Proteo: A Self-adaptive Software Architecture to Support Quality Attributes in Ubiquitous Systems

Gabriel Guerrero-Contreras

Supervisors: Dr José Luis Garrido
Dr María José Rodríguez-Fórtiz

Department of Software Engineering
University of Granada

PhD Program in Information and Communication Technologies

June 2018

Editor: Universidad de Granada. Tesis Doctorales
Autor: Gabriel José Guerrero Contreras
ISBN: 978-84-1306-009-5
URI: http://hdl.handle.net/10481/53829

Gabriel Guerrero-Contreras. *Proteo: A Self-adaptive Software Architecture to Support Quality Attributes in Ubiquitous Systems.*

Supervisors
Dr José Luis Garrido
Dr María José Rodríguez-Fórtiz

Software Engineering Department
University of Granada

June 2018

gjguerrero@ugr.es
www.ugr.es/~gjguerrero

Resumen

Desde que los Sistemas Distribuidos evolucionaron a Sistemas Móviles, la adaptación, específicamente, la auto-adaptación ha sido un requisito fundamental para abordar satisfactoria y eficientemente el desarrollo de sistemas software. Hoy en día, la auto-adaptación es un requisito recurrente en la mayoría de los entornos de computación existentes, primero en Computación Ubicua, posteriormente en el Internet de las Cosas (IoT) y la Computación en la Nube Móvil (MCC). Sin embargo, a pesar de la importancia de este requisito, todavía existe la necesidad de proponer un diseño arquitectónico software para abordarlo de forma apropiada.

Mientras que la Arquitectura Orientada a Servicios (SOA) enfatiza la importancia de la autonomía de servicio, la existencia de arquitecturas dinámicas reconfigurables en tiempo de ejecución es todavía un desafío. Con este fin, SOA tiene que ser combinada con métodos y técnicas del campo de la Computación Autonómica, así como de otros estilos y variantes arquitectónicos, como la Arquitectura Dirigida por Eventos (EDA) y Arquitecturas de Microservicios.

En este contexto, este trabajo presenta una arquitectura software auto-adaptativa, denominada Proteo. El principal objetivo de Proteo es proporcionar una solución sensible al contexto para soportar atributos de calidad relevantes (específicamente, disponibilidad y fiabilidad) en sistemas dinámicos, aplicando técnicas de replicación y auto-configuración.

La arquitectura Proteo está compuesta de tres componentes principales, los cuales están replicados en los nodos que componen la red del sistema distribuido: (1) *Subsistema de Monitorización*, (2) *Servicio de Gestión del Contexto*, y (3) *Servicio de Gestión de Réplicas*. Las responsabilidades de estos componentes están estrechamente relacionadas con las fases del bucle autonómico MAPE-K, de la Computación Autonómica: el *Subsistema de Monitorización* monitoriza y percibe la información del contexto relacionada con el nodo en el cual se encuentra desplegado; (2) el *Servicio de Gestión del Contexto*, además de procesar la información recibida por el *Subsistema de Monitorización*, es responsable de analizarla y decidir cuándo es necesaria una nueva adaptación en el sistema; y el *Servicio de Gestión de Réplicas*, el cual es el responsable de llegar a un acuerdo con el resto de los *Servicios de Gestión de Réplicas* desplegados en el sistema para establecer cuáles de las réplicas existentes será activada. Esta coordinación se lleva a cabo utilizando un algoritmo distribuido de elección, dicha elección

se basa en una puntuación dinámica obtenida a través de una función de utilidad en tiempo de ejecución. Dicha función de utilidad indica cómo de adecuado es un nodo para alojar una réplica activa de un servicio.

En este trabajo, se presenta una revisión de los enfoques actualmente existentes para la provisión de servicios o datos (recursos) en entornos dinámicos. Adicionalmente, Proteo se ha modelado utilizando SysML. Dicho modelo está dividido en: (1) *Modelo del Dominio Operacional*, (2) *Modelo de Información*, (3) *Modelo Estructural*, y (4) *Modelo de Comportamiento*. Estos cuatro modelos abarcan la definición detallada de los componentes de la arquitectura Proteo, sus relaciones y su comportamiento durante la operación del sistema.

