliver m .
- Δ -Agreement: If a process in a Δ -component delivers a message m , then all processes in the same Δ -component eventually deliver m .
- Δ -Integrity: For any message m in a Δ -component, every process in the Δ -component delivers at most one message, and if it delivers $m \neq SF$ then sender(m) must have broadcast m .

As usual in TRB, the broadcast at time t_{init} of a message m should be considered in the scope of m .

To guarantee the Δ -Agreement property we should understand when a message m broadcast by $p \notin C$ should be delivered by all processes in C . If p has been able to propagate m to some process $q \in C$, then we assume that there exists a Δ -journey from p to q . Observe that this assumption is consistent with the fact that our model allows the existence of open Δ -components.

A solution to the TRB problem is described in Figure 4.1. Informally, the distinguished process p_B Δ -TRB broadcasts a message m by sending m on all its active edges



4. CONNECTIVITY MODELS FOR SOLVING AGREEMENT

To Δ -TRBroadcast a message m at time t_{init} :

if $p = p_B$ **then**
 for all edge $e = (p_B, -)$ s.t. $\rho(e, t_{init}) = true$ **do**
 $send(m)$ on e at t_{init}

On appearance of $e = (p_B, -)$ at time $t \in [t_{init}, t_{init} + \Delta)$:
 $send(m)$ on e at t

On reception of a message m for the first time at time $t_{rec} \in [t_{init}, t_{init} + 2\Delta)$:

if $p \neq p_B$ **then**
 for all edge $e = (p, -)$ s.t. $\rho(e, t_{rec}) = true$ **do**
 $send(m)$ on e at t_{rec}

On appearance of $e = (p, -)$ at time $t \in [t_{init}, t_{init} + 2\Delta)$:

if a message m has been previously received **then**
 $send(m)$ on e at t

At time $t_{init} + 2\Delta$:

if a message m has been previously received **then**
 Δ -TRDeliver(m)
else
 Δ -TRDeliver(SF)

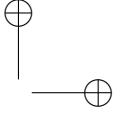
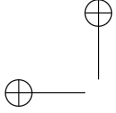
Figure 4.1: Terminating Reliable Broadcast for $\mathcal{TC}(\Delta)$ executed in a node p .

at time t_{init} . Whenever an edge in p_B 's neighborhood appears¹, p_B also sends m on that edge. Every other process p , upon reception of m for the first time, forwards m on all its active edges, as well as upon the appearance of a new edge. Finally, at time $t_{init} + 2\Delta$ every process p Δ -TRBdelivers either m (if m has been received) or SF (*sender faulty* in the classical TRB terminology).

We explain next why a time of 2Δ is necessary and sufficient to deliver m .

Observe that, since we are assuming that p_B could be not in C , p_B could not be

¹We assume here the existence of an abstract *oracle* to capture events of edge appearance. In the next section we will address the implementation of such an oracle.



4.1 A Timely Model for Dynamic Systems

able to communicate to all nodes in C in Δ time, (otherwise $p_B \in C$), thus, after m is resent by q , every process in C will receive m into a second Δ time interval. Henceforth the bound for a process in C to $TRDeliver$ a message is 2Δ .

Correctness proof

We prove that the algorithm specified in Figure 4.1 is a solution to Δ -TRBroadcast in a Δ -component.

Observation 4.1.1. *Observe that, by the system model assumptions, if a message m has been communicated between any two processes p and q then the communication time is bounded by Δ even if p and q are not in the same Δ -component.*

Lemma 4.1.4. *The specification in Figure 4.1 provides the Δ -Termination property.*

Proof. Observe that at time $t_{init} + 2\Delta$ a process p Δ -TRDelivers either m or a SF message. □

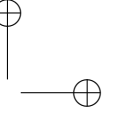
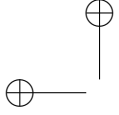
Lemma 4.1.5. *The specification in Figure 4.1 provides the Δ -Validity property.*

Proof. Observe that p_B sends a message m at time t_{init} to all processes with an edge with p at t_{init} and p keeps sending m for every edge whenever it appears during 2Δ time. Since every process $p \in C$ resends m on the appearance of its edges during the same time interval, then m will be received by all processes in C and Δ -TRDelivered at time $t_{init} + 2\Delta$. □

Lemma 4.1.6. *The specification in Figure 4.1 provides the Δ -Agreement property.*

Proof. By Lemma 4.1.5, every process q in a Δ -component C eventually receives and Δ -TRRdelivers m if $p_B \in C$. Else, if $p_B \notin C$, we prove now that either (a) eventually every process $q \in C$ will Δ -TRDeliver m , or (b) no process in C will deliver m .

Assume first that a process $q \in C$ has received m . Since p_B has communicated with q , by Observation 4.1.1, m has been received by q no later than $t_{init} + \Delta$. Since q resends m whenever an edge appears during the next 2Δ time by definition of Δ -component C , and by the proof of Lemma 4.1.5 every process in C receives m before $t_{init} + 2\Delta$ and Δ -TRDelivers m at time $t_{init} + 2\Delta$.



4. CONNECTIVITY MODELS FOR SOLVING AGREEMENT

Otherwise, if no process in C has received m before $t_{init} + \Delta$, again by Observation 4.1.1 m will not be received by any process in C , thus no process in C will deliver m . □

Lemma 4.1.7. *The specification in Figure 4.1 provides the Δ -Integrity property.*

Proof. Observe that a process p Δ -TRDelivers a message just once. Observe also that m and SF are the only messages that can be delivered, been m the message that is Δ -TRBroadcast by process p_B . □

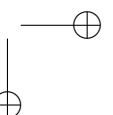
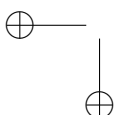
Theorem 4.1.8. *The specification in Figure 4.1 satisfies the properties of Terminating Reliable Broadcast in a Δ -component.*

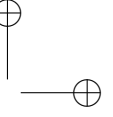
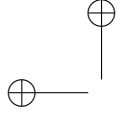
Proof. Straightforward from Lemmas 4.1.4, 4.1.5, 4.1.6 and 4.1.7. □

4.2 Implementability of TRB

The specification of TRB provided in Figure 4.1 relies on an “oracle” available at every process p , which informs p instantaneously upon the appearance of a new edge in its neighborhood. Such an abstraction has been recently used by Raynal et al. [79] to implement a broadcast algorithm for recurrent dynamic systems. However, a strict implementation of this oracle in a real system is far from being trivial, as we discuss now.

Observe that the only temporal assumption in Δ -journeys is that they satisfy a given upper-bound Δ in its temporal length, thus the duration of an edge may be as short as the latency of the message. In consequence, an implementation of this oracle should be able to allow the sending of a message at the very same time that the edge gets activated, which is unrealistic since the oracle should be able to predict the behavior of the links in a real network. Alternatively, an algorithm could continuously send message m along the whole time interval in the hope that one of the sending attempts





4.2 Implementability of TRB

will succeed in the appearance of an edge. Observe, however, that this iteration would require a period of time zero between two consecutive sends. In other words, the algorithm should be able to send an infinite number of messages per unit of time, which is impossible.

Therefore, additional assumptions should be introduced in order to provide an implementation for the above specification of TRB. Specifically, we first propose an extra assumption that allows to maintain active the edge not only for communicating the message but also to detect its appearance.

4.2.1 (Lower)-bounding the Edge Stability

We assume that the edge latency is bounded, i.e., there exists a bound on $\max\{\zeta(e, t) : t \in \mathcal{T}, e \in E\}$, that we call ζ_{MAX} . Additionally, we assume that edges are active at least β time. Let us call β -edge an edge that fulfills this bounded availability. For this new model we define β -journeys as follows:

Definition 4.2.1. A β -journey \mathcal{J} is a Δ -journey such that:

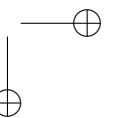
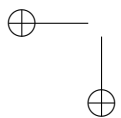
1. $\zeta_{MAX} < \beta \leq \Delta$.
2. $\forall i \in [0, k), e_i$ is a β -edge.
3. the times when edges are activated and their corresponding latencies allow a bounded sequential traversal (formally, $\forall i \in [0, k), t_{i+1} \geq t_i + \beta$).

We now define β -components as a subset of Δ -components that use β -journeys. Formally:

Definition 4.2.2. A β -component is a Δ -component where a set $V' \subseteq V$ satisfies that $\forall t \in [\mathcal{T}^-, \mathcal{T}^+ - \Delta], V'$ is a β -journey based temporal component in $\mathcal{G}_{[t, t+\Delta]}$.

We define the parametrized timely connectivity class $\mathcal{TC}'(\beta)$ as follows:

Definition 4.2.3. $\mathcal{G} \in \mathcal{TC}'(\beta) \iff V$ is a β -component.



4. CONNECTIVITY MODELS FOR SOLVING AGREEMENT

TRB in $\mathcal{TC}'(\beta)$

We give now a TRB algorithm for the $\mathcal{TC}'(\beta)$ model, which is shown in Figure 4.2.

```

1  $W \leftarrow \text{value} \in (0, \beta - \zeta_{MAX}]$ 
2 To  $\Delta$ -TRBroadcast a message  $m$  at time  $t_{init}$ :
3   if  $p = p_B$  then
4     while  $\text{now}() < t_{init} + \Delta$  do
5       send( $m$ ) to all
6       wait( $W$ )
7     end
8   end
9
10 On reception of a message  $m$  for the first time at time  $t_{rec} \in [t_{init}, t_{init} + 2\Delta)$ :
11    $\Delta$ -TRDeliver( $m$ )
12   if  $p \neq p_B$  then
13     while  $\text{now}() < t_{rec} + \Delta$  do
14       send( $m$ ) to all
15       wait( $W$ )
16     end
17   end
18
19 At time  $t_{init} + 2\Delta$ :
20   if  $p$  has not  $\Delta$ -TRDelivered any message then
21      $\Delta$ -TRDeliver( $SF$ )
22   end
23

```

Figure 4.2: Terminating Reliable Broadcast for $\mathcal{TC}'(\beta)$.

In the algorithm proposed in Figure 4.2, process p_B sends at time t_{init} a message m by Δ -TRBroadcasting it, and p_B keeps sending m each W time in order to ensures the correct sending of m through every β -journey. Observe that, according to the definition

4.2 Implementability of TRB

of β -edge, for a β -edge $e = (p, q)$ in a β -journey, if process p sends a message m on e each $W \leq \beta - \zeta_{MAX}$ time during Δ , q will receive m at least once. When a process p receives the message m it Δ -TRDelivers m , and additionally, if $p \neq p_B$, p sends m each W time during Δ . Finally, if a process does not receive the message m , at time $t_{init} + 2\Delta$, it Δ -TRDelivers the special message SF .

Correctness proof

We prove that Algorithm in Figure 4.2 solves Δ -TRBroadcast in a β -component. Thus, note that Δ -TRB properties hold on β -components.

Lemma 4.2.4. *Let $\beta > \zeta_{MAX}$, $0 < W \leq \beta - \zeta_{MAX}$ and let $e = (p, q)$ be a β -edge belonging to a β -journey \mathcal{J} such that $\text{departure}(\mathcal{J}) = t$ where $t \in [\mathcal{T}^-, \mathcal{T}^+ - \Delta]$. If a process p tries to send a message m on e each W from time \mathcal{T}^- to time $\mathcal{T}^+ - \Delta$, q will receive m at least once.*

Proof. Since \mathcal{J} is a β -journey, by definition \mathcal{J} is also a Δ -journey and thus its temporal length is bounded by Δ . Also by definition of β -journey, e should appear at least once and be active for at least β time. Let t' be the time when the edge e appears, thus e is active in the interval $t' + \beta$. Observe that $t \leq t' \leq t + \Delta - \beta$.

We prove now that if p is sending m on e at times \mathcal{T}^- , $\mathcal{T}^- + W$, $\mathcal{T}^- + 2W$, ..., m will be received by q no later than $t + \Delta$.

Consider the worst-case situation, in which: (a) e becomes active only once in the interval (recall that $t' \leq t + \Delta - \beta$), (b) e has activated just after a sending attempt at time $\mathcal{T}^- + kW$, and (c) the latency of the sending attempt at time $\mathcal{T}^- + (k+1)W$ is the maximum latency we are assuming, ζ_{MAX} .

In this situation e is active in the interval $(\mathcal{T}^- + kW, \mathcal{T}^- + kW + \beta]$. Process p will try to send m at time $\mathcal{T}^- + (k+1)W$, thus, in order to be a successful attempt, e should be active in the interval $[\mathcal{T}^- + (k+1)W$ to $\mathcal{T}^- + (k+1)W + \zeta_{MAX}]$. We prove then that $[\mathcal{T}^- + (k+1)W, \mathcal{T}^- + (k+1)W + \zeta_{MAX}] \subset (\mathcal{T}^- + kW, \mathcal{T}^- + kW + \beta]$

Observe first that at the time of the new sending of m by p , e continues to be active, since $W < \beta$. Observe now that the bound for m to be received by q is not higher than the time at which e disappears, since by definition $W \leq \beta - \zeta_{MAX}$. In effect,

$$\mathcal{T}^- + (k+1)W + \zeta_{MAX} \leq \mathcal{T}^- + kW + \beta$$

4. CONNECTIVITY MODELS FOR SOLVING AGREEMENT

which results in

$$\zeta_{MAX} \leq \beta - W$$

Finally we show that m is received by q before $t + \Delta$.

In the limit, the only activation of e could happen at a time $t' \leq t + \Delta - \beta$. Thus, $\mathcal{T}^- + kW < \mathcal{T}^- + \Delta - \beta$. Since the last sending attempt of p to q could be done as late as at time $\mathcal{T}^- + (k+1)W$, m would be received by p before $\mathcal{T}^- + \Delta - \beta + W + \zeta_{MAX}$. Again, since $W \leq \beta - \zeta_{MAX}$, the previous results in that m is received by q before $t + \Delta$. \square

Lemma 4.2.5. *Algorithm in Figure 4.2 provides the Δ -Termination property: Every process in the same β -component eventually delivers some message.*

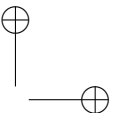
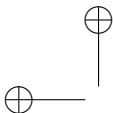
Proof. Observe that by Lines 20-21 a process p executing the algorithm in Figure 4.2 Δ -TRDelivers a SF message at time $t_{init} + 2\Delta$ if p has not previously Δ -TRDeliver m by Line 11. \square

Lemma 4.2.6. *Algorithm in Figure 4.2 provides the Δ -Validity property: If a process in a β -component broadcasts a message m , then all processes in the same β -component eventually deliver m .*

Proof. Observe first that, since $W \in (0, \beta - \zeta_{MAX}]$ by Line 1, Lemma 4.2.4 is applicable. By Lemma 4.2.4 and the Definition 4.2.2, if a process p_B in a β -component C sends a message m to all processes at time t_{init} and p_B keeps sending m periodically with a period $W < \beta - \zeta_{MAX}$ (lines 4-7), then m will be received by Line 11 at least by one process in C , otherwise p_B is the only process in C . A process $q \in C$ receiving m by Line 11 will Δ -TRDeliver m , and will resend m by lines 13-16 of the algorithm. Reasoning as previously by iteration on Lemma 4.2.4 and the Definition 4.2.2, every process in C will Δ -TRDeliver m before $t_{init} + \Delta$. \square

Lemma 4.2.7. *Algorithm in Figure 4.2 provides the Δ -Agreement property: If a process in a β -component delivers a message m , then all processes in the same β -component eventually deliver m .*

Proof. By Lemma 4.2.6, every process q in a β -component C eventually receives and Δ -TRDelivers m if $p_B \in C$. Else, if $p_B \notin C$, we prove now that either (a) eventually every process $q \in C$ will Δ -TRDeliver m , or (b) no process in C will deliver m .



4.2 Implementability of TRB

Assume first that a process $q \in C$ has received m (by Line 10). Before resending m by lines 13-17, q will Δ -TRDeliver m (by Line 11), thus we should prove now that every process in C will Δ -TRDeliver m . Since p_B has communicated with q , by Observation 4.1.1 m has been received by q no later than $t_{init} + \Delta$. Since q resends m by lines 13-17, by definition of β -component C , and by the proof of Lemma 4.2.6 every process in C eventually receives and Δ -TRDelivers m .

Otherwise, if no process in C has received m before $t_{init} + \Delta$, again by Observation 4.1.1, m will not be received by any process in C , thus no process in C will deliver m . \square

Lemma 4.2.8. *Algorithm in Figure 4.2 provides the Δ -Integrity property: For any message m present in a β -component, every process in the β -component delivers at most one message, and if it delivers $m \neq SF$ then the sender(m) must have broadcast m .*

Proof. A process p executes the Δ -TRDeliver primitive (after receiving m , Line 11) just once since the Line 10 explicitly denotes “for the first time”, or by SF by Line 21 at time $t_{init} + \Delta$. Observe that, by Line 20, Δ -TRDeliver(SF) is only executed if p has not previously delivered m by Line 11, thus either one message m or SF will be delivered.

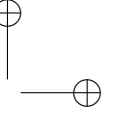
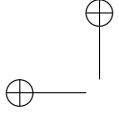
Observe also by the algorithm that m and SF are the only messages that can be delivered, been m the message that is Δ -TRBroadcast by process p_B . \square

Theorem 4.2.9. *The algorithm in Figure 4.2 satisfies the properties of Δ -TRB in a β -component.*

Proof. Straightforward from Lemmas 4.2.5, 4.2.6, 4.2.7 and 4.2.8. \square

4.2.2 (Upper)-bounding the Edge Appearance

In previous reading we have presented a new timely connectivity model stronger than $\mathcal{TC}(\Delta)$. Based on the assumption of an edge activity higher than the maximum latency of the system, the $\mathcal{TC}'(\beta)$ connectivity offers us the possibility to implement TRB without using any kind of oracle. Nevertheless, observe that in the algorithm in Figure 4.2 messages are forwarded during the whole Δ interval. This is necessary because



4. CONNECTIVITY MODELS FOR SOLVING AGREEMENT

the ending edge of a β -journey could be activated at a time as late as $t_{init} + \Delta - \beta$. It is apparent that more efficient implementations of a TRB algorithm in terms of number of messages could be envisaged if stronger connectivity assumptions are introduced in the model. Specifically, in this section we introduce an additional timely assumption on the appearance of edges.

We adopt the assumption of [5], where, besides β , a bound α on the appearance of links is defined. We define a new type of journey, that we call (α, β) -journey. Formally:

Definition 4.2.10. A (α, β) -journey \mathcal{J} is a β -journey such that:

1. The appearance of e_1 is bounded by $t_1 \leq t + \alpha$.
2. The appearance of the subsequent edges are also bounded by α . Formally, $t_{i+1} \leq t_i + \zeta(e_i, t_i) + \alpha$ for all $i \in [1, k)$.

We define a (α, β) -component as follows:

Definition 4.2.11. A (α, β) -component is a β -component where a set $V' \subseteq V$ satisfies that $\forall t \in [\mathcal{T}^-, \mathcal{T}^+ - \Delta], V'$ is a (α, β) -journey based temporal component in $\mathcal{G}_{[t, t+\Delta)}$.

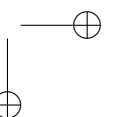
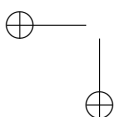
We define the parametrized timely connectivity class $\mathcal{TC}''(\alpha, \beta)$ as follows:

Definition 4.2.12. $\mathcal{G} \in \mathcal{TC}''(\alpha, \beta) \iff V$ is a (α, β) -component.

TRB in $\mathcal{TC}''(\alpha, \beta)$

The algorithm in Figure 4.3, present a TRB algorithm in a $\mathcal{TC}''(\alpha, \beta)$ dynamic system.

The new bound α , together with the latency bound β and ζ_{MAX} , allows to calculate global system bounds, namely the period W and a time to deliver Γ , strictly in terms of specific network parameters. In the algorithm proposed in Figure 4.3 process p_B Δ -TRBROADCAST a message m at time t_{init} by sending each W time m until the time is strictly higher than $t_{init} + \alpha$, in order to ensure the correct sending of m by every



4.2 Implementability of TRB

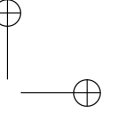
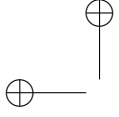
```

1  $W \leftarrow \text{value} \in (0, \beta - \zeta_{MAX}]$ 
2  $\Gamma \leftarrow (\lceil \frac{\alpha}{W} \rceil + (|V| - 2) \lceil \frac{\zeta_{MAX} + \alpha}{W} \rceil)W + \zeta_{MAX}$ 
3 To  $\Delta$ -TRBroadcast a message  $m$  at time  $t_{init}$ :
4   if  $p = p_B$  then
5     send( $m$ ) to all
6   repeat
7     wait( $W$ )
8     send( $m$ ) to all
9   until  $\text{now}() > t_{init} + \alpha$ 
10  end
11
12 On reception of a message  $m$  for the first time at time  $t_{rec}$ :
13   $\Delta$ -TRDeliver( $m$ )
14  if  $p \neq p_B$  then
15    send( $m$ ) to all
16  repeat
17    wait( $W$ )
18    send( $m$ ) to all
19  until  $\text{now}() > t_{rec} + \alpha$ 
20  end
21
22 At time  $t_{init} + \Gamma$ :
23  if  $p$  has not  $\Delta$ -TRDelivered any message then
24     $\Delta$ -TRDeliver( $SF$ )
25  end
26

```

Figure 4.3: Terminating Reliable Broadcast for $\mathcal{TC}''(\alpha, \beta)$.

(α, β) -journey. When a process p receives the message m at time t_{rec} for the first time, it Δ -TRDelivers m and, additionally, if $p \neq p_B$, p sends m each W until the time is strictly higher than $t_{rec} + \alpha$. Finally, if any of the processes in C does not receive the



4. CONNECTIVITY MODELS FOR SOLVING AGREEMENT

message m at time $t_{init} + \Gamma$, Δ -TRDelivers the special message SF denoting the sender failure.

It is important to note that in the TRB algorithm for $\mathcal{TC}''(\alpha, \beta)$, differently to the upper classes, processes need to know the network diameter, which is bounded by $|V| - 1$. This is a consequence of the fact of considering strictly local bounds in $\mathcal{TC}''(\alpha, \beta)$. Instead, both $\mathcal{TC}(\Delta)$ and $\mathcal{TC}'(\beta)$ rely on a system-wide bound, Δ .

Bounding the Time-to-Deliver

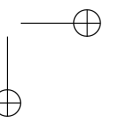
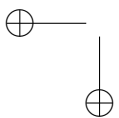
We explain now how we calculate Γ , the bound used in the algorithm in Figure 4.3 for a process to Δ -TRBdeliver the message (see Figure 4.4 for a graphical illustration).

A process p_1 (the sender) will send a copy of m from t_{init} on, each W time units. In the worst case, the first edge of the journey will appear at $t_{init} + \alpha$, but p_1 will not success sending a copy of m until $t_{init} + \lceil \frac{\alpha}{W} \rceil W$, i.e., the $\lceil \frac{\alpha}{W} \rceil W$'s sending attempt. Observe that, $t_{init} + \alpha < t_{init} + \lceil \frac{\alpha}{W} \rceil W \leq t_{init} + \alpha + W$.

For a journey including a single edge (p_1, p_2) , the message m would be delivered by p_2 at time $t_{init} + \lceil \frac{\alpha}{W} \rceil W + \zeta_{MAX}$.

In general, for a journey including k nodes (and thus $k - 1$ hops), excluding the first hop, the subsequent $k - 2$ hops can be time-bounded as follows: a message m resent by a process p_i is received by p_{i+1} in ζ_{MAX} time units and p_{i+1} waits α time units until the appearance of edge e_{i+1} to re-send m on this edge. Consequently, p_{i+1} will succeed in the sending attempt made on e_{i+1} at a time not greater than $t_{init} + \lceil \frac{\alpha}{W} \rceil W + \lceil \frac{\zeta_{MAX} + \alpha}{W} \rceil W$. Summarizing, a message from p_1 to p_k by a (α, β) -journey at time t_{init} will be delivered by p_k at time $t_{init} + \Gamma$, where $\Gamma = (\lceil \frac{\alpha}{W} \rceil + (k - 2) \lceil \frac{\zeta_{MAX} + \alpha}{W} \rceil)W + \zeta_{MAX}$.

In the worst case, $k = |V| - 1$, the bound to deliver a message will be $t_{init} + \Gamma$, where $\Gamma = (\lceil \frac{\alpha}{W} \rceil + (|V| - 2) \lceil \frac{\zeta_{MAX} + \alpha}{W} \rceil)W + \zeta_{MAX}$.



4.2 Implementability of TRB

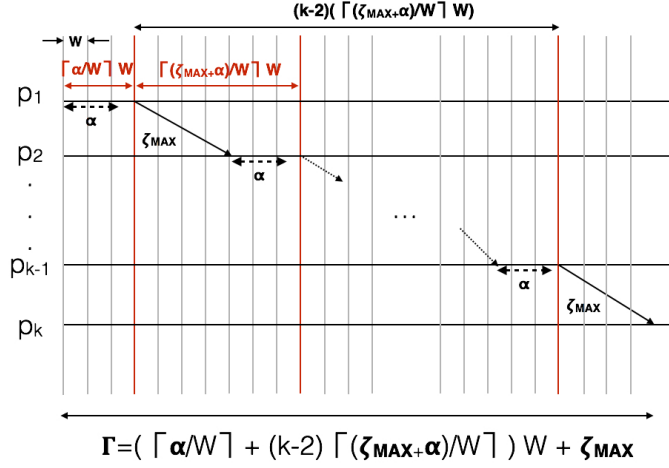


Figure 4.4: A time-line explaining the Γ upper-bound for the worst case (α, β) -journey from a process p to another process q in the system.

Correctness proof

Lemma 4.2.13. *Algorithm in Figure 4.3 provides the Δ -Termination property: Every process in the same (α, β) -component eventually delivers some message.*

Proof. Observe that by Lines 22-25 a process p executing the algorithm in Figure 4.3 Δ -TRDelivers a SF message at time $t_{init} + \Gamma$ if p has not previously Δ -TRDelivered m by Line 13. \square

Lemma 4.2.14. *Algorithm in Figure 4.3 provides the Δ -Validity property: If a process in a (α, β) -component broadcasts a message m , then all processes in the same (α, β) -component eventually deliver m .*

Proof. By Definition 4.2.11, if a process p_B in a (α, β) -component C sends a message m to all processes at time t_{init} and p_B keeps sending m periodically with a period $W < \beta - \zeta_{MAX}$ (Lines 6-9) during the maximum time for the appearance of the link, α , then m will be received by Line 12 at least by one process in C , otherwise p_B is the unique process in C . Reasoning in the same way by iteration on Lines 16-19 of the

4. CONNECTIVITY MODELS FOR SOLVING AGREEMENT

algorithm, by Definition 4.2.11, every process $q \in C$ will Δ -TRDeliver m by Line 13 before $t_{init} + \Gamma$. Observe by Line 2 that Γ has been set as a bound of the temporal length of an (α, β) -journey in a system with $|V|$ nodes. □

Lemma 4.2.15. *Algorithm in Figure 4.3 provides the Δ -Agreement property: If a process in a (α, β) -component delivers a message m , then all processes in the same (α, β) -component eventually deliver m .*

Proof. By Lemma 4.2.14, every process q in a (α, β) -component C eventually receives and Δ TRDelivers m if $p_B \in C$. Else, if $p_B \notin C$, we prove now that either (a) eventually every process $q \in C$ will Δ TRDeliver m , or (b) no process in C will deliver m .

Assume first that a process $q \in C$ has received m (by Line 12). Since p_B has communicated with q , m has been received by Line 12 of q according the time bounds α and β following a journey topologically bounded by the maximum network diameter, $|V - 1|$, and q Δ -TRDelivers m by Line 13. By Line 2, Γ has been set as a bound on the temporal length of such an (α, β) -journey. By Lines 16-19 m is resent by q during the maximum time for the appearance of the link, α , and, again by Lemma 4.2.14, every process in a (α, β) -component C eventually receives and Δ TRDelivers m . Again, Γ holds as the general bound, since it considers the worst-case diameter, which includes all the processes in the system.

Otherwise, if no process in C has received m before $t_{init} + \Gamma$, every process in C will Δ -TRDeliver SF at time $t_{init} + \Gamma$. □

Lemma 4.2.16. *Algorithm in Figure 4.3 provides the Δ -Integrity property: For any message m present in a (α, β) -component, every process in the (α, β) -component delivers at most one message, and if it delivers $m \neq SF$ then the sender(m) must have broadcast m .*

Proof. A process p executes the Δ -TRDeliver primitive (after receiving m , Line 13) just once since the Line 12 explicitly denotes “for the first time”, or by SF by Line 24 at time $t_{init} + \Gamma$. Observe that, by Line 24, Δ -TRDeliver(SF) is only executed if p has not previously delivered a message m by Line 13.

Observe also by the algorithm that m and SF are the only messages that can be delivered, been m the message that is Δ -TRBroadcast by process p_B .

□

Theorem 4.2.17. *The algorithm in Figure 4.3 satisfies the properties of Terminating Reliable Broadcast in a (α, β) -component.*

Proof. Straightforward from Lemmas 4.2.13, 4.2.14, 4.2.15 and 4.2.16. □

4.2.3 Relating Timely Classes

We have defined a hierarchy of classes with increasingly stronger timely assumptions. Being $\mathcal{TC}(\Delta)$, $\mathcal{TC}'(\beta)$ and $\mathcal{TC}''(\alpha, \beta)$ the *parametrized* classes, we define now for each one the union of all its possible instances:

$$\mathcal{G} \in \mathcal{TC}^* \iff \exists \Delta \neq \infty : \mathcal{G} \in \mathcal{TC}(\Delta)$$

$$\mathcal{G} \in \mathcal{TC}^* \iff \exists \beta \neq \infty : \mathcal{G} \in \mathcal{TC}'(\beta)$$

$$\mathcal{G} \in \mathcal{TC}^{**} \iff \exists \alpha, \beta \neq \infty : \mathcal{G} \in \mathcal{TC}''(\alpha, \beta)$$

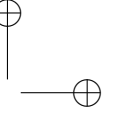
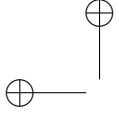
In spite of the different strength of the parametrized classes, we show that $\mathcal{TC}^{**} \equiv \mathcal{TC}^*$. Besides, $\mathcal{TC}'^* \subset \mathcal{TC}^*$ and $\mathcal{TC}^{**} \subset \mathcal{TC}^*$.

Correctness proof

Theorem 4.2.18. $\mathcal{TC}^{**} \equiv \mathcal{TC}'^*$

Proof. On the one hand, $\forall \mathcal{G} \in \mathcal{TC}^{**}, \exists \beta : \mathcal{G} \in \mathcal{TC}'(\beta)$, since by definition, a (α, β) -component in $\mathcal{TC}'(\beta)$ is a β -component. More specifically, if $\mathcal{G} \in \mathcal{TC}''(\alpha, \beta)$ then $\mathcal{G} \in \mathcal{TC}'((|V| - 1)(\alpha + \beta), \beta)$. On the other hand, $\forall \mathcal{G} \in \mathcal{TC}'^*, \exists \alpha, \beta : \mathcal{G} \in \mathcal{TC}''(\alpha, \beta)$. Observe that $\mathcal{G} \in \mathcal{TC}''(\alpha', \beta)$ such that α' is $\max(t - t') : t \in [\mathcal{T}^-, \mathcal{T}^+], t' = t_0$ where $\forall \mathcal{J} = \{(e_0, t_0), (e_1, t_1), \dots\} \in \mathcal{J}^* \wedge \text{arrival}(\mathcal{J}) < t + \Delta$ and $\mathcal{G} \in \mathcal{TC}'(\beta)$. Since, $\mathcal{TC}'^* \setminus \mathcal{TC}^{**} = \emptyset$, then $\mathcal{TC}^{**} \equiv \mathcal{TC}'^*$. □

Theorem 4.2.19. $\mathcal{TC}'^* \subset \mathcal{TC}^*$



4. CONNECTIVITY MODELS FOR SOLVING AGREEMENT

Proof. By the definition of $\mathcal{TC}'(\beta)$, $\forall G \in \mathcal{TC}'^*$, $\exists \Delta : G \in \mathcal{TC}(\Delta)$. We prove now that there exists $\mathcal{G} \in \mathcal{TC}^*$ such that $G \notin \mathcal{TC}'^*$. Assume a graph \mathcal{G} such that any of its journey is composed by an edge e_i which is active during a β_i time where $\beta_i = \zeta_{MAX}$, and, by Definition 4.2.1, those journeys are not allowed in any parametrized $\mathcal{TC}'(\beta)$ class. Consequently, $\mathcal{TC}'^* \subset \mathcal{TC}^*$. \square

Theorem 4.2.20. $\mathcal{TC}''^* \subset \mathcal{TC}^*$

Proof. By Theorem 4.2.18 and Theorem 4.2.19, $\mathcal{TC}''^* \subset \mathcal{TC}^*$. \square

4.3 From Δ -TRB to Δ -Consensus in Dynamic Systems

In this section we analyse the equivalence between TRB and Consensus, originally stated for synchronous static systems [29], in terms of a dynamic system as the one we have modelled.

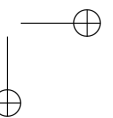
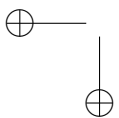
In the former sections we have presented three Δ -TRB algorithms in the scope of respectively Δ -, β - and (α, β) -components. We show now how the Consensus problem can be reduced¹ to a Δ -TRB problem. We will refer as Δ -Consensus to this kind of Consensus in the scope of Δ -components.

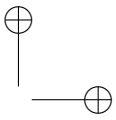
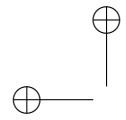
By the properties of Δ -TRB, it is straightforward to define the Δ -Consensus properties as follows:

- *Δ -Termination:* Every process in the Δ -component eventually decides.
- *Δ -Agreement:* Every process in the Δ -component decides the same value.
- *Δ -Validity:* The decided value is a proposed one.

Without losing generality we focus here on Δ -Consensus using the Δ -TRB specification of Figure 4.1 for the $\mathcal{TC}(\Delta)$ Class.

¹We say that a problem A can be reduced to a problem B if A can be solved using B.





4.4 On the Weakest Implementable Timely Connectivity Class

```
1 Vector  $V_p(i) \leftarrow \perp : i \in [0, |V|)$ 
2 To  $\Delta$ -Propose  $v$  at time  $t_{init}$ :
3    $\Delta$ -TRBroadcast( $v$ )
4
5 On  $\Delta$ -TRDeliver( $m$ ) by  $q$ :
6    $V_p(q) \leftarrow m$ 
7
8 At time  $t_{init} + 2\Delta$ :
9    $\Delta$ -decide( $V_p(\min(i : V_p(i) \neq SF)$ ))
10
```

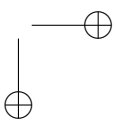
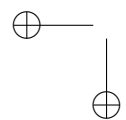
Figure 4.5: Δ -TRB based Δ -Consensus algorithm for $\mathcal{TC}(\Delta)$.

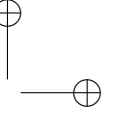
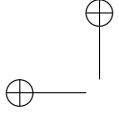
The resulting Δ -Consensus algorithm is shown in Figure 4.5. Every process p holds a vector V_p initialized to \perp . At time t_{init} , $|V|$ instances of Δ -TRB are started, one per process, being each process the sender in one instance. Every process p records in vector $V_p(q)$ the message m_q delivered from process q (or SF in case m_q has not been received on time). At time $t_{init} + 2\Delta$, p decides on the first non- SF value of V_p .

Note that solving Consensus at system level would require an additional Δ -TRB instance to agree on the decision of the majority, provided that the temporal interval $[\mathcal{T}^-, \mathcal{T}^+]$ in which the Δ -component is defined covers both rounds. In other words, the stability of Δ -components must be temporally extended to solve Consensus at system level.