Algunas características de Proteo que se pueden destacar son: (1) es reactivo a los cambios del contexto, como por ejemplo las desconexiones de los nodos; (2) usa información local al nodo a través de un enfoque cross-layer para monitorizar la red; (3) utiliza un método heurístico, propuesto en este trabajo, para determinar el nodo más céntrico en la topología de la red basándose en la información de la tabla de ruta del nodo; (4) utiliza una función de utilidad para evaluar cómo de adecuado es un nodo para alojar una réplica del servicio activa, lo que facilita adaptar el criterio de elección y tener en consideración los recursos de los nodos, como la energía, proporcionando así una solución consciente del consumo de recursos.

Con respecto a los algoritmos de elección, se han propuesto en este trabajo dos nuevos algoritmos: *Consenso* y *Votación*. Estos algoritmos de elección están diseñados para operar en entornos móviles con topologías de red altamente dinámicas y bajo canales de comunicación no fiables. Adicionalmente, entre los enfoques existentes, los algoritmos de elección de Bully, Kordafshair y Vasudevan se han seleccionado e incorporado a Proteo, con el objetivo de comparar y valorar el rendimiento de los dos nuevos algoritmos propuestos con estos algoritmos ya existentes.

Para validar y evaluar la propuesta, se ha utilizado el simulador de redes ns-3. A este respecto, Proteo ha sido diseñado e implementado como un nuevo módulo en ns-3, lo cual ha permitido analizar la arquitectura auto-adaptativa en términos de disponibilidad de servicios, fiabilidad del algoritmo de elección, utilización de mensajes de coordinación y vida útil de la red (consumo de energía). El uso de herramientas avanzadas de simulación nos permite entender mejor el comportamiento de la arquitectura propuesta mediante el manejo y evaluación de modelos dinámicos.

Finalmente, se introduce brevemente el trabajo que ya está en progreso para continuar desarrollando y extendiendo la arquitectura Proteo. Actualmente existen dos líneas de investigación principales. Por una parte, para abordar la sincronización de las réplicas y la interoperabilidad del sistema se ha diseñado y propuesto una plataforma de servicios. Ésta está diseñada para proporcionar, desde la fase de diseño, una base común para la gestión consistente de recursos en sistemas ubicuos colaborativos. Por otra parte, para abordar la escalabilidad de la propuesta, se ha llevado a cabo un estudio para conocer el número de nodos que Proteo puede gestionar, utilizando el algoritmo de Consenso, sin degradar su rendimiento. Esto es esencial para poder aplicar Proteo en entornos de IoT, los cuales están pensados para soportar un alto número de dispositivos heterogéneos conectados. Además, dicho estudio pretende determinar si la fiabilidad de TCP podría resultar en un mejor funcionamiento del sistema, a pesar de su menor tiempo de respuesta y mayor consumo de ancho de banda en comparación con UDP.

Abstract

Since Distributed Systems evolved to Mobile Systems, adaptation, and more specifically self-adaptation, has been a fundamental requirement to successfully and efficiently address the development of software systems. Nowadays, selft-adaptation is a recurrent requirement in most of the existing computation environments, first in Ubiquitous Computing, and later in the Internet of Things (IoT) and Mobile Cloud Computing (MCC). However, despite the importance of this requirement, still, there is a need to propose a suitable architectural approach to address it.

Whereas the Service Oriented Architecture (SOA) emphasizes the importance of service autonomy, the provision of dynamically reconfigurable runtime architectures is still a challenge. To this end, SOA has to be combined with methods and techniques from Autonomic Computing field, as well as other complementary architectural styles and variants, such as Event-Driven Architecture (EDA) and Microservices.

In this context, this work presents a self-adaptive software architecture, named Proteo. The primary objective of Proteo is to provide a context and resource-aware solution to support relevant quality attributes (especially availability and reliability) in dynamic systems by applying replication and self-configuration techniques.