4.4 On the Weakest Implementable Timely Connectivity Class

In the previous Section, we have introduced three different connectivity classes by which Consensus can be solved. We have also identified that in order to be implement-





4. CONNECTIVITY MODELS FOR SOLVING AGREEMENT

able a dynamic distributed agreement protocol, two essential assumptions are required. On the one hand it is necessary a time bound in communications. On the other hand, the link active time must be strictly higher than the time required for a message to be correctly delivered.

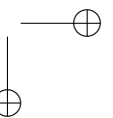
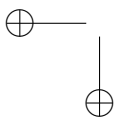
The class $\mathcal{TC}'(\beta)$, although it is a weak option, it is based on β -journeys, which are not the weakest journeys allowing implementable timely communication primitives. We are interested in determining which is the weakest connectivity class providing implementable timely communication primitives, and consequently which is the weakest connectivity class allowing to solve Consensus without using any kind of oracle.

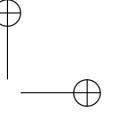
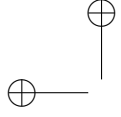
Note that by Definition 4.2.2, every edge belonging to a journey of a β -component share the same β value. This means that exists a common lower-bound satisfied by every journey of the system which is $\beta - \zeta_{MAX}$. Observe that Lemma 4.2.4 the assumption of a non-zero drift between β and ζ_{MAX} allows the existence of an implementable communication protocol.

Observation 4.4.1. *Trivially, the Lemma 4.2.4 also illustrates that it is necessary for providing implementable communication protocols in Δ -components since a $W = 0$ value corresponds to send each 0 time a message, or in other words, send infinite messages.*

The target communication protocol states that if processes of a β -journey send a message m each W where $0 < W \leq \beta - \zeta_{MAX}$, then m will be delivered by each node involved in the journey.

Every edge involved in a β -journey is assumed to be active β time, i.e., more than the maximum latency of the system. However, an edge e_i belonging to a β -journey delivers messages at most $\zeta(e_i, t_i + \beta - \zeta_{MAX})$, which can be a latency lower than the ζ_{MAX} itself. In summary, in β -journeys edges are assumed to be active more time than the required to guarantee implementable communications.





4.4 On the Weakest Implementable Timely Connectivity Class

In the following we present the weakest connectivity class providing implementable timely communication primitives.

In the same way that β is the activation time required for every edge in the system, we consider that β_i is the specific activation time of an edge e_i at time t_i . We call ω -edge to the edge that satisfies that $\beta_i - \zeta(e_i, t_i) \geq \omega$. We also consider that $\forall t \in [t_i, t_i + \omega)$, $\zeta(e_i, t) \leq \zeta(e_i, t_i)$. Observe that, in the worst case communication $\zeta(e_i, t_i) + \omega = \Delta$, which in essence could be considered as the previous ζ_{MAX} .

We now define ω -journeys as follows:

Definition 4.4.1. A ω -journey \mathcal{J} is a Δ -journey such that:

1. $\forall i \in [0, k), e_i$ is a ω -edge.
2. $\zeta(e_i, t_i) < \beta_i \leq \Delta$.
3. the times when edges are activated and their corresponding latencies allow a bounded sequential traversal (formally, $\forall i \in [0, k), t_{i+1} \geq t_i + \beta_i$).

We now define ω -components as a subset of Δ -components that uses ω -journeys. Formally:

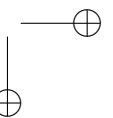
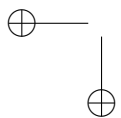
Definition 4.4.2. A ω -component is a Δ -component where a set $V' \subseteq V$ satisfies that $\forall t \in [\mathcal{T}^-, \mathcal{T}^+ - \Delta], V'$ is a ω -journey based temporal component in $\mathcal{G}_{[t, t+\Delta)}$.

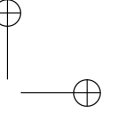
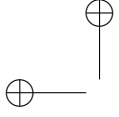
We define the parametrized timely connectivity class $\mathcal{TC}'(\omega)$ as follows:

Definition 4.4.3. $\mathcal{G} \in \mathcal{TC}'(\omega) \iff V$ is a ω -component.

Observation 4.4.2. Observe that, by Definition 4.4.2 every process is connected by ω -journeys, i.e., with the same ω value. In essence, exists a common lower-bound in the drift $\beta_i - \zeta(e_i, t_i)$ satisfied by every edge of the component. Since the W period of the Lemma 4.2.4 also relies on a common global drift $\beta - \zeta_{MAX}$, Lemma 4.2.4 also holds for ω -journey.

Theorem 4.4.4. $\mathcal{TC}'^* \subset \mathcal{TC}^w$





4. CONNECTIVITY MODELS FOR SOLVING AGREEMENT

Proof. By the definition of $\mathcal{TC}'(\beta)$, $\forall G \in \mathcal{TC}'^*$, $\exists \omega : G \in \mathcal{TC}^w(\omega)$, more precisely, $\omega = \beta - \zeta_{MAX}$. We prove now that there exists $\mathcal{G} \in \mathcal{TC}^w$ such that $G \notin \mathcal{TC}'^*$.

In a β -journey \mathcal{J} the lower-bound β is the time required for every edge e_i to be active, i.e., $\forall i, \beta_i \geq \beta > \zeta_{MAX}$ which is bounded according the maximum ζ_{MAX} in the system. Thus, every link must be active a time higher than ζ_{MAX} . Besides, in ω -journeys, even having an $\omega = \beta - \zeta_{MAX}$, by Definition 4.4.1 each e_i is allowed to be a $\beta_i = \zeta(e_i, t_i) + \omega$. Every edge of an ω -journey can be active different times while the drift $\beta_i - \zeta(e_i, t_i) \geq \omega$ holds. However, note that β_i is not restricted to be higher than ζ_{MAX} . A possibility to represent ω -journeys in terms of β -journeys would be assuming that $\beta = \omega + \min\{\zeta(e_i, t_i)\}$. Unfortunately this statement would result in $\zeta_{MAX} = \min\{\zeta(e_i, t_i)\}$ which is a contradiction.

Therefore, there are ω -journeys than cannot be represented as any possible β -journeys. Consequently, $\mathcal{TC}'^* \subset \mathcal{TC}^w$. \square

Theorem 4.4.5. \mathcal{TC}^w is the weakest connectivity class based on Δ -components able to provide timely without assuming an oracle.

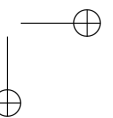
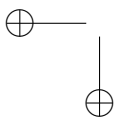
Proof. The Observation 4.4.1 shows that is necessary the fulfillment of this property. Note that, by Observation 4.4.2, Lemma 4.2.4 holds for ω -journeys. Observe also that ω -edges drift are based exclusively on their own β_i and $\zeta(e_i, t_i)$ parameters, but not in other existing edge's behavior. This fact is ratified by Theorem 4.4.4, which shows that \mathcal{TC}^w contains \mathcal{TC}'^* .

Consequently, the $\mathcal{TC}^w(\omega)$ is the weakest connectivity class based on Δ -components able to provide timely communications by implementable communication primitives. \square

TRB in $\mathcal{TC}^w(\omega)$

We give now a TRB algorithm for the $\mathcal{TC}^w(\omega)$ model, which is shown in Figure 4.6.

In the algorithm proposed in Figure 4.6 process p_B sends at time t_{init} a message m by Δ -TRBroadcasting it, and p_B keeps sending m each W time in order to guarantee the correct sending of m by every ω -journey. Observe that, according to the definition of ω -edge, for a ω -edge $e = (p, q)$ in a ω -journey, if process p sends a message m on



4.4 On the Weakest Implementable Timely Connectivity Class

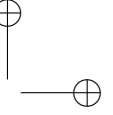
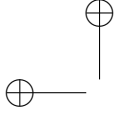
```

1  $W \leftarrow \text{value} \in (0, \omega]$ 
2 To  $\Delta$ -TRBroadcast a message  $m$  at time  $t_{init}$ :
3   if  $p = p_B$  then
4     while  $\text{now}() < t_{init} + \Delta$  do
5       send( $m$ ) to all
6       wait( $W$ )
7     end
8   end
9
10 On reception of a message  $m$  for the first time at time  $t_{rec} \in [t_{init}, t_{init} + 2\Delta)$ :
11    $\Delta$ -TRDeliver( $m$ )
12   if  $p \neq p_B$  then
13     while  $\text{now}() < t_{rec} + \Delta$  do
14       send( $m$ ) to all
15       wait( $W$ )
16     end
17   end
18
19 At time  $t_{init} + 2\Delta$ :
20   if  $p$  has not  $\Delta$ -TRDelivered any message then
21      $\Delta$ -TRDeliver( $SF$ )
22   end
23

```

Figure 4.6: Terminating Reliable Broadcast for $\mathcal{TC}^w(\omega)$.

e each $W \leq \omega$ time during Δ , q will receive m at least once. When a process p receives the message m it Δ -TRDelivers m , and additionally, if $p \neq p_B$, p sends m each W time during Δ . Finally, if a process does not receive the message m , at time $t_{init} + 2\Delta$, it Δ -TRDelivers the special message SF .



4. CONNECTIVITY MODELS FOR SOLVING AGREEMENT

Correctness proof

Similar to previous proofs, we prove that Algorithm in Figure 4.6 solves Δ -TRBroadcast in a ω -component. Thus, note that Δ -TRB properties hold on ω -components.

Lemma 4.4.6. *Algorithm in Figure 4.6 provides the Δ -Termination property: Every process in the same ω -component eventually delivers some message.*

Proof. Observe that by Lines 19-22 a process p executing the algorithm in Figure 4.6 Δ -TRDelivers a SF message at time $t_{init} + 2\Delta$ if p has not previously Δ -TRDeliver m by Line 11. \square

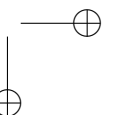
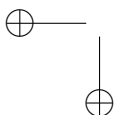
Lemma 4.4.7. *Algorithm in Figure 4.6 provides the Δ -Validity property: If a process in a ω -component broadcasts a message m , then all processes in the same ω -component eventually deliver m .*

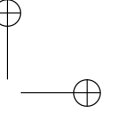
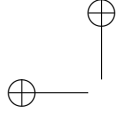
Proof. Observe first that, since $W \in (0, \omega]$ by Line 1 and thus Lemma 4.2.4 is applicable. By Observation 4.4.2, Lemma 4.2.4 and the definition of ω -component, if a process p_B in a ω -component C sends a message m to all processes at time t_{init} and p_B keeps sending m periodically with a period $W < \omega$ (lines 4-7), then m will be received by Line 11 at least by one process in C , otherwise p_B is the only process in C . A process $q \in C$ receiving m by Lines 10-11 will Δ -TRDeliver m , and will resend m by lines 13-16 of the algorithm. Reasoning as previously by iteration on Lemma 4.2.4 and the definition of ω -component, every process in C will Δ -TRDeliver m before $t_{init} + \Delta$. \square

Lemma 4.4.8. *Algorithm in Figure 4.6 provides the Δ -Agreement property: If a process in a ω -component delivers a message m , then all processes in the same ω -component eventually deliver m .*

Proof. By Lemma 4.4.7, every process q in a ω -component C eventually receives and Δ -TRDelivers m if $p_B \in C$. Else, if $p_B \notin C$, we prove now that either (a) eventually every process $q \in C$ will Δ -TRDeliver m , or (b) no process in C will deliver m .

Assume first that a process $q \in C$ has received m (by Line 10). Before resending m by lines 13-16, q will Δ -TRDeliver m (by Line 11), thus we should prove now that every process in C will Δ -TRDeliver m . Since p_B has communicated with q , by Observation 4.1.1 m has been received by q not later than $t_{init} + \Delta$. Since q resends





4.5 Conclusions

m by lines 13-16, by definition of ω -component C , and by the proof of Lemma 4.4.7 every process in C eventually receives and Δ -TRDelivers m .

Otherwise, if no process in C has received m before $t_{init} + \Delta$, again by Observation 4.1.1 m will not be received by any process in C , thus no process in C will deliver m . \square

Lemma 4.4.9. *Algorithm in Figure 4.6 provides the Δ -Integrity property: For any message m present in a ω -component, every process in the same ω -component delivers at most one message, and if it delivers $m \neq SF$ then the sender(m) must have broadcast m .*

Proof. A process p executes the Δ -TRDeliver primitive (after receiving m , Line 11) just once since the Line 10 explicitly denotes “for the first time”, or by SF by Line 21 at time $t_{init} + \Delta$. Observe that, by Line 20, Δ -TRDeliver(SF) is only executed if p has not previously delivered m by Line 11, thus either one message m or SF will be delivered.

Observe also by the Algorithm that m and SF are the only messages that can be delivered, been m the message that is Δ -TRBroadcast by process p_B . \square

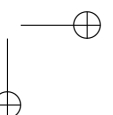
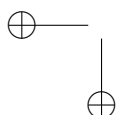
Theorem 4.4.10. *The algorithm in Figure 4.6 satisfies the properties of Δ -TRB in a ω -component.*

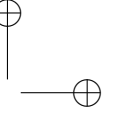
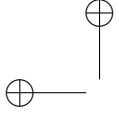
Proof. Straightforward from Lemmas 4.4.6, 4.4.7, 4.4.8 and 4.4.9. \square

4.5 Conclusions

In this chapter we have studied how to introduce timeliness in evolving systems so that the resolution of agreement problems (specifically Consensus) is possible. On the basis of previous works, we have adopted the concept of journey or temporal path and have introduced the necessary timeliness (i.e., time bounds) to describe the specific assumptions that are required by an agreement algorithm to terminate and satisfy the Consensus properties.

We have first proposed a general class, $\mathcal{TC}(\Delta)$, with a very abstract property on the temporal connectivity of the TVG to provide the necessary stability condition, namely,





4. CONNECTIVITY MODELS FOR SOLVING AGREEMENT

that the temporal diameter of a recurrent component in the TVG is bounded. We refer to such a component as a Δ -component. To approach the Consensus problem we have defined a TRB specification in terms of Δ -components, Δ -TRB. However, Δ -TRB is not implementable in $\mathcal{TC}(\Delta)$ by message-passing without zero processing time assumptions. Henceforth, by introducing increasingly stronger connectivity assumptions, we have provided two *implementable* connectivity classes, namely $\mathcal{TC}'(\beta)$ and $\mathcal{TC}''(\alpha, \beta)$, as well as two respective implementations of Δ -TRB in these classes.

Instead of using a general, system-wide bound, in $\mathcal{TC}''(\alpha, \beta)$, journeys have a bound, Γ , that can be calculated in terms of parameters that are linked to entities of real-world networks, namely, temporal bounds on the appearance of a link and on the stability of the link, as well as the diameter of the network. It should be noted that for every instance of the $\mathcal{TC}''(\alpha, \beta)$ class, there exists a global bound that can be used to derive an instance of the upper-level classes, which yields to define relations between the classes, as summarized in Figure 4.7.

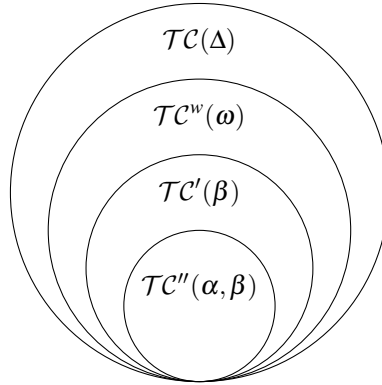
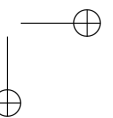
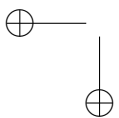
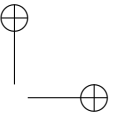
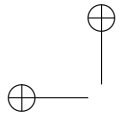


Figure 4.7: $\mathcal{TC}''(\alpha, \beta) \subset \mathcal{TC}'(\beta) \subset \mathcal{TC}^w(\omega) \subset \mathcal{TC}(\Delta)$

We have shown that Consensus at Δ -component level is easily reduced to Δ -TRB. Nevertheless, extending the stability period of Δ -components is necessary to solve Con-





4.5 Conclusions

sensus at system level.

Observe that the $\mathcal{TC}'(\beta)$ impose a β minimum time of activity on each e_i of a β -journey. In the same way, in ω -components we offer a timely component definition based on ω -journeys. An ω -component based connectivity class is weaker than $\mathcal{TC}'(\beta)$ since it does not assume a β bound, but only a minimum a priori known drift between the lifetime of every edge e_i at any time t and the latency $\zeta(e, t)$. We have proved that this connectivity model is the weakest connectivity class allowing implementable Δ -TRB algorithms.

Finally, the proposed connectivity classes are represented in Figure 4.8 according to the inclusion tree classification introduced in Section 2.3.2.2.

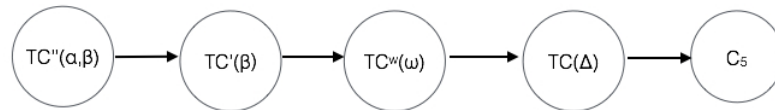
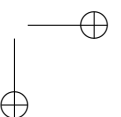
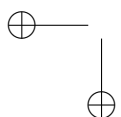
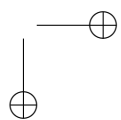
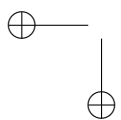
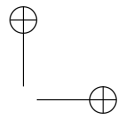
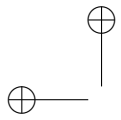
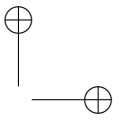
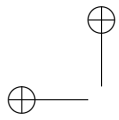


Figure 4.8: The proposed connectivity classes classified in terms of relations of inclusion with respect of class 5 of [16].







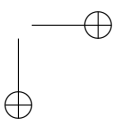
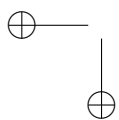
CHAPTER

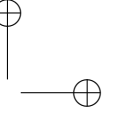
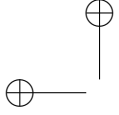
5

Eventual Leader Election in Dynamic Distributed Systems

In the previous chapter we have provided several TRB algorithms which, under synchrony assumptions, are equivalent to Consensus. However, TRB in partially synchronous systems is not equivalent to Consensus.

As mentioned in Chapter 2, many Consensus protocols rely on the existence of a leader election service. Failure detectors in general, and Ω in particular, were initially defined to face asynchrony and crash failures. Nevertheless, nowadays there exist several proposals that implement Ω under dynamic characteristics like unknown membership, crash-recovery or infinite arrival [57, 59, 63, 64]. In this chapter we will focus on leader election to board the agreement problem in dynamic systems, including process





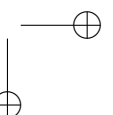
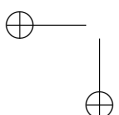
5. EVENTUAL LEADER ELECTION IN DYNAMIC DISTRIBUTED SYSTEMS

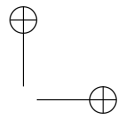
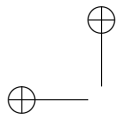
mobility and the several dimension of dynamicity described in Chapter 3.

Recall that the type of systems we are considering are highly dynamic, and processes execute in different types of devices, some of them mobile and connected via wireless networks. As a consequence, communication can sometimes fail, messages can get lost, and the system can even partition. Due to that, in Chapter 4 we have studied the limits on process mobility in order to Consensus be solvable. Observe that the assumption of some kind of Δ -component in the system is necessary to success in the attempt of reaching agreement, be it on a leader (Ω) or on a proposed value (Consensus).

The aim of this chapter is to provide an algorithm to solve leader election in a system model as weak (i.e., dynamic) as possible. To do so, we assume that the system alternates periods of “good” and “bad” behavior, i.e, the algorithm is aimed to work in level s in terms of the categorization we introduced in Chapter 3. Recall that this system model is weaker than the one based in the existence of unknown Global Stabilization Times after which the system behaves timely forever, corresponding to level n . Also, while previous works like the timed asynchronous model by Cristian and Fetzer [24] or the Heard-Of model of Charron-Bost and Schiper [20], assume good/bad periods in terms of asynchrony and failures for a finite and static set of processes, we extend the assumption to all the dimensions of the systems as described in Chapter 3. We call \mathcal{M}^* to this model.

We extend the eventual leadership properties to adapt them to Δ -components based dynamic scenarios (as we did with Δ -Consensus). The resulting eventual leader election is based on $\Delta\Omega$. Recall that $\Delta\Omega$ is the non-mobile dynamic leader election proposed by Larrea et al. [59]. We extend $\Delta\Omega$ in order to tolerate two new properties that take into account temporal-subgraph joins/fragmentations, which in general corresponds to a Δ -component based system model, introduced in Chapter 4. We call $\Delta^*\Omega$ to the resulting eventual leader election. Then, we propose a new leader election algorithm for \mathcal{M}^* .





5.1 Problem Specification and System Model

Finally, we compare our model to others using the framework defined in Chapter 3 and presents some conclusions obtained from the results of simulations.

Note that we will start using the definitions of $\Delta^*\Omega$ as defined in a previous work of ours [36] and we redefine them in terms of the TVG formalism of Chapter 3. In this way we integrate the fault-tolerant perspective into the (more general) connectivity models provided by the TVG formalism.

5.1 Problem Specification and System Model

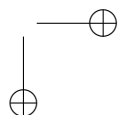
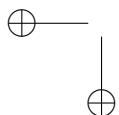
5.1.1 Problem Specification

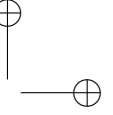
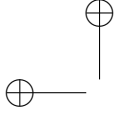
We first specify the properties in terms of an Ω -like failure detector in dynamic distributed systems, denoting the resulting class of failure detectors $\Delta^*\Omega$. We show this relation with the connectivity model defined in Chapter 4. Then, we propose a weak dynamic distributed system model called \mathcal{M}^* .

Chandra et al. proposed in [18] the Ω failure detector class, which provides an eventual leader election functionality. The system model in which Ω was originally proposed was a static distributed system with reliable communication links and known membership, where processes may fail by crashing. The property satisfied by Ω is the following:

- **EL** (Eventual Leadership): There is a time after which all the correct processes permanently *trust*, i.e., have as their leader, the same correct process.

More recently, Larrea et al. [59] have proposed a redefinition of the eventual leader election problem in non-mobile dynamic systems with unknown initial membership, where processes may join and crash/leave arbitrarily. The two properties that define the Dynamic Omega failure detector class (denoted $\Delta\Omega$) are the following:





5. EVENTUAL LEADER ELECTION IN DYNAMIC DISTRIBUTED SYSTEMS

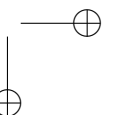
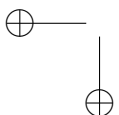
- **EL-NI** (Eventual Leadership in Non-Increasing systems): If after some time the system does not increase (i.e., no new process joins the system), then a correct leader must eventually be elected.
- **EL-ND** (Eventual Leadership in Non-Decreasing systems): If after some time the system does not decrease (i.e., no process leaves the system or crashes), then (1) a leader must eventually be elected, and (2) the new processes that join the system after a leader has been elected have to eventually consider this process as their leader.

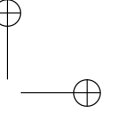
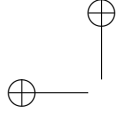
Note that, by the definition, $\Omega \subset \Delta\Omega$.

The authors remark that the continuous joins and leaves of processes could affect the convergence of any leader election algorithm. However, if processes remain “long enough” in the system, then a leader will be elected, even if any of the **EL-NI** or **EL-ND** properties are not fully satisfied. In other words, it is sufficient in practice that a “long enough” period of time allows to elect a leader that does neither leave the system nor crash, after which processes may continue joining and leaving/crashing arbitrarily and forever. Thus, the properties **EL-NI** and **EL-ND** must be seen as conditions under which the leader election is ensured.

In general, in dynamic distributed systems we can not assume a unique and fully connected graph. Even if the system converges to a common and correct leader, it can be impossible to ensure the communication with the leader. For this reason, in [36], we defined a connectivity property for the leader election problem in dynamic distributed systems:

- **EL_{BC}** (Eventual Leadership with Bidirectional Connectivity): Eventually and for sufficiently long, there is a bidirectional path between the leader and the rest of processes.





5.1 Problem Specification and System Model

Observe that, in terms of the connectivity models defined in Chapter 4, since a Δ -component C provides by definition bidirectional connectivity among processes in C . Consequently, the Δ -component trivially satisfies the \mathbf{EL}_{BC} property.

Additionally, also in [36], we redefined the dynamic leader election properties $\mathbf{EL-NI}$ and $\mathbf{EL-ND}$ for systems with process mobility as follows:

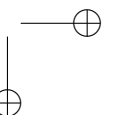
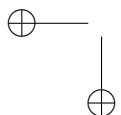
- \mathbf{EL}_{BC-NI} (Eventual Leadership with Bidirectional Connectivity in Non-Increasing systems): If after some time the system does not increase, then a correct leader must eventually be elected and this leader must satisfy the \mathbf{EL}_{BC} property.
- \mathbf{EL}_{BC-ND} (Eventual Leadership with Bidirectional Connectivity in Non-Decreasing systems): If after some time the system does not decrease, then (1) a leader must eventually be elected and this leader must satisfy the \mathbf{EL}_{BC} property, and (2) the new processes that join the system after a leader has been elected have to eventually consider this process as their leader.

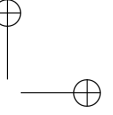
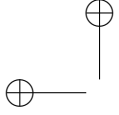
Again, in terms of the Δ -component based connectivity model of Chapter 4, the previous properties result as follows:

Definition 5.1.1. \mathbf{EL}_{BC-NI} (Eventual Leadership with Bidirectional Connectivity in Non-Increasing systems): *If after some time a Δ -component C does not increase, then a correct leader belonging to C must eventually be elected.*

Definition 5.1.2. \mathbf{EL}_{BC-ND} (Eventual Leadership with Bidirectional Connectivity in Non-Decreasing systems): *If after some time a Δ -component C does not decrease, then (1) a leader belonging to C must eventually be elected by every process in C , and (2) the new processes that join to C after a leader has been elected have to eventually consider this process as their leader.*

As we have said, a dynamic distributed system allows the existence of more than one Δ -component. Due to process mobility and system dynamicity, the number of components can continuously evolve, either by the join of two or more components





5. EVENTUAL LEADER ELECTION IN DYNAMIC DISTRIBUTED SYSTEMS

or by the fragmentation of a component. In order to cope with these situations, two additional properties are defined for the leader election problem in dynamic distributed systems:

Definition 5.1.3. EL-GJ (*Eventual Leadership in a Graph Joining situation*¹): *If two or more Δ -components join into a single Δ -component, then the resulting Δ -component must satisfy the **EL_{BC-NI}** and **EL_{BC-ND}** properties.*

Definition 5.1.4. EL-GF (*Eventual Leadership in a Graph Fragmentation situation*): *If a Δ -component is fragmented in two or more Δ -components, then each of the resulting Δ -components must satisfy independently the **EL_{BC-NI}** and **EL_{BC-ND}** properties.*

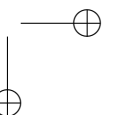
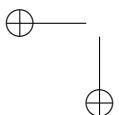
Observe that while the **EL_{BC-NI}** and **EL_{BC-ND}** properties consider a single Δ -component, the **EL-GJ** and **EL-GF** properties consider the whole system, potentially composed of several Δ -components. By analogy with the classes Ω and $\Delta\Omega$, in [36] we call *Mobile Dynamic Omega*, denoted $\Delta^*\Omega$, to the class of failure detectors satisfying the **EL-GJ** and **EL-GF** properties². Note that, by the definition of **EL-GJ** and **EL-GF**, $\Delta^*\Omega$ implicitly satisfies **EL_{BC-NI}** and **EL_{BC-ND}** in each Δ -component. Note also that $\Delta\Omega \subset \Delta^*\Omega$.

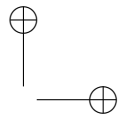
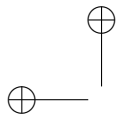
5.1.2 System Model \mathcal{M}^*

In this section, we propose a weak mobile dynamic distributed system model \mathcal{M}^* . According to the categorization framework proposed in Chapter 3, \mathcal{M}^* has all dimensions tagged with the level s , i.e., $\mathcal{M}^* = \mathcal{M}(T^s, \mathcal{P}_F^s, C_F^s, G_{\#}^s, G_{\Pi}^s, G_D^s, G_S^s)$ following the notation presented in Chapter 3. This model will subsequently be used by a leader election algorithm implementing $\Delta^*\Omega$. In the remainder of this section we will describe the main characteristics of model \mathcal{M}^* , and in particular its stability assumption.

¹We have maintained the original property names given in [36].

²In [36] **EL-GJ** and **EL-GF** were defined in terms of subgraphs, not in terms of Δ -components as in the present work.





5.1 Problem Specification and System Model

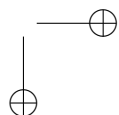
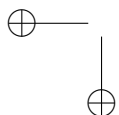
As in general in the literature, in \mathcal{M}^* processes are assumed to be synchronous. Moreover, for simplicity processing times are considered as negligible compared to communication delays. Each process in the system has a unique identifier, and is equipped with a clock that can measure real-time intervals. However, clocks are not synchronized. Regarding the process failure model, as denoted by the level s , we assume a crash-recovery model where processes can leave/crash and join/recover the system at will.

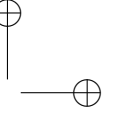
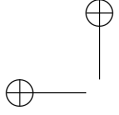
To send a message, processes have access to a reliable broadcast communication primitive. From a communication reliability and synchrony point of view, we assume that periodically all links act as eventually timely links [3]. An eventually timely link allows messages to get lost, but ensures that eventually every message that is sent is received before an unknown bound.

According to the level s of each dimension of \mathcal{M}^* , we assume that \mathcal{M}^* alternates periods of good and bad behavior in the line of the timed asynchronous model of Cristian and Fetzer [24]. Good periods are both “stable enough” (from a dynamicity and timeliness point of view) and “long enough” (from a duration point of view) to elect a leader in every graph of the system. We also assume that good periods occur infinitely often.

Recall that we are considering that a process is correct with respect to a Δ -component. In this way, the stability assumption of a good period, necessary to elect a leader, relies on the non occurrence of Δ -component partitioning.

Also, during a good period the Δ -components can “grow”. In particular, new nodes may be added (e.g., due to a new process join, or upon the recovery of a crashed process), and even two disconnected graphs may join into a single graph. However, new node additions and/or graph joinings during a good period must not affect the s level for all the dimensions of \mathcal{M}^* . In particular, the graph membership and graph diameter must remain bounded (with possibly unknown bounds).





5. EVENTUAL LEADER ELECTION IN DYNAMIC DISTRIBUTED SYSTEMS

5.2 A Leader Election Algorithm for \mathcal{M}^*

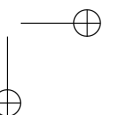
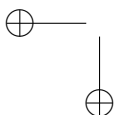
In this section, we propose a new leader election algorithm that implements the class $\Delta^*\Omega$ in the model \mathcal{M}^* , i.e., satisfies the **EL-GJ** and **EL-GF** properties previously defined in good periods. Without loss of generality, we specifically choose the class \mathcal{TC}^w to implement algorithm. Recall that \mathcal{TC}^w is the weakest class of Δ -components in which Terminating Reliable Broadcast can be implemented in a message-passing system.

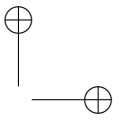
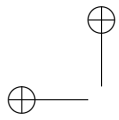
5.2.1 A Reliable Broadcast Primitive for \mathcal{M}^*

In contrast to static networks, where reliable links are usually assumed, and thus a message broadcast by a process p_i will be delivered by all correct processes dynamic distributed systems this is no longer true. Hence, we need to provide for \mathcal{M}^* a reliable broadcast primitive in a Δ -component C such that, sending a finite number of copies of each message, a process p_i is able to deterministically communicate with all processes in C in a bounded time by temporal paths, i.e., ω -journeys. However, since bounds are not known a priori, as usual in partial synchronous approaches the algorithm must learn the communication bounds. Consequently, the proposed reliable broadcast will have an eventual approach. Note that, for a process to learn a communication bound, there must exist a periodical send pattern included in the algorithm in order to processes be able to learn that bound. We assume that every process in the system periodically broadcast messages.

In this regard, the proposed primitive must satisfy the following properties:

- **Validity:** Eventually if a correct process belonging to a Δ -component C sends a message, then some correct process belonging to C will eventually deliver that message.





5.2 A Leader Election Algorithm for \mathcal{M}^*

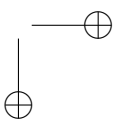
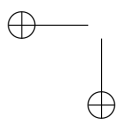
- **Agreement:** Eventually if a correct process belonging to a Δ -component C delivers a message, then all correct processes belonging to C eventually deliver that message.
- **Integrity:** Eventually every correct process belonging to a Δ -component C delivers the same message at most once and only if that message has been sent by a process in the system.

Figure 5.1 illustrates the proposed *eventual reliable broadcast*. In essence, the algorithm follows the scheme proposed in classical reliable broadcast. The differences relies on the mechanisms required in order to cope with temporal paths as we made for the TRB algorithms in Chapter 4. The most relevant characteristics of the proposed reliable broadcast are the following:

Differently to the TRB algorithms proposed in Chapter 4, the system model follows an eventual approach, i.e., communication bounds are not known but are learned during the execution. Specifically we use “echo” messages to learn this bounds. Observe that, according to Δ -component’s definition a receiver of a message m will communicate with the sender of m in at most Δ time again. This means that if the first reception of m lasted Δ time and the receptor maintains broadcasting m during other Δ time, the original sender of m will receive an “echo” of m in at most 2Δ . Note, however, that for the algorithm to converge it is required that both sender and receiver process *timeout-RBConnectivity* _{i} $\geq \Delta$. For timers to converge, it is required a periodical communication pattern in order a process be able to approximate the bounds, and thus bounds can be learned by a timeout mechanism.

Correctness Proof

Lemma 5.2.1. *The algorithm proposed in Figure 5.1 satisfies the Validity property of Reliable Broadcast in Δ -components.*



5. EVENTUAL LEADER ELECTION IN DYNAMIC DISTRIBUTED SYSTEMS

```

1  $W \leftarrow v \in (0, \omega]$ 
2  $timer\_RBCConnectivity_i \leftarrow 0$ 
3  $timeout\_RBCConnectivity_i \leftarrow 0$ 

4  $RBroadcast(m)$  at time  $t_i$ :
5   while  $now() < t_i + timeout\_RBCConnectivity_i$  do
6     broadcast( $m$ )
7     wait( $W$ )

8   when a message  $m$  is received from  $p_j \neq p_i$  for the first time:
9     if  $sender(m) \neq p_i$  then
10      RDeliver( $m$ )
11      while  $now() < t_i + timeout\_RBCConnectivity_i$  do
12        broadcast( $m$ )
13        wait( $W$ )
14     else
15      reset  $timer\_RBCConnectivity_i$ 

16   when  $timer\_RBCConnectivity_i$  expires:
17     increase  $timeout\_RBCConnectivity_i$ 

```

Figure 5.1: Algorithm implementing Reliable Broadcast by ω -journeys (code for process p_i).

Proof. Let us assume that p_i is a correct process belonging to an ω -component C and $timer_RBCConnectivity_i$ is higher than Δ . By definition, every process of the system periodically broadcasts a message m by Line 6 during $timer_RBCConnectivity_i$ time. Let us assume that p_i broadcasts a message m at time t_i . By definition of ω -component and Lemma 4.2.4, since m is broadcast each time unit and ω is higher than that period, p_j in C will *RDeliver* a copy of m by Line 10.

On the other hand, if a process p_i has a $timer_RBCConnectivity_i$ lower than Δ , the message m could not be received by any other process since the connectivity is assured by the bound Δ and Lemma 4.2.4 is not applicable. In that case, $timer_RBCConnectivity_i$ will expire and will be increased by Lines 16-17. Observe that every process in the system is assumed to broadcast a new message m periodically. Consequently, for every correct process in C eventually $timer_RBCConnectivity_i$ will be higher than Δ and consequently every message broadcast from that moment on by the primitive described in Figure 5.1 will be *RDelivered* by Line 10. \square



5.2 A Leader Election Algorithm for \mathcal{M}^*

Lemma 5.2.2. *The algorithm proposed in Figure 5.1 satisfies the Agreement property of Reliable Broadcast in Δ -components.*

Proof. Let us assume that p_i is a correct process belonging to an ω -component C and $timer_RBCConnectivity_i$ is higher than Δ . If p_i receives a message m broadcasts by p_j , since p_i is not the sender, by Lines 10-13, p_i will *RDeliver* m and will broadcast a copy of m during $timer_RBCConnectivity_i$ time. By definition, every process of the system periodically broadcasts a message m by Line 6 during $timer_RBCConnectivity_i$ time. By definition of ω -component and Lemma 4.2.4, since m is broadcast each time W unit and ω is higher or equal to W , p_j in C will *RDeliver* by Line 10 and propagate a copy of m at least one hop in the temporal path.