Proteo architecture is composed of three main components, which are replicated on the nodes that compose the distributed system network: (1) *Monitoring Subsystem*, (2) *Context Manager Service*, and (3) *Replica Manager Service*. Their responsibilities are closely related to the phases of MAPE-K autonomic loop of Autonomic Computing: the *Monitoring Subsystem* senses the context information in relation to the node in which is deployed; the *Context Manager Service*, in addition to process this information, is also responsible for analysing it to decide when a new system adaptation is necessary; and the *Replica Manager Service* will be responsible for coming to an agreement with the rest of the *Replica Manager Services* deployed in the system to establish what replica will be activated. This coordination is performed by using a distributed host election algorithm, and the election is based on a dynamic score obtained through a utility function at run-time. This utility function indicates how suitable is a node to host a service replica.

Throughout this work, a review of the existing proposals on dynamic service or data provisioning in dynamic environments will be presented. Additionally, Proteo is modelled using SysML. This model is divided into: (1) *Operational Domain Model*, (2) *Information Model*, (3) *Structural Model*, and (4) *Behavioural Model*. These four models encompass the detailed definition of the components of Proteo architecture, their relationships, and their behaviour during the system operation.

Some notable features of Proteo are: (1) reactivity to context changes, such as node disconnections; (2) use of local knowledge to the node through a cross-layer approach to monitoring the network; (3) a heuristic method has been proposed to determine the most centric node in the network topology, on the basis of the information of its routing table; (4) to use a utility function to evaluate the feasibility of the nodes to host an active service replica, which facilitates to adapt the election criteria and to take into consideration the resources of the nodes, as energy, providing a resource-aware solution.

Regarding the election algorithms, two new algorithms are proposed: *Consensus* and *Voting*. These election algorithms are devised to operate in mobile environments with highly dynamic network topologies and under unreliable communication channels. In addition, between the existing approaches, the election algorithms of Bully, Kordafshari, and Vasudevan have been selected and incorporated to Proteo to compare the performance of the new two proposed election algorithms with them.

To validate and evaluate the proposal, the ns-3 network simulator has been used. To this regard, Proteo has been designed and developed as a new module in ns-3, which has allowed to analyse the self-adaptive architecture in terms of service availability, election algorithm reliability, coordination messages usage and network lifetime. The use of advanced simulation tools allows us to better understand the behaviour of the approach by managing and evaluating the dynamic models that it embraces.

Finally, some of the work already in progress to continue developing Proteo architecture is introduced. Nowadays, two main lines of research are being followed. Firstly, to address synchronization and system interoperability, for which a service platform has been proposed. It intends to provide a common basis for the consistent management of shared resources, from the software design stage, in ubiquitous collaborative systems. And secondly, to address scalability, for which a study has been performed to know the number of nodes that Proteo can manage using the Consensus election algorithm without degrading its performance. This is essential for applying Proteo in IoT environments, which are devised to support a large number of heterogeneous devices. Additionally, this study aims to state if the reliability of TCP could

result in an enhanced system operation, despite its high latency and higher consumption of bandwidth in comparison to UDP.

Table of contents

List of figures

List of tables

Part I

Introduction, Foundations and Related Work

Chapter 1

Introduction

Chapter Abstract

Nowadays, Mobile and Ubiquitous Computing are gaining more and more importance. Current computation systems provide an infrastructure where data storage and processing could happen outside of the mobile node. Specifically, there is a significant interest in the use of the services that are transparently provided by nearby mobile nodes in the form of a distributed resource pooling. This kind of systems is useful in application domains such as emergencies, education, and tourism. However, these systems are commonly based on dynamic network topologies, in which disconnections and network partitions can occur frequently, and thus the availability and other quality attributes of the system are usually compromised. In this context, techniques and methods from Autonomic Computing can be applied to build quality service models taking into account changes in the context. The main objective of this thesis is to design and implement a proposal based on a self-adaptive approach to improve the quality and the applicability of Software Oriented Architectures in Mobile and Ubiquitous Computing, addressing the new challenges posed by these dynamic systems. Finally, the list of publications derived from this work is included.

Chapter Contents

1.1 Introduction

In 1988 Weiser introduced the **Ubiquitous Computing** term, subsequently developed in [167], referring to:

> a scenario in which computers become more numerous and fade into the background, providing information to human users and embedding intelligence and computing capabilities in seemingly everyday objects. [122]

The introduction of the Ubiquitous Computing, not as a technology but as a new computation paradigm, brings *The Age of Calm Technology* [168], in which, technology vanishes into the background of our daily lives, providing an unconscious human-computer interaction.