On the other hand, if a process p_i has a $timer_RBCConnectivity_i$ lower than Δ , the message m could not be received by any other process since the connectivity is assured by the bound Δ and Lemma 4.2.4 is not applicable. In that case, $timer_RBCConnectivity_i$ will expire and will be increased by Lines 16-17. Observe that every process in the system is assumed to broadcast a new message m periodically. Consequently, for every correct process in C eventually $timer_RBCConnectivity_i$ will be higher than Δ . From that moment on, Lemma 4.2.4 is applicable for every process belonging to any ω -journey. Therefore, every message m received by any process p_i belonging to C subsequently will be *RDelivered* and rebroadcast by Line 10-13 and thus *RDelivered* by every correct process in C . □

Lemma 5.2.3. *The algorithm proposed in Figure 5.1 satisfies the Integrity property of Reliable Broadcast in Δ -components.*

Proof. Observe that messages are only broadcast by Line 4. By Line 8, a message is only delivered the first time it is received by a process. On the other hand, by the algorithm, no other messages are inserted in the system. □

Theorem 5.2.4. *The algorithm in Figure 5.1 satisfies the eventual reliable broadcast properties of Reliable Broadcast in Δ -components.*

Proof. Follows directly from Lemmas 5.2.1, 5.2.2 and 5.2.3. □

5. EVENTUAL LEADER ELECTION IN DYNAMIC DISTRIBUTED SYSTEMS

5.2.2 $\Delta^*\Omega$ Implementation

In the algorithm proposed in Figure 5.4, we introduce a mechanism that detects whether a leader is connected with some other process or not. We exploit the radiation nature of the broadcast primitive for implementing the mechanism to detect self-isolation situations. *Connected* processes are those that are able to receive messages from other processes, while *unconnected* processes are those that, being isolated, are executing the algorithm, trying to communicate with any other process. Figure 5.2 shows the process state diagram with the transitions between states. The goal of this mechanism is preventing isolated processes from being leaders. The timer in charge of this mechanism in a process p_i is called *timer_connectivity_i*, and is handled by Task 4.

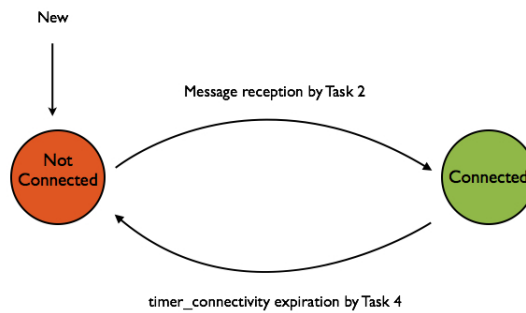


Figure 5.2: Process states during the algorithm.

5.2 A Leader Election Algorithm for \mathcal{M}^*

```

1 Initialization:
2 set_unconnected()

3 Repeat forever every  $\beta$  time units:                                Message broadcast
4   if ( $connected_i = FALSE$ ) then
5     | RBroadcast(JOIN,  $i$ )
6   else if ( $leader_i = i$ ) then
7     | RBroadcast(LEADER,  $i$ ,  $|joins_i|$ )

8 when a message  $m$  is received from  $p_j \neq p_i$ :                        Msg reception
9   if ( $connected_i = FALSE$ ) then
10    |  $connected_i \leftarrow TRUE$ 
11    | set_leader( $i$ ,  $|joins_i|$ )
12    | reset timer_connectivity $_i$  to timeout_connectivity $_i$  to check connectivity
13    | RBroadcast(JOIN,  $i$ )
14   if (message is of type (JOIN,  $k$ )) then
15     | if ( $k \notin joins_i$ ) then
16       |  $joins_i \leftarrow joins_i \cup \{k\}$ 
17       | update_if_I_am_leader()
18       | RBroadcast(JOIN,  $k$ )
19   else if (message is of type (LEADER,  $\ell$ , num_joins $_\ell$ )) then
20     | if ( $(leader_i = i) \wedge (\ell = i)$ ) then
21       | reset timer_connectivity $_i$  to timeout_connectivity $_i$  to check connectivity
22     | else if (best_candidate( $\ell$ , num_joins $_\ell$ )) then
23       | set_leader( $\ell$ , num_joins $_\ell$ )
24       | reset timer_leader $_i$  to timeout_leader $_i$  to monitor leader $_i$ 
25       | RBroadcast(LEADER,  $\ell$ , num_joins $_\ell$ )

26 when timer_leader $_i$  expires:                                        Leader timeout expiration
27   | increase the timeout value timeout_leader $_i$  used to control timer_leader $_i$ 
28   | set_leader( $i$ ,  $|joins_i|$ )
29   | reset timer_connectivity $_i$  to timeout_connectivity $_i$  to check connectivity

30 when timer_connectivity $_i$  expires:                                Connectivity timeout expiration
31   | if leader $_i = i$  then
32     | increase the timeout value timeout_connectivity $_i$  user to control timer_connectivity $_i$ 
33     | set_unconnected()

```

Figure 5.3: Algorithm implementing $\Delta^*\Omega$ in model \mathcal{M}^* (code for process p_i).

5. EVENTUAL LEADER ELECTION IN DYNAMIC DISTRIBUTED SYSTEMS

34 Procedure *set_unconnected*: Disconnect a process
35 | *connected*_{*i*} ← *FALSE*
36 | *joins*_{*i*} ← {*i*}
37 | *leader*_{*i*} ← *i*
38 | *num_joins*_{*leader*_{*i*}} ← 1

39 Procedure *set_leader*(*id*, *num_joins*): Change leader
40 | *leader*_{*i*} ← *id*
41 | *num_joins*_{*leader*_{*i*}} ← *num_joins*

42 Procedure *update_if_I_am_leader*: Update leader info
43 | **if** *leader*_{*i*} = *i* **then**
44 | | *num_joins*_{*leader*_{*i*}} ← |*joins*_{*i*}|

45 Function *best_candidate*(*ℓ*, *num_joins*_{*ℓ*}) **returns** Boolean Leadership criteria
46 | **if** ((*num_joins*_{*ℓ*} > |*joins*_{*i*}|) ∧ ((*num_joins*_{*ℓ*} ≥ (*num_joins*_{*leader*_{*i*}}, *leader*_{*i*}))) **then**
47 | | **return** *TRUE*
48 | **return** *FALSE*

Figure 5.4: Auxiliary functions.



5.2 A Leader Election Algorithm for \mathcal{M}^*

The algorithm uses the following leadership criteria: the leader process will be the oldest one of the connected component with the highest identifier (see Line 46)¹.

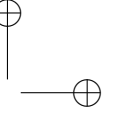
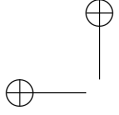
Processes initiate the execution of the algorithm as unconnected, until they receive a message from any other process in the system. Whenever a message is received by a unconnected process, the process becomes connected and all its data structures are re-initialized (Lines 9-13).

Task 1 is in charge of broadcasting periodical messages. If a process p_i is unconnected, by Task 1 it broadcasts JOIN messages (Line 5). Otherwise, if p_i is connected and $leader_i = i$, p_i broadcasts LEADER messages (Line 7), postulating itself as leader. All neighbor processes of p_i , i.e., those which have one hop communication with p_i according the assumptions of ω -components, will become connected at the reception of one of the message sent in Task 1. When a process becomes connected, it starts considering itself as leader (Line 11), and therefore, broadcasts also LEADER messages periodically.

The reception of a JOIN message at process p_i , sent by process p_k , implies that p_i is “older” than p_k (or both are equal). For maintaining this information, p_i checks whether k is in a set $joins_i$, and introduces k into $joins_i$ if k was not already there (Line 16). Additionally, if $leader_i = i$ this inclusion must be consistent with the $num_joins_{leader_i}$ variable. This is ensured by p_i executing Line 17.

The reception of a LEADER message at process p_i has three goals: checking the connectivity with the graph, electing a common leader, and maintaining a coherent leader among the execution. First of all, when a LEADER message is received at p_i , if the message was previously sent by p_i itself, then p_i recognizes this message as an *echo*. This fact remarks that some other process has received a LEADER message from

¹For the leadership criteria, we consider that the $(a_1, a_2) > (b_1, b_2)$ relationship returns true if $a_1 > b_1$ or $a_1 = b_1 \wedge a_2 > b_2$. Similarly, $(a_1, a_2) \geq (b_1, b_2)$ returns true if $a_1 > b_1$ or $a_1 = b_1 \wedge a_2 \geq b_2$.



5. EVENTUAL LEADER ELECTION IN DYNAMIC DISTRIBUTED SYSTEMS

p_i and has resent it. For this reason, p_i resets $timer_connectivity_i$ (Line 21), preventing it to expire.

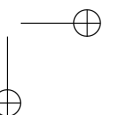
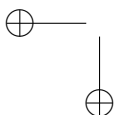
If p_i receives a LEADER message from a process $\ell \neq i$, then the message is not an echo and p_i must decide whether p_ℓ must be p_i 's leader or not.

Observe that, since links are assumed to be eventually timely, even during a good period some messages may be not received in time. The non reception of some JOIN messages by a leader process p_ℓ could imply that $|joins_i| > |joins_\ell|$ and therefore, the leadership criteria would not be satisfied. For coping with this scenario, process p_i always checks whether the leader p_ℓ still satisfies the leadership criteria or p_i itself is a better candidate (Line 46). Note that, the leader process eventually will receive the same JOIN messages as the rest of processes, maintaining stable the leadership criteria.

If a new process p_i joins the graph G_t , and assuming that the leader of G_t is p_ℓ , then p_i will eventually receive a LEADER message from p_ℓ and p_ℓ will receive a JOIN message sent by p_i . The reception at p_i of a LEADER message satisfying the leadership criteria will result in the adoption of p_ℓ as p_i 's leader by Line 23. In contrast, if two graphs join into a single one, messages sent by both previous leaders will be received by each other. The previous leader not satisfying the leadership criteria will eventually stop broadcasting LEADER messages.

Whenever p_i sets its leader to ℓ (Line 23), p_i resets a timer $timer_leader_i$ on its leader (Line 24) and forwards the LEADER message to the rest of the system (Line 25).

Task 3 is in charge of monitoring $leader_i$ if $leader_i \neq i$. If a LEADER message is not received in time by p_{leader_i} , then $timer_{leader_i}$ will expire. As a consequence, p_i will consider itself as leader (Line 28) and will reset again the timer $timer_connectivity_i$ (Line 29) for self-monitoring p_i 's connectivity. Observe that, by Line 27, $timer_leader_i$ will be increased in order for p_i to eventually avoid erroneous premature suspicions on $leader_i$.



5.2 A Leader Election Algorithm for \mathcal{M}^*

Task 4 decides if a connected and leader process becomes isolated or not by monitoring if any of its LEADER messages has been resent by some neighbor process before a timer expires. If a leader process p_i does not receive such an echo message before $timer_connectivity_i$ expires, then p_i will become unconnected and will reset its data structures (see Line 33). Similarly to $timer_leader_i$, each time $timer_connectivity_i$ expires it is increased in order to eventually avoid premature expiration (Line 32).

5.2.2.1 Correctness proof of the algorithm

In this section we prove that the algorithm proposed in Figure 5.4 implements $\Delta^*\Omega$ in a \mathcal{M}^* model.

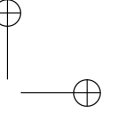
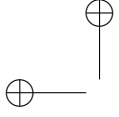
We will prove the correctness of the algorithm in two steps. For simplicity, we will first prove that the algorithm implements $\Delta^*\Omega$ in a stronger system model \mathcal{M}^- that assumes reliable links instead of the s one:

$$\mathcal{M}^- = \mathcal{M}(T^s, \mathcal{P}_F^s, \mathcal{C}_F^b, G_{\#}^s, G_{\Pi}^s, G_D^s, G_S^s)$$

Then, we will study how the assumption of finite but unknown message losses on links may affect the time of convergence of the algorithm but not its correctness. Then, we will be able to assure that the algorithm also implements $\Delta^*\Omega$ in the system model \mathcal{M}^* .

The **EL_{BC}-NI** and **EL_{BC}-ND** properties define a correct leader election behavior in a single ω -component. Similarly, the **EL-GJ** and **EL-GF** properties define a correct leader election behavior in the whole system. Note however that **EL-GJ** and **EL-GF** imply that **EL_{BC}-NI** and **EL_{BC}-ND** are satisfied in each ω -component of the system. We will first prove the **EL_{BC}-NI** and **EL_{BC}-ND** properties, and then the **EL-GJ** and **EL-GF** properties.

Implicitly, by the assumption of an ω -component, during a good period the graph diameter is also bounded by d . We also assume an unknown bound in the number of ω -components and in the number of processes in an ω -component. Of course, processes



5. EVENTUAL LEADER ELECTION IN DYNAMIC DISTRIBUTED SYSTEMS

can crash and recover, but only if those crashes do not affect the connectivity among the rest of processes.

We also assume that the duration of the good period is long enough for the algorithm to converge. Thus, any time instant t of the proof occurs in the considered good period.

Finally, from now $p_\ell \in C$ will be the *non faulty* process with the highest cardinality $|joins_\ell|$, with $connected_\ell = TRUE$ permanently and lowest identifier.

Regarding notation, given an ω -component C in $\mathcal{G}_{(\mathcal{T}^-, \mathcal{T}^+)}$ we denote C_t the temporal subcomponent in $\mathcal{G}_{[t, t+\Delta)}$.

Lemma 5.2.5. *In \mathcal{M}^- , $\forall t \in \mathcal{T}$, $\forall p_i \in C_t$, eventually and permanently $connected_i = TRUE$.*

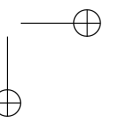
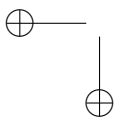
Proof. Observe the adaptive stability mechanisms of variable $timer_connectivity_i$ in Task 4 (for the leader) and the $timer_leader_i$ in Task 3. By Lines 27 and 32, both variables increase their value each time they expires. Since no one of them are reinitialized even changing to unconnected mode and by Theorem 5.2.4 eventually every message broadcast by processes in C will be delivered, we can say that eventually the $timer_connectivity_\ell$ and all $timer_leader_i$ will never expire again, i.e., Line 33 will never be executed again since that moment. Consequently, $\forall t, \forall p_i \in C_t$, eventually and permanently $connected_i = TRUE$. \square

Lemma 5.2.6. *In \mathcal{M}^- , for every message m sent by a process $p_i \in C_t$ at time t , m will be eventually delivered by every process $p_j \in C_t$.*

Proof. Follows directly from the assumption of reliable links, the Definition 4.4.2, Theorem 5.2.4 and the re-broadcasting mechanism of Lines 18 and 25. \square

Definition 5.2.7. *We say that a process p_i joins C at a time t when, after entering the system, p_i is able to communicate timely with every process $p_j \in C_t$.*

Lemma 5.2.8. *In \mathcal{M}^- , $\forall t, \forall p_i \in C_t$ with $connected_i = TRUE$ permanently, for all process $p_j \in C'_t$ where $t' \geq t$ and p_j does not have $connected_j = TRUE$ permanently at t , then eventually $|joins_j| < |joins_i|$.*



5.2 A Leader Election Algorithm for \mathcal{M}^*

Proof. We know that eventually every process p_k in C will have $connected_k = TRUE$, however, any process p_j that do not have $connected_j = TRUE$ permanently at t , eventually will execute Line 33 deleting all gathered information and becoming unconnected. Beside, every processes p_i that has $connected_i = TRUE$ permanently in t will never execute Line 33 again, and thus, no p_i will broadcast a JOIN message again. When p_j joins the C at time $t' \geq t$, it broadcasts a JOIN message by Task 2 before becoming connected (Line 13). By Lemma 5.2.6, this message will eventually be delivered by Task 2 of p_i , which will include $j \in joins_i$ (Line 16). Taking into account that $join_i$ will never decrease, then $|joins_j| < |joins_i|$. Consequently, $\forall p_i \in C_t$ with $connected_i = TRUE$ permanently, for all process $p_j \in C'_t$ where $t' \geq t$ and p_j does not have $connected_j = TRUE$ permanently at t , then eventually $|joins_j| < |joins_i|$. \square

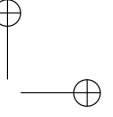
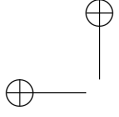
Lemma 5.2.9. *In \mathcal{M}^- , if $p_i \in C_t$ such that $connected_i = TRUE$ forever, then eventually $|joins_i| \leq |joins_\ell|$ permanently.*

Proof. The proof is by contradiction. Let us assume that p_i is connected permanently, with $i \notin joins_\ell$ eventually and permanently. In other words, p_ℓ has not delivered any JOIN message sent by p_i . This can only happen if p_i becomes connected permanently before p_ℓ (i.e., without p_ℓ receiving a JOIN message from p_i after p_ℓ has become connected permanently). In this case, p_i will never broadcast again a JOIN message by Line 13 and thus for p_ℓ $i \notin joins_\ell$ permanently. In contrast, p_i by Line 14 and Lemma 5.2.6, eventually will deliver the last JOIN message sent by p_ℓ before it becomes connected permanently, and thus $\ell \in joins_i$ permanently by Line 16. Consequently, p_ℓ will not be the process with the highest number of elements in its $joins_\ell$ set in C_t (a contradiction). Consequently, eventually $|joins_i| \leq |joins_\ell|$. \square

Lemma 5.2.10. *In \mathcal{M}^- , $\forall t, \forall p_i \in C_t$, if $leader_i = j$ and $connected_j = FALSE$, then eventually $leader_i \neq j$.*

Proof. Let us assume that $leader_i = j$ and $connected_j = FALSE$. By the algorithm, process p_i will have $timer_leader_i$ active, and p_i is waiting for a next LEADER message from p_j . We have two possible cases:

1. $timer_leader_i$ expires before p_j becomes connected again.
2. p_j becomes connected and broadcasts a LEADER message (Line 7), which is delivered by p_i before $timer_leader_i$ expires. In this case, when p_j becomes un-



5. EVENTUAL LEADER ELECTION IN DYNAMIC DISTRIBUTED SYSTEMS

connected, by Line 33 p_j will initialize $joins_j$ to $\{j\}$ (Line 36), and consequently p_j will broadcast a value 1 in the LEADER message (Line 7). Upon the reception of that message, by not satisfying the condition of Line 46 p_i will not re-set the timer for p_j .

In both cases, $timer_leader_i$ will expire for p_j and p_i will execute Task 3, setting $leader_i = i \neq j$ (Line 28). Consequently, $\forall t, \forall p_i \in C_t$, if $leader_i = j$ and $connected_j = FALSE$, then eventually $leader_i \neq j$. \square

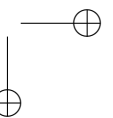
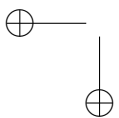
Lemma 5.2.11. *In \mathcal{M}^- , eventually $leader_\ell = \ell$ permanently.*

Proof. By Lemma 5.2.5, eventually and permanently $connected_\ell = TRUE$. If any other process p_i has $|joins_i| > |joins_\ell|$, by Lemma 5.2.9, then eventually p_i will set $connected_i = FALSE$ resetting all data structures by Line 34 or will crash. As a consequence, by Lemma 5.2.10 eventually $leader_\ell \neq i$ for every process p_i such that $|joins_i| > |joins_\ell|$. Observe that, by Lemma 5.2.8, p_ℓ will include in $joins_\ell$ (Line 16) both new joining/recovering processes and processes becoming connected. Joining/recovering processes broadcast a JOIN message either by Line 5 or Line 13.

By the previous reasoning, the last process p_i such that $|joins_i| > |joins_\ell|$ which sets $connected_i = FALSE$ will make p_ℓ set $leader_\ell$ to ℓ by Line 28. Since p_ℓ has the highest number of elements in its $joins_\ell$ set, any subsequent LEADER message delivered by p_ℓ will not satisfy the condition of Lines 46. Consequently, eventually $leader_\ell = \ell$ permanently. \square

Lemma 5.2.12. *In \mathcal{M}^- , $\forall t, \forall p_i \in C_t$, there is a time after which $timer_leader_i$ will not expire.*

Proof. By Lemmas 5.2.5 and 5.2.11, eventually p_ℓ will have permanently $connected_\ell = TRUE$ and $leader_\ell = \ell$ respectively. Hence, by Task 1 p_ℓ will be continuously broadcasting LEADER messages (Line 7). Every process $p_i \in C_t$ will deliver those messages by Lemma 5.2.6, setting $leader_i$ to ℓ (Line 23) and resetting $timer_leader_i$ to $timeout_i(\ell)$ (Line 24). Observe that each time $timer_leader_i$ expires for process p_ℓ , p_i increases the value of $timeout_i(\ell)$ by 1 (Line 27). Thus, eventually $timeout_i(\ell)$ will have a value such that p_i always delivers a new LEADER message from p_ℓ before $timer_leader_i$ expires. Consequently, $\forall t, \forall p_i \in C_t$, eventually $timer_leader_i$ will not expire. \square



5.2 A Leader Election Algorithm for \mathcal{M}^*

Lemma 5.2.13. *In \mathcal{M}^- , $\forall t, \forall p_i \in C_t$, eventually and permanently $leader_i = \ell$.*

Proof. Follows directly from Lemma 5.2.12 if there is an ω -component C during the whole good period.

Without loss of generality, assume now that we have initially two disjoint ω -components $(C'_t)_i$ and $(C'_t)_j$ where $t' < t$ and each one has its own “temporary” leader p_{ℓ_i} and p_{ℓ_j} respectively. Let us assume that $(C'_t)_i$ and $(C'_t)_j$ joins into a unique ω -component C_t at a time t . Clearly, only one of the two leaders satisfies the conditions for being p_ℓ (highest number of elements in its *joins* set, with lowest identifier in case of ties) at t . Assume also without loss of generality that $p_\ell = p_{\ell_i}$. Following the same reasoning as before, eventually p_{ℓ_j} will deliver, by Lemma 5.2.6, a LEADER message from p_ℓ and will set its $leader_{\ell_j}$ variable to ℓ by Line 23, since the message satisfies the condition of Line 46. Moreover, eventually this will also happen to every process $p_i \in C_t$. By definition of $C_{\#}^s$, the number of ω -component joins in a good period is unknown but bounded, thus, this situation will happen a bounded number of times. Therefore, by Lemma 5.2.12, eventually and permanently $leader_i = \ell$.

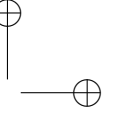
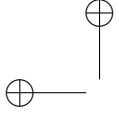
Regarding to process crashes, if a process p_h crashes during a good period, timers of processes p_i whose shortest path included p_h could expire. However, at expiration time by Line 27 the timer will be increased. Additionally, being C a ω -component and by Theorem 5.2.4, the LEADER message sent by p_ℓ will eventually reach to p_i by other way (Lemma 5.2.6). Consequently, in the worst case timers will converge to Δ . Since the diameter d is also unknown but finite (G_D^s), eventually $leader_i = \ell$ again permanently.

Concerning the new process p_i joining at ω -component C_t , observe that it will not disturb ℓ 's leadership, since $\ell \notin joins_i$ and eventually $i \in joins_\ell$ permanently. Therefore $|joins_i| < |joins_\ell|$ permanently. By Lemma 5.2.12, eventually $timer_leader_i$ will converge at p_i and $leader_i = \ell$ permanently.

Consequently, $\forall p_i \in C_t$, eventually and permanently $leader_i = \ell$. □

Lemma 5.2.14. *In \mathcal{M}^- , the algorithm of Figure 5.4 satisfies the \mathbf{EL}_{BC} property with process p_ℓ as the leader.*

Proof. Follows directly from Definition 4.4.2 and the definition of good period (see Section 5.1.2). □



5. EVENTUAL LEADER ELECTION IN DYNAMIC DISTRIBUTED SYSTEMS

Lemma 5.2.15. *In \mathcal{M}^- , the algorithm of Figure 5.4 satisfies the **EL-GJ** property of $\Delta^*\Omega$. Therefore, it also satisfies the **EL_{BC}-NI** and **EL_{BC}-ND** properties of $\Delta^*\Omega$ in \mathcal{M}^- .*

Proof. Follows directly from Lemmas 5.2.13 and 5.2.14. □

Lemma 5.2.16. *In \mathcal{M}^- , the algorithm of Figure 5.4 satisfies the **EL-GF** property of $\Delta^*\Omega$.*

Proof. Follows from Lemma 5.2.15, applied to each ω -component resulting from the fragmentation of a ω -component C due to an asynchronous episode previous to the beginning of the good period. □

Theorem 5.2.17. *In \mathcal{M}^- , the algorithm of Figure 5.4 implements $\Delta^*\Omega$.*

Proof. Follows directly from Lemmas 5.2.15 and 5.2.16. □

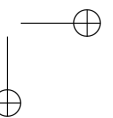
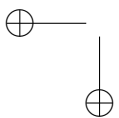
We will now prove that the algorithm of Figure 5.4 implements $\Delta^*\Omega$ in the system model \mathcal{M}^* , i.e., assuming s link failures. To do so, we assume an unknown bound in the number of messages lost k . However, we still assume that an unknown bound Δ in delivery time, but it only applies to the subset of messages that are not lost. Hence, the delay between two message receptions among any pair of processes $p_i, p_j \in C$ is eventually finite.

Lemma 5.2.18. *In \mathcal{M}^* , there is a time after which Lemma 5.2.6 holds.*

Proof. Observe that the number of messages lost is unknown but finite. Thus eventually the link will be reliable in terms of failures and Lemma 5.2.6 will hold for every message sent in the system. □

Lemma 5.2.19. *In \mathcal{M}^* , eventually and permanently $\forall p_i \in C_i : leader_i = \ell$.*

Proof. The main goal of the leadership criteria is to establish an order among processes of a ω -component and hold it during the election of a leader. Observe that the non-reception by the process p_i of a JOIN message could cause that another process p_j had a $|joins_j| > |joins_i|$ even p_i had joined to C before p_j . Thus, we cannot guarantee a order between processes in presence of message loses.



5.3 Evaluation

However, by Lemma 5.2.18 there exist a time t after which $\forall p, q \in C_t$ every message broadcast by p will be delivered by q and Theorem 5.2.4 holds. Observe that, independently from the with independence of the process and message history before t , after t all join messages will be delivered by every process $p_i \in C_t$, i.e., the differences between every $|join_i|$ will be maintained permanently. Therefore, from time t on, every Lemma of the \mathcal{M}^- model can be directly applied to \mathcal{M}^* . Consequently, by Lemma 5.2.13, eventually and permanently $\forall p_i \in C_t : leader_i = \ell$. \square

Lemma 5.2.20. *In \mathcal{M}^* , the algorithm of Figure 5.4 satisfies the **EL_{BC}** property with process p_ℓ as the leader.*

Proof. Follows directly from Lemma 5.2.19 and the definition of good period (see Section 5.1.2). \square

Lemma 5.2.21. *In \mathcal{M}^* , the algorithm of Figure 5.4 satisfies the **EL-GJ** property of $\Delta^*\Omega$. Therefore, it also satisfies the **EL_{BC}-NI** and **EL_{BC}-ND** properties of $\Delta^*\Omega$ in \mathcal{M}^* .*

Proof. Follows directly from Lemmas 5.2.19 and 5.2.20. \square

Lemma 5.2.22. *In \mathcal{M}^* , the algorithm of Figure 5.4 satisfies the **EL-GF** property of $\Delta^*\Omega$.*

Proof. Follows from Lemma 5.2.21, applied to each ω -component C_i resulting from the fragmentation of a ω -component C due to an asynchronous episode previous to the beginning of the good period. \square

Theorem 5.2.23. *In \mathcal{M}^* , the algorithm of Figure 5.4 implements $\Delta^*\Omega$.*

Proof. Follows directly from Lemmas 5.2.21 and 5.2.22. \square

5.3 Evaluation

In the previous we have proposed a weak system model \mathcal{M}^* for dynamic distributed systems and an eventual leader election algorithm for this model. In this section, we use the formalism introduced in Chapter 3 to compare the model \mathcal{M}^* to other system models for eventual leader election and illustrate the behavior of our algorithm comparatively to the others by mean of simulations.

5. EVENTUAL LEADER ELECTION IN DYNAMIC DISTRIBUTED SYSTEMS

5.3.1 Comparing \mathcal{M}^* to Other Models

The aim of this section is to compare the system model where our algorithm works, \mathcal{M}^* , to system models of other leader election algorithms proposed in the literature. More specifically, in this section we compare \mathcal{M}^* with the system models analyzed in Chapter 3. Recall that we have previously summarized in Table 3.1 these system models. The graphical tool introduced in Chapter 3 will facilitate the comparison. Basically, the bigger the area covered for a model in the chart, the weaker the model is.

Observe first in Figure 5.5 (left) the representation of the model defined for our leader election algorithm, i.e., $\mathcal{M}^* = \mathcal{M}(T^s, \mathcal{P}_F^s, \mathcal{C}_F^s, G_{\#}^s, G_{\Pi}^s, G_D^s, G_S^s)$, rating s in every dimension.

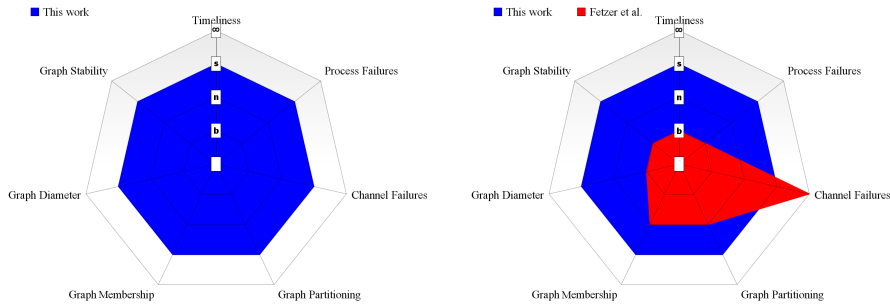


Figure 5.5: Figure on the left illustrates the graphical representation of the model \mathcal{M}^* . Figure on the right illustrates the comparison between model \mathcal{M}^* and the model proposed by Fetzer and Cristian [32].

Figures 5.5 (right), 5.6 and 5.7, illustrate how \mathcal{M}^* compares to the other system models we have analyzed. Overall, \mathcal{M}^* is weaker than the rest of models. Some models, as for example Ingram et al. [46] in Figure 5.6 (right), and Arantes et al. [6] in Figure 5.7 (right), rate some particular dimensions as infinite. However, recall that in the case of Ingram et al., they assume an oracle that provides processes information about the existing neighborhood (which is an agreement problem itself) which makes

5.3 Evaluation

the timeliness assumption unrealistic. On the other hand, in the case of Arantes et al., they make a very specific additional assumption to face the FLP impossibility, requiring a process in each neighborhood that never is the last process responding in a query-response communication attempt.

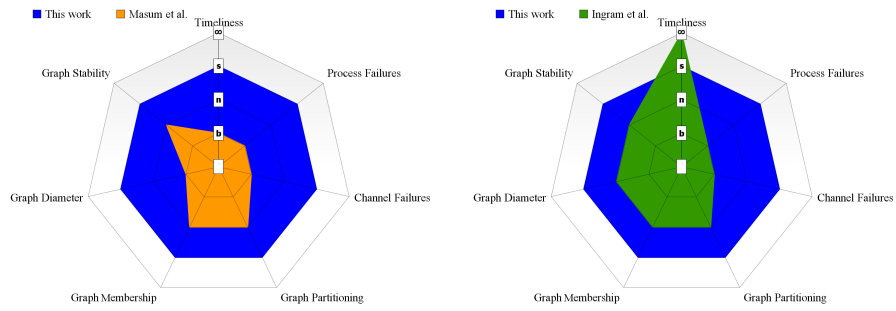


Figure 5.6: Figure on the left illustrates the comparison between model \mathcal{M}^* and the model proposed by Masum et al. [65]. Figure on the right illustrates the comparison between model \mathcal{M}^* and the model proposed by Ingram et al. [46].

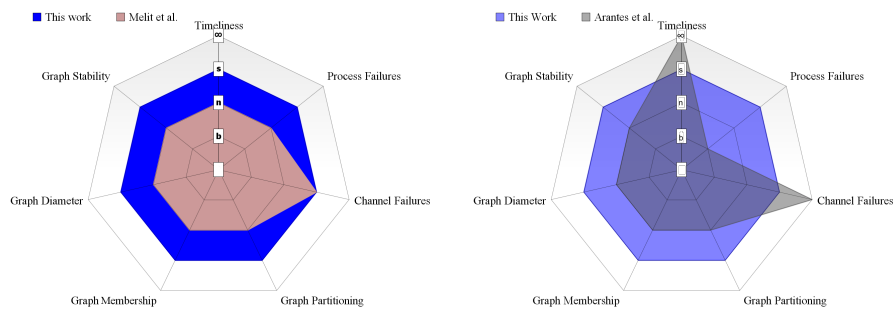


Figure 5.7: Figure on the left illustrates the comparison between model \mathcal{M}^* and the model proposed by Melit and Badache [67]. Figure on the right illustrates the comparison between model \mathcal{M}^* and the model proposed by Arantes et al. [6].

5. EVENTUAL LEADER ELECTION IN DYNAMIC DISTRIBUTED SYSTEMS

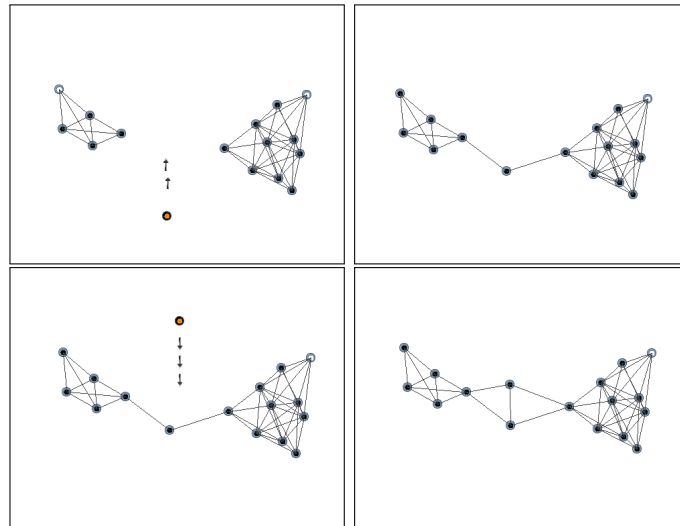


Figure 5.8: Screen-shot sequence of a simulation showing how the merge of two graphs leads to a unique leader and how new joins do not affect the leadership. Leaders are represented by white circles.

5.3.2 Simulation Examples

The proposed algorithm has been implemented and simulated using the JBotSim simulation tool [15]. Figure 5.8 shows some screen-shots of a simulation to illustrate an example of process mobility resulting in the join of two graphs and the management of the leaders according to the defined properties. JBotSim has been revealed very useful for a qualitative evaluation of our algorithm with respect to the others we have analyzed. In the rest of this section we use some simulated scenarios to illustrate how the other leader election algorithms cannot manage dynamic scenarios where our algorithm would converge to a stable leader.

Together our eventual leader election algorithm, we have implemented with JBotSim the eventual leader election algorithm proposed by Melit and Badache [67] and the eventual leader election algorithm proposed by Arantes et al. [6] to illustrate the fact

5.3 Evaluation

that both algorithms are more restrictive than ours in some highly dynamic scenarios.