This pioneering view that was ahead of its times has not been possible until now, when, thanks to advances in Information and Communication Technology (ICT), there are devices powerful enough to create these new computing systems.

Ubiquitous Computing, considered as an evolution of **Mobile Computing** [143], represents the basis for multiple ICT areas, such as **Ambient Intelligence** (AmI) [40], **Mobile Cloud Computing** (MCC) [52] or **Internet of Things** (IoT) [11]. Moreover, it is closely related to other areas, such as **Artificial Intelligence** (IA) [137] or **Context-Aware Computing** [46].

These new approaches and computational paradigms are expected to be useful in a diversity of **application domains**, where the common factors are **mobility** and **many-to-many communication under collaborative groups** [29]:

- **Advanced Intelligent Transportation Systems** (ITS) intend to provide a set of technological solutions to improve the security and management of transport network [32]. The Vehicular Ad-hoc NETworks (VANETs) [70] arise from a multi-hop ad-hoc network paradigm to support a wide range of applications in this area, from collision avoidance to traffic services. In this kind of network both vehicle-to-vehicle (V2V) and vehicle-to-roadside (V2R) communications can be found. The union of V2V and V2R provides a better network coverage and performance. However, the high mobility of vehicles and the different traffic patterns are challenging for designers and developers. In [152] a prototype system is presented. It is based on a V2V network which allows the vehicles to communicate through an IEEE 802.11b connection and sharing information about traffic and critical situations.

- **Smart cities** are outlined as an ICT solution to the management of complex dynamics (food and water supplies, local waste, public transport, tourism or security, among others) that can be found in major cities, as a result of the current population growth [27]. In this context, Wireless Mesh Networks (WMNs) [5] can be used as an extension of the Internet, which intends to provide a set of specific services to their participants in an economic and efficient manner. For instance, collaborative translation in a museum through a local Mobile Cloud, which is made up of a group of tourists [71]; a self-replicated and self-deployed tour guide service to provide continuous service to all users despite of network partitions [45]; a network of information about environmental variables of the city (temperature, humidity, human density, noise or illumination) gathered and shared by pedestrians [2][3]; a context-aware service to provide information about points of interest (restaurant, gas stations, etc.) in the vicinity of the tourists [136]; a system to provide a set of services within a shopping mall (navigation, offers, shop lists, reservations, restaurants, etc.) to its customers [64]; a system to support students in a university campus, providing services to document (e.g., minutes of lectures or homework) or information sharing (e.g., such as exam timetable or campus maps) [151] [23]; or a private social network made up for the participant of a conference, through such network they can meet up at the airport to share a taxi, meet other participants that speak the same language, or share information about the conference [131].

- **Emergency Scenarios & Situations**. In some scenarios, such as accidents in remote areas (e.g., high mountains or high seas), natural disasters (e.g., earthquake) or terrorist attacks, the common network infrastructures may not be available. In those cases, it is necessary to provide a support system to allow managing effectively and efficiently the action plan (situation, intervention, and risks), resources (human, material and technological) and the cost of all involved parts [69]. In this context, in [110] a system to support firefighters in the rescue tasks after an earthquake in a large city is proposed; or a collaborative system to support archaeological and scientific expeditions in the desert is proposed in [71] .

However, to achieve all the potential that these new computing environments offer and to provide dependable and efficient solutions to the scenarios in the above described, several research challenges have to be addressed.

1.2 Description of the Problem and Motivation

Since the introduction of the Mobile Computing, as an extension of Distributed Computing, it has been clear that **the main constraints faced in these environments are intrinsic to mobility, not to the technology** [142]. These constrains were identified as:

- Mobile elements are **resource-poor** relative to static elements.
- Mobility is **inherently hazardous**.
- Mobile **connectivity is highly variable** in performance and reliability.
- Mobile elements rely on a **finite energy source**.

These constrains have been inherited in the following computation paradigms, based on mobile systems, such as Ubiquitous Computing, MCC, and IoT.