Figure 5.9 illustrates how the continuous joining of processes results in an unstable leader due to the choice of a random value as leadership criteria in the algorithm of Melit and Badache. On the other hand, according to the movement pattern shown in Figure 5.10 we observed that as a consequence of periodic movement of processes, messages are lost and not a single leader is elected in the system. This is due to the fact that Melit and Badache do not consider temporal paths and use a classical one-shot broadcast primitive.

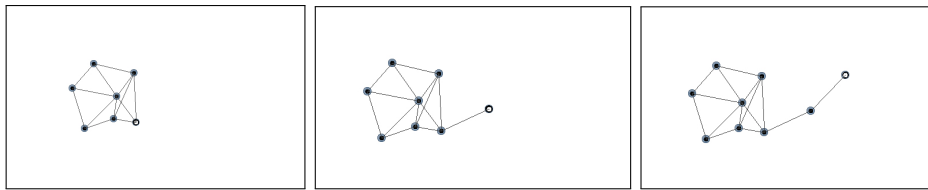


Figure 5.9: A continuous joining situation in Melit and Badache's algorithm. The leader is represented in white.

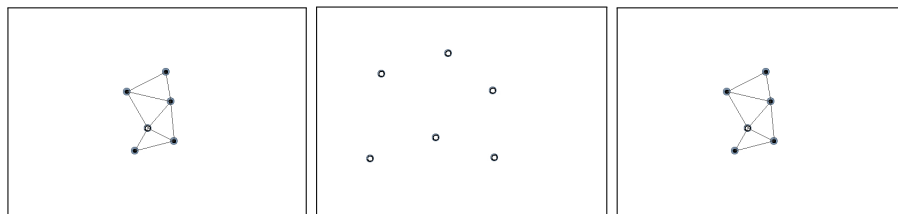


Figure 5.10: A periodic mobility pattern in Melit and Badache's algorithm. The leader is represented in white.

Regarding to Arantes et al.'s algorithm, in Figure 5.11 illustrates how the mobility of leader's neighbor makes the leadership unstable. Recall that Arantes et al.'s algorithm requires a stability of the leader's neighborhood. Consequently, this algorithm can not assume a fully dynamic connectivity assumption.

5. EVENTUAL LEADER ELECTION IN DYNAMIC DISTRIBUTED SYSTEMS

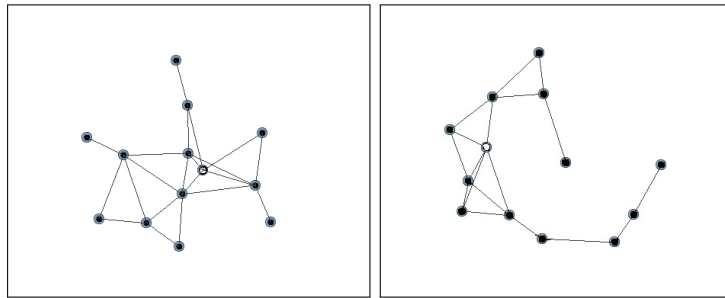
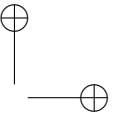
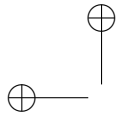


Figure 5.11: A varying neighborhood situation in Arante et al.'s algorithm. The leader is represented in white.

The previous toy examples show how our algorithm performs better than the two algorithms more similar to ours regarding leadership stability. In the present work we do not have boarded the large-scale experiments necessary to obtain a quantitative performance, which is proposed as a future work. However we have used the CMINDPA random graph generation [74] in order to have some performance figures of the structural complexity of our algorithm. We have generated a set of random graph situations with sizes 5, 10, 20 and 40 (number of nodes) as input topologies for our simulations (see Figure 5.12). The aim is to measure the time required for the algorithm to converge to a stable leader. Figure 5.13 shows an example of execution with 10 processes that illustrate how, as expected, our algorithm requires a longer time to converge to a leader than the algorithm by Melit and Badache, and the algorithm by Arantes et al. The overhead is due to the isolation detection mechanism used by our algorithm. This is the price to pay for tolerating a high degree of dynamicity.

On the other hand, we have obtained that the structural complexity of our algorithm with respect to the number of nodes is linear with a low coefficient. Although more experimentation should be carry out, this preliminary experiment show that the coefficient is in all the tested situations less than one.



5.4 Conclusions

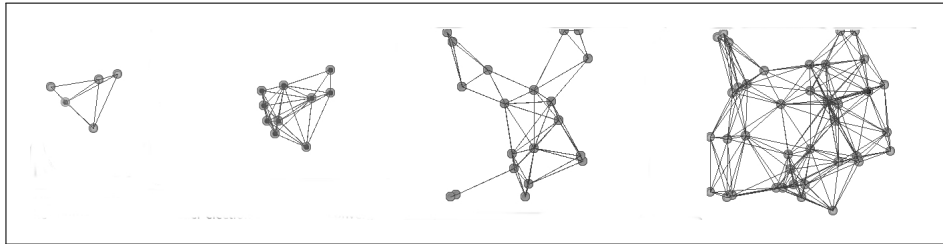


Figure 5.12: The set of initial graphs used for obtaining the convergence metrics of each algorithm studied.

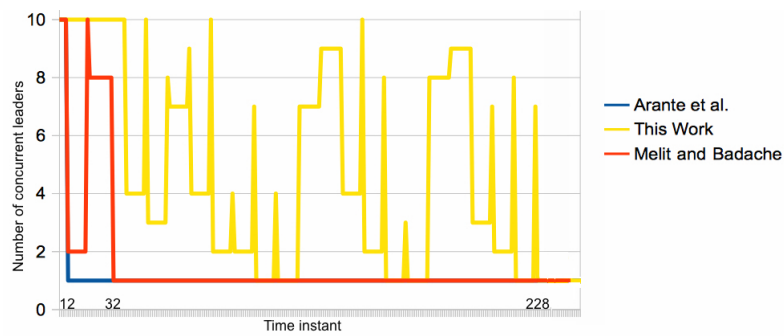
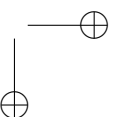
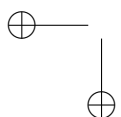
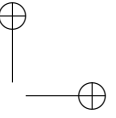
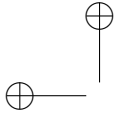


Figure 5.13: Time to converge a stable leader for the three analyzed algorithms. The simulated environment is composed by 10 processes with the same random topology.

5.4 Conclusions

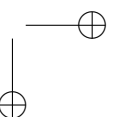
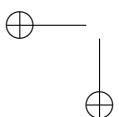
In this Chapter we have boarded leader election in a dynamic distributed system \mathcal{M}^* , using the proposed formalism, which is weaker than others we have found in the literature. A new failure detector class for \mathcal{M}^* , denoted $\Delta^*\Omega$, has been defined by means of a set of properties, namely **EL-GJ** and **EL-GF** for respectively Eventual Leadership in a Graph Joining and Eventual Leadership in a Graph Fragmentation situations. An algorithm for $\Delta^*\Omega$ has been proposed. The algorithm guarantees the election of a leader in sufficiently long stability periods for each one of the system partitions. Even

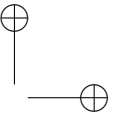
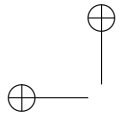




5. EVENTUAL LEADER ELECTION IN DYNAMIC DISTRIBUTED SYSTEMS

in the presence of unstable processes that crash/recover or move during the execution, the algorithm conveniently manages graph joining and partitioning situations.





CHAPTER

6

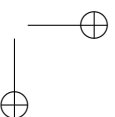
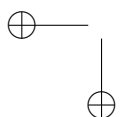
Conclusions

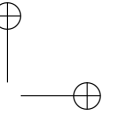
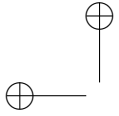
In this work we have studied how to solve agreement in dynamic distributed systems. We have combined two different perspectives, on the one hand the classical approaches of fault-tolerant distributed systems, and on the other hand the formalism of the Time-Varying Graph theory, in order to search for connectivity models that allow distributed agreement algorithms to terminate.

In this Section we present the contributions of the work and identify directions for future research.

6.1 Summary of Contributions

We believe that the present work has contributed to the state of the art of dynamic distributed systems in the following aspects.





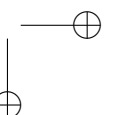
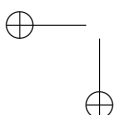
6. CONCLUSIONS

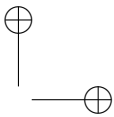
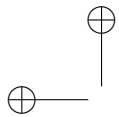
A formalism to categorise dynamic distributed systems

We have extended the formalism introduced by Baldoni et al. and provide a framework and a notation which has been revealed very useful to compare our system to other proposals in the literature. This contribution was published in the PRDC 2013 Conference [36] and a summary of it in the DISC 2013 Conference [41] as a brief announcement. Basically, the formalism defines a range of levels and a set of dimensions. Each dimension refer to an aspect of the system model (e.g., network diameter or process synchrony). A feature of the formalism is its uniformity: the same range of levels is defined for all dimensions of the system. With respect to Baldoni's formalism we have defined one additional level to represent the alternation of bounded and unbounded periods, inspired on the timed asynchronous model of Cristian and Fetzer. We have represented the new level as s and shown that level s is weaker than level n defined by Baldoni et al. In essence, contrary to level n , level s allows for unbounded behaviors after the global stabilization time of the classical partial synchrony models, provided that the unbounded behaviors do not last forever (in other words, a new period of bounded behavior will follow).

A connectivity timely model for dynamic distributed systems

We have studied how to introduce timeliness in evolving systems, formalized as Time-Varying Graphs, so that the resolution of agreement problems was possible in synchronous dynamic distributed systems. This contribution has been accepted for publication in the Proceedings of the EuroPar 2015 Conference as a regular paper [40]. On the basis of previous works by Casteigts et al., we have adopted the concept of journey or temporal path and have introduced the necessary timeliness (i.e., time bounds) to describe the specific assumptions that are required by an agreement algorithm to terminate and satisfy the formal properties. We have first proposed a general class, $\mathcal{TC}(\Delta)$,





6.1 Summary of Contributions

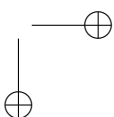
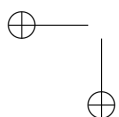
with a very abstract property on the temporal connectivity of the TVG to provide the necessary stability condition, namely, that the temporal diameter of a recurrent component in the TVG is bounded. We refer to such a component as a Δ -component. To approach the Consensus problem we have defined a TRB specification in terms of Δ -components, Δ -TRB. However, we have shown that Δ -TRB is not implementable in $\mathcal{TC}(\Delta)$ by message-passing without zero processing time assumptions. Henceforth, by introducing increasingly stronger connectivity assumptions, we have provided two *implementable* connectivity classes, namely $\mathcal{TC}'(\beta)$ and $\mathcal{TC}''(\alpha, \beta)$, as well as two respective implementations of Δ -TRB in these classes. Instead of using a general, system-wide bound, in $\mathcal{TC}''(\alpha, \beta)$, journeys have a bound, Γ , that can be calculated in terms of parameters that are linked to entities of real-world networks, namely, temporal bounds on the appearance of a link and on the stability of the link, as well as the diameter of the network.

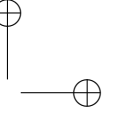
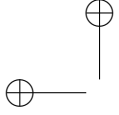
We have shown that consensus at Δ -component level is easily reduced to Δ -TRB. Nevertheless, extending the stability period of Δ -components is necessary to solve consensus at system level.

We have introduced an implementable class \mathcal{TC}^w , based on a new ω -component, which is weaker than $\mathcal{TC}'(\beta)$ since it does not assume a β bound, but only a minimum a priori known drift between the lifetime of every edge and its latency at any time t . We have proved that \mathcal{TC}^w is the weakest connectivity class allowing implementable Δ -TRB algorithms.

An Eventual leader election algorithm for a weak dynamic system model

As a specific agreement problem, we have boarded eventual leader election in a dynamic distributed system in a system model \mathcal{M}^* . This contribution was published in the Proceedings of the PRDC 2013 Conference [36]. A new failure detector class for





6. CONCLUSIONS

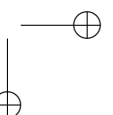
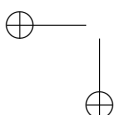
\mathcal{M}^* , denoted $\Delta^*\Omega$, has been defined by means of a set of properties, namely **EL-GJ** and **EL-GF**. The properties have been formulated in terms of Δ -components. An algorithm that solves eventual leader election in $\Delta^*\Omega$ has been proposed and proved. The algorithm guarantees the election of a leader in sufficiently long stability periods for each one of the system partitions, i.e., rating s in each of the dimensions of our categorization framework. Even in the presence of unstable processes that crash/recover or move during the execution, the algorithm conveniently manages graph joining and partitioning situations. Using the categorization framework, we have compared our system model \mathcal{M}^* to other models proposed in the literature for the eventual leader election problem. We have found that \mathcal{M}^* is weaker than other models.

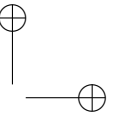
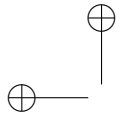
We have carry out simulations of our algorithm as well as of some of the reference eventual leader election algorithms. As expected, our proposal has higher stabilization times which is the price to pay for tolerating more dynamicity and hence for covering a wider range of behaviors and real-life scenarios.

6.2 Future Work

The present work can be extended in several directions.

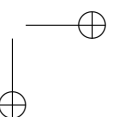
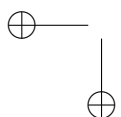
- Regarding the categorization of dynamic systems, the set of seven dimensions we have defined covers the general aspects considered in usual system models. However, some specific dimensions could be added. This is the case, for example, of clock synchronization. The current availability of resources for external synchronization (specifically GPS) provides in many scenarios an alternative to the classical internal clock synchronization that relies on timely channels. Another dimension is process homonymia. In the range of levels for homonymia, bounds would refer to the number of processes using the same identity. Finally, an adversary model can consider a range of bounds on Byzantine attacks.

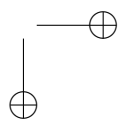
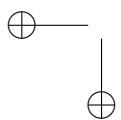
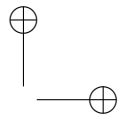
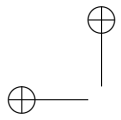




6.2 Future Work

- Temporal paths and components have revealed as powerful concepts to define connectivity models and a hierarchy of classes. Although we have identify a minimal connectivity class in this framework, the research in alternative approaches could bring results that would be interesting to compare to ours in terms of minimalism.
- Empirical evaluation could be enriched by considering real-world situations, for example those based in population dynamics, leading to initial graph distributions and mobility patterns which would provide a basis to drive simulations and compare the behavior and performance of the algorithms, as well as determining the structural complexity for our algorithm in different scenarios.
- A membership service could be built on top of the eventual leadership algorithm. Under additional stability assumptions the membership service would provide the conditions to solve consensus in (a part of) the dynamic system.







Bibliography

- [1] Marcos Kawazoe Aguilera, Wei Chen, and Sam Toueg. Failure detection and consensus in the crash-recovery model. *Distributed computing*, 13(2):99–125, 2000. 2, 19, 27, 28
- [2] Marcos Kawazoe Aguilera, Carole Delporte-Gallet, Hugues Fauconnier, and Sam Toueg. Stable leader election. In *Proceedings of the 15th International Conference on Distributed Computing, DISC 2001, Lisbon, Portugal*, volume 2180 of *Lecture Notes in Computer Science*, pages 108–122. Springer, 2001. 2, 17, 26
- [3] Marcos Kawazoe Aguilera, Carole Delporte-Gallet, Hugues Fauconnier, and Sam Toueg. On implementing omega in systems with weak reliability and synchrony assumptions. *Distributed Computing*, 21(4):285–314, 2008. 26, 97
- [4] Alaa N. Alslaity and Sanaa A. Alwidian. A K-Neighbor-based, Energy Aware Leader Election Algorithm (KELEA) for Mobile Ad hoc Networks. *International Journal of Computer Applications*, 59(19):38–43, December 2012. Published by Foundation of Computer Science, New York, USA. 43
- [5] Antonio Fernández Anta, Alessia Milani, Miguel A Mosteiro, and Shmuel Zaks. Opportunistic information dissemination in mobile ad-hoc networks: The profit of global synchrony. *Distributed Computing*, 25(4):279–296, 2012. 6, 30, 36, 61, 74



BIBLIOGRAPHY

- [6] Luciana Arantes, Fabíola Greve, Pierre Sens, and Véronique Simon. Eventual leader election in evolving mobile networks. In *Principles of Distributed Systems*, pages 23–37. Springer, 2013. v, vi, 3, 36, 41, 51, 55, 57, 114, 115, 116
- [7] Sergio Arévalo, Ernesto Jiménez, Mikel Larrea, and Luis Mengual. Communication-efficient and crash-quiescent Omega with unknown membership. *Information Processing Letters*, 111(4):194–199, 2011. 2, 28
- [8] James Aspnes. Fast deterministic consensus in a noisy environment. In *Proceedings of the nineteenth annual ACM symposium on Principles of distributed computing*, pages 299–308. ACM, 2000. 22
- [9] James Aspnes. Randomized protocols for asynchronous consensus. *Distributed Computing*, 16(2-3):165–175, 2003. 22
- [10] Roberto Baldoni, Marin Bertier, Michel Raynal, and Sara Tucci Piergiovanni. Looking for a definition of dynamic distributed systems. In *Proceedings of the 9th International Conference on Parallel Computing Technologies, PaCT 2007, Pereslavl-Zalessky, Russia*, volume 4671 of *Lecture Notes in Computer Science*, pages 1–14. Springer, 2007. ix, 5, 11, 38, 39, 45, 48, 49, 56
- [11] Michael Ben-Or. Another advantage of free choice (extended abstract): Completely asynchronous agreement protocols. In *Proceedings of the second annual ACM symposium on Principles of distributed computing*, pages 27–30. ACM, 1983. 22
- [12] Sandeep Bhadra and Afonso Ferreira. Complexity of connected components in evolving graphs and the computation of multicast trees in dynamic networks. In *Ad-Hoc, Mobile, and Wireless Networks*, pages 259–270. Springer, 2003. 33



BIBLIOGRAPHY

- [13] Martin Biely, Peter Robinson, and Ulrich Schmid. Agreement in directed dynamic networks. In *Structural Information and Communication Complexity - 19th International Colloquium, SIROCCO 2012, Reykjavik, Iceland*, volume 7355 of *Lecture Notes in Computer Science*, pages 73–84. Springer, 2012. 29, 35
- [14] Gabriel Bracha and Sam Toueg. Asynchronous consensus and broadcast protocols. *Journal of the ACM (JACM)*, 32(4):824–840, 1985. 22
- [15] Arnaud Casteigts. The JBotSim Library. *CoRR*, <http://arxiv.org/abs/1001.1435>, 2010. 116
- [16] Arnaud Casteigts, Paola Flocchini, Walter Quattrociocchi, and Nicola Santoro. Time-varying graphs and dynamic networks. *International Journal of Parallel, Emergent and Distributed Systems*, 27(5):387–408, 2012. vi, 3, 29, 30, 33, 60, 63, 89
- [17] Tushar Deepak Chandra, Vassos Hadzilacos, and Sam Toueg. On the impossibility of group membership. In *Proceedings of the 15th Annual ACM Symposium on Principles of Distributed Computing, PODC 1996, Philadelphia, Pennsylvania, USA*, pages 322–330. ACM, 1996. 13
- [18] Tushar Deepak Chandra, Vassos Hadzilacos, and Sam Toueg. The weakest failure detector for solving consensus. *Journal of the ACM*, 43(4):685–722, 1996. 2, 24, 93
- [19] Tushar Deepak Chandra and Sam Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2):225–267, 1996. 2, 16, 22
- [20] Bernadette Charron-Bost and André Schiper. The Heard-Of model: computing in distributed systems with benign faults. *Distributed Computing*, 22(1):49–71, 2009. 2, 92



BIBLIOGRAPHY

- [21] Wei Chen, Sam Toueg, and Marcos Kawazoe Aguilera. On the quality of service of failure detectors. *Computers, IEEE Transactions on*, 51(5):561–580, 2002. 26
- [22] Roberto Cortiñas, Felix C. Freiling, Marjan Ghajar-Azadanlou, Alberto Lafuente, Mikel Larrea, Lucia Draque Penso, and Iratxe Soraluze Arriola. Secure failure detection and Consensus in TrustedPals. *IEEE Transactions on Dependable and Secure Computing*, 9(4):610–625, 2012. 2, 28
- [23] Flaviu Cristian. Reaching agreement on processor-group membership in synchronous distributed systems. *Distributed Computing*, 4(4):175–187, 1991. 2
- [24] Flaviu Cristian and Christof Fetzer. The timed asynchronous distributed system model. *IEEE Transactions on Parallel and Distributed Systems*, 10(6):642–657, 1999. 2, 5, 17, 46, 47, 56, 92, 97
- [25] Abdelouahid Derhab and Nadjib Badache. A Self-Stabilizing Leader Election Algorithm in Highly Dynamic Ad Hoc Mobile Networks. *IEEE Transactions on Parallel and Distributed Systems*, 19(7):926–939, 2008. 42
- [26] Edsger W. Dijkstra and Carel S. Scholten. Termination detection for diffusing computations. *Inf. Process. Lett.*, 11(1):1–4, 1980. 43
- [27] Danny Dolev, Cynthia Dwork, and Larry Stockmeyer. On the minimal synchronism needed for distributed consensus. *J. ACM*, 34(1):77–97, January 1987. 2, 15, 22
- [28] Cynthia Dwork, Nancy A. Lynch, and Larry J. Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM*, 35(2):288–323, 1988. 2, 16, 22



BIBLIOGRAPHY

- [29] Paul Feldman and Silvio Micali. Optimal algorithms for byzantine agreement. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 148–161. ACM, 1988. 13, 64, 80
- [30] Afonso Ferreira. Building a reference combinatorial model for manets. *Network, IEEE*, 18(5):24–29, 2004. 3
- [31] Christof Fetzer. The message classification model. In *Proceedings of the seventeenth annual ACM symposium on Principles of distributed computing*, PODC '98, pages 153–162, New York, NY, USA, 1998. ACM. 17
- [32] Christof Fetzer and Flaviu Cristian. A Highly Available Local Leader Election Service. *IEEE Transactions on Software Engineering*, 25(5):603–618, 1999. v, vi, 2, 41, 51, 52, 53, 114
- [33] Christof Fetzer, Ulrich Schmid, and Martin Susskraut. On the possibility of consensus in asynchronous systems with finite average response times. In *Distributed Computing Systems, 2005. ICDCS 2005. Proceedings. 25th IEEE International Conference on*, pages 271–280. IEEE, 2005. 17
- [34] Michael J. Fischer, Nancy A. Lynch, and Mike Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, 1985. 2, 10
- [35] Paola Flocchini, Bernard Mans, and Nicola Santoro. Exploration of periodically varying graphs. In *Algorithms and Computation*, pages 534–543. Springer, 2009. 3
- [36] Carlos Gomez-Calzado, Alberto Lafuente, Mikel Larrea, and Michel Raynal. Fault-tolerant leader election in mobile dynamic distributed systems. In *Depend-*



BIBLIOGRAPHY

- able Computing (PRDC), 2013 IEEE 19th Pacific Rim International Symposium on*, pages 78–87, Dec 2013. 5, 6, 48, 50, 93, 94, 95, 96, 122, 123
- [37] Carlos Gómez-Calzado, Mikel Larrea, Iratxe Soraluze, Alberto Lafuente, and Roberto Cortinas. An evaluation of efficient leader election algorithms for crash-recovery systems. In *Parallel, Distributed and Network-Based Processing (PDP), 2013 21st Euromicro International Conference on*, pages 180–188. IEEE, 2013. 28
- [38] Fabíola Greve, Pierre Sens, Luciana Arantes, and Véronique Simon. A failure detector for wireless networks with unknown membership. In *Euro-Par 2011 Parallel Processing*, pages 27–38. Springer, 2011. 41
- [39] Rachid Guerraoui and Michel Raynal. The information structure of indulgent consensus. *IEEE Transactions on Computers*, 53(4):453–466, 2004. 2
- [40] Carlos Gómez-Calzado, Arnaud Casteigts, Alberto Lafuente, and Mikel Larrea. A connectivity model for agreement in dynamic systems. In *Euro-Par 2015 - Parallel Processing, 21th International Euro-Par Conference, Vienna, Austria, August 24-28, 2015, Proceedings (accepted)*, 2015. 4, 5, 122
- [41] Carlos Gómez-Calzado, Alberto Lafuente, Mikel Larrea, and Michel Raynal. Brief Announcement: Revisiting Dynamic Distributed Systems. In *Proceedings of the 27th International Symposium on Distributed Computing, DISC 2013, Jerusalem, Israel, October 14-18, 2013*. 5, 122
- [42] Zygmunt J Haas, Marc R Pearlman, and Prince Samar. The zone routing protocol (zrp) for ad hoc networks. 2002. 43
- [43] Med Amine Haddar, A Hadj Kacem, Yves Métivier, Mohamed Mosbah, and Mohamed Jmaiel. Electing a leader in the local computation model using mobile



BIBLIOGRAPHY

- agents. In *Computer Systems and Applications, 2008. AICCSA 2008. IEEE/ACS International Conference on*, pages 473–480. IEEE, 2008. 42
- [44] Michel Hurfin, Achour Mostefaoui, and Michel Raynal. Consensus in asynchronous systems where processes can crash and recover. In *Reliable Distributed Systems, 1998. Proceedings. Seventeenth IEEE Symposium on*, pages 280–286. IEEE, 1998. 19
- [45] Michel Hurfin and Michel Raynal. A simple and fast asynchronous consensus protocol based on a weak failure detector. *Distributed Computing*, 12(4):209–223, 1999. 24
- [46] Rebecca Ingram, Patrick Shields, Jennifer E. Walter, and Jennifer L. Welch. An asynchronous leader election algorithm for dynamic networks. In *Proceedings of the 23rd IEEE International Symposium on Parallel and Distributed Processing, IPDPS 2009, Rome, Italy, May 23-29*, pages 1–12. IEEE Computer Society, 2009. v, vi, 42, 51, 53, 55, 114, 115
- [47] Ernesto Jiménez, Sergio Arévalo, and Antonio Fernández. Implementing unreliable failure detectors with unknown membership. *Information Processing Letters*, 100(2):60–63, 2006. 2, 28
- [48] David Kempe, Jon Kleinberg, and Amit Kumar. Connectivity and inference problems for temporal networks. In *Proceedings of the thirty-second annual ACM symposium on Theory of computing*, pages 504–513. ACM, 2000. 3
- [49] Fabian Kuhn, Nancy A. Lynch, and Rotem Oshman. Distributed computation in dynamic networks. In *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*, pages 513–522. ACM, 2010. 3, 35



BIBLIOGRAPHY

- [50] Alberto Lafuente, Mikel Larrea, Iratxe Soraluze, and Roberto Cortiñas. Communication-optimal eventually perfect failure detection in partially synchronous systems. *J. Comput. Syst. Sci.*, 81(2):383–397, 2015. 26
- [51] Leslie Lamport. The part-time parliament. *ACM Transactions on Computer Systems (TOCS)*, 16(2):133–169, 1998. 2, 25
- [52] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4(3):382–401, 1982. 18
- [53] Mikel Larrea, Sergio Arévalo, and Antonio Fernandez. Efficient algorithms to implement unreliable failure detectors in partially synchronous systems. In *Distributed Computing*, pages 34–49. Springer, 1999. 26
- [54] Mikel Larrea, Antonio Fernández, and Sergio Arévalo. On the implementation of unreliable failure detectors in partially synchronous systems. *Computers, IEEE Transactions on*, 53(7):815–828, 2004. 26
- [55] Mikel Larrea, Antonio Fernández, and Sergio Arévalo. Eventually consistent failure detectors. *Journal of Parallel and Distributed Computing*, 65(3):361–373, 2005. 2
- [56] Mikel Larrea, Alberto Lafuente, Iratxe Soraluze, Roberto Cortiñas, and Joachim Wieland. Designing efficient algorithms for the eventually perfect failure detector class. *JSW*, 2(4):1–11, 2007. 26
- [57] Mikel Larrea and Cristian Martín. Implementing the Omega Failure Detector in the Crash-Recovery Model with partial Connectivity and/or Synchrony. In *DEXA Workshops, 17th International Workshop on Database and Expert Systems*



BIBLIOGRAPHY

- Applications (DEXA 2006), 4-8 September 2006, Krakow, Poland*, pages 400–405, 2006. 28, 91
- [58] Mikel Larrea, Cristian Martín, and Iratxe Soraluze Arriola. Communication-efficient leader election in crash-recovery systems. *Journal of Systems and Software*, 84(12):2186–2195, 2011. 2, 28
- [59] Mikel Larrea, Michel Raynal, Iratxe Soraluze, and Roberto Cortiñas. Specifying and implementing an eventual leader service for dynamic systems. *International Journal of Web and Grid Services*, 8(3):204–224, 2012. 2, 3, 6, 28, 40, 91, 92, 93
- [60] Gérard Le Lann and Ulrich Schmid. How to implement a timer-free perfect failure detector in partially synchronous systems. *Department of Automation, Technische Universität Wien, Tech. Rep*, 183:1–127, 2003. 17
- [61] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graph evolution: Densification and shrinking diameters. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 1(1):2, 2007. 3
- [62] Navneet Malpani, Jennifer L. Welch, and Nitin H. Vaidya. Leader election algorithms for mobile ad hoc networks. In *Proceedings of the 4th International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications, DIAL-M 2000, Boston, Massachusetts, USA, August 11*, pages 96–103. ACM, 2000. 42
- [63] Cristian Martín, Mikel Larrea, and Ernesto Jiménez. On the implementation of the omega failure detector in the crash-recovery failure model. In *Proceedings of the 2nd International Conference on Availability, Reliability and Security, ARES 2007, April 10-13 2007, Vienna, Austria*, pages 975–982, 2007. 28, 91



BIBLIOGRAPHY

- [64] Cristian Martín, Mikel Larrea, and Ernesto Jiménez. Implementing the omega failure detector in the crash-recovery failure model. *Journal of Computer and System Sciences*, 75(3):178–189, 2009. 28, 91
- [65] Salahuddin Mohammad Masum, Amin Ahsan Ali, and Mohammad Touhid youl Islam Bhuiyan. Asynchronous Leader Election in Mobile Ad Hoc Networks. In *Proceedings of the 20th International Conference on Advanced Information Networking and Applications, AINA 2006, Vienna, Austria, 18-20 April*, pages 827–831. IEEE Computer Society, 2006. v, vi, 43, 51, 52, 54, 115
- [66] Leila Melit and Nadjib Badache. An energy efficient leader election algorithm for mobile ad hoc networks. In *Programming and Systems (ISPS), 2011 10th International Symposium on*, pages 54 –59, april 2011. 41
- [67] Leila Melit and Nadjib Badache. An Ω -Based Leader Election Algorithm for Mobile Ad Hoc Networks. In *Proceedings of the 4th International Conference on Networked Digital Technologies, NDT 2012, Dubai, UAE, April 24-26*, volume 293, pages 483–490. Springer, 2012. v, vi, 3, 41, 51, 54, 56, 115, 116
- [68] Othon Michail, Ioannis Chatzigiannakis, and Paul G Spirakis. Causality, influence, and computation in possibly disconnected synchronous dynamic networks. *Journal of Parallel and Distributed Computing*, 74(1):2016–2026, 2014. 37
- [69] Henrique Moniz, Nuno F. Neves, and Miguel Correia. Byzantine fault-tolerant consensus in wireless ad hoc networks. *Mobile Computing, IEEE Transactions on*, 12(12):2441–2454, Dec 2013. 22
- [70] Henrique Moniz, Nuno Ferreira Neves, Miguel Correia, and Paulo Veríssimo. Randomization can be a healer: Consensus with dynamic omission failures. In *Proceedings of the 23rd International Conference on Distributed Computing, DISC’09*, pages 63–77, Berlin, Heidelberg, 2009. Springer-Verlag. 22



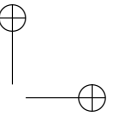
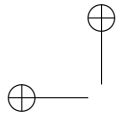
BIBLIOGRAPHY

- [71] Achour Mostefaoui, Eric Mourgaya, and Michel Raynal. Asynchronous implementation of failure detectors. In *DSN*, pages 351–360. Citeseer, 2003. 17
- [72] Achour Mostéfaoui, Eric Mourgaya, Michel Raynal, and Corentin Travers. A time-free assumption to implement eventual leadership. *Parallel Processing Letters*, 16(02):189–207, 2006. 17
- [73] Achour Mostéfaoui and Michel Raynal. Leader-based consensus. *Parallel Processing Letters*, 11(01):95–107, 2001. 2, 25
- [74] Furuzan Atay Onat, Ivan Stojmenovic, and Halim Yanikomeroglu. Generating random graphs for the simulation of wireless ad hoc, actuator, sensor, and internet networks. *Pervasive and Mobile Computing*, 4(5):597 – 615, 2008. 118
- [75] Diego Ongaro and John Ousterhout. In search of an understandable consensus algorithm. In *Proc. USENIX Annual Technical Conference*, pages 305–320, 2014. 2
- [76] Vincent Douglas Park and M Scott Corson. A highly adaptive distributed routing algorithm for mobile wireless networks. In *INFOCOM'97. Sixteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, volume 3, pages 1405–1413. IEEE, 1997. 42
- [77] Pradeep Parvathipuram, Vijay Kumar, and Gi-Chul Yang. An Efficient Leader Election Algorithm for Mobile Ad Hoc Networks. In *Proceedings of the First International Conference on Distributed Computing and Internet Technology, ICD-CIT 2004, Bhubaneswar, India, December 22-24*, volume 3347 of *Lecture Notes in Computer Science*, pages 32–41. Springer, 2004. 43
- [78] Sara Tucci Piergiovanni and Roberto Baldoni. Eventual leader election in infinite arrival message-passing system model with bounded concurrency. In *Proceedings*



BIBLIOGRAPHY

- of the 8th European Dependable Computing Conference, EDCC 2010, Valencia, Spain*, pages 127–134. IEEE Computer Society, 2010. 2, 28, 41
- [79] Michel Raynal, Julien Stainer, Jiannong Cao, and Weigang Wu. A simple broadcast algorithm for recurrent dynamic systems. In *Advanced Information Networking and Applications (AINA), 2014 IEEE 28th International Conference on*, pages 933–939, May 2014. 37, 68
- [80] Srikanth Sastry and Scott M. Pike. Eventually perfect failure detectors using ADD channels. In *Proceedings of the 5th International Symposium on Parallel and Distributed Processing and Applications, ISPA 2007, Niagara Falls, Canada*, volume 4742 of *Lecture Notes in Computer Science*, pages 483–496. Springer, 2007. 2, 17
- [81] Mahadev Satyanarayanan. Pervasive computing: vision and challenges. *IEEE Personal Communications*, 8(4):10–17, 2001. 3
- [82] André Schiper. Early consensus in an asynchronous system with a weak failure detector. *Distributed Computing*, 10(3):149–157, 1997. 24
- [83] John Tang, Mirco Musolesi, Cecilia Mascolo, and Vito Latora. Characterising temporal distance and reachability in mobile and online social networks. *ACM SIGCOMM Computer Communication Review*, 40(1):118–124, 2010. 3
- [84] Sudarshan Vasudevan, James F. Kurose, and Donald F. Towsley. Design and Analysis of a Leader Election Algorithm for Mobile Ad Hoc Networks. In *Proceedings of the 12th IEEE International Conference on Network Protocols, ICNP 2004, Berlin, Germany, 5-8 October*, pages 350–360. IEEE Computer Society, 2004. 43



BIBLIOGRAPHY

- [85] Paulo Verissimo, Antonio Casimiro, and Christof Fetzer. The timely computing base: Timely actions in the presence of uncertain timeliness. In *Dependable Systems and Networks, 2000. DSN 2000. Proceedings International Conference on*, pages 533–542. IEEE, 2000. 2, 17
- [86] Paul M.B. Vitányi. Distributed elections in an archimedean ring of processors. In *Proceedings of the sixteenth annual ACM symposium on Theory of computing, STOC '84*, pages 542–547, New York, NY, USA, 1984. ACM. 2, 16