In ubiquitous scenarios in general and within Mobile Computing, **Mobile Ad-hoc NETworks** (MANETs), play a fundamental role, owing to the autonomy and flexibility that they provide in the communication system [33]. However, the dynamic network topology that characterizes MANETs must be conveniently addressed to guarantee the **dependability** of the system [12]. The dynamicity in the network topology is produced by the mobility of the nodes that make up the network. Further, the nodes may be switched off or may be disconnected (temporarily or permanently). Since these networks are typically multi-hop, this usually implies link failures, route changes or even network partitions, which could have a profound impact on the availability of the services deployed in the network.

This supposes a challenge for software architects, who currently do not have a practical and complete architectural approach to address the dynamicity of these mobile systems. Among the currently existing software architectural approaches, the **Service Oriented Architecture** (SOA) [50], due to its flexibility, interoperability and scalability, stands between the most promising approach to address software system design for Ubiquitous Computing. It has been also claimed as the approach to accomplish IoT [102].

However, **SOA itself is not enough to satisfactorily address software design in dynamic systems** [22]. And currently, **dynamically reconfigurable runtime architectures** are one of the main research challenges of SOA:

> The services runtime infrastructure should automatically leverage distributed service components and resources to create an optimal architectural configuration according to both a particular user's requirements and the application characteristics. [126]

As a result, **self-adaptive architectures** combined with traditional approaches, such as SOA, have been gaining importance in the research community [169]. This confirms what Satyanarayanan already noted in 1996: ***"adaptation is key to mobility"*** [142].

A self-adaptive or autonomic architecture has been complemented with self-* features from **Autonomic Computing** field (i.e., self-healing, self-configuration, self-optimization, and self-protection [84]). In this way, the system has the capability of reducing the consequences of context changes in the quality attributes.

Self-adaptive software is also the solution to address the new research challenges introduced by Ubiquitous Computing, and to effectively and satisfactorily provide services and applications in these environments [143]:

- **Effective Use of Smart Spaces**. In a well-defined space, software and hardware elements have to be able to interact automatically between them, being conscious about the context by which are surrounded and adapting their behaviour accordingly.
- **Invisibility**. The increasing number of devices, services, applications, and data could become unmanageable for users. Thus, to the extent possible, services and applications in ubiquitous environments should adapt and operate without the explicit intervention of the users.
- **Localized scalability**. As computer environments grow in complexity (user, devices, applications or services) the demand of computational resources increases, especially bandwidth. This comes into direct conflict with the resource-aware approach of mobile systems, and it may cause serious scalability problems in full developed Ubiquitous environments (e.g., IoT). Therefore, ubiquitous systems have to break with the traditional inexistence of distance on the Internet (i.e., a server has to serve a client independently of the distance). Software designers have to achieve Localized Scalability, this is, an information item has to reach only the area in which it has meaning and relevance.
- **Masking Uneven Conditioning**. Smart spaces are usually made up of heterogeneous technologies, which can vary as the users move between different of these spaces. Additionally, all the spaces in which the users are moving may not offer the same technologies. This is why software designers have to foresee the execution conditions in which their system can be found, and take the necessary measures to make it able to adapt to these situations.

Additionally, these new computing environments are characterized by being resource-constrained. Therefore, not only to provide a self-adaptive solution is necessary, but also a **resource-aware** solution that balances the performance of the system with efficient resource consumption.

Nowadays, in the literature some proposals that aim to address this problem can be found. However, these are frequently ad-hoc solutions that have been developed for specific scenarios, and they are based on an implicit and often restricted, context model. Moreover, they often assume reliable communication channels. This can be considered as a strong assumption, since data or message loss is an inherent property of mobile and ad-hoc wireless networks, especially in highly dynamic and large-scale systems. These proposals will be thoroughly analysed in Chapter 3.

1.3 Hypothesis and Objective

The **hypothesis** for this research work is the following one:

> Advanced Service Oriented Architectures applied to highly dynamic systems (e.g., ubiquitous or IoT systems) could contribute to the improvement of the quality attributes of the system as a whole.

The **main objective** of this thesis is

> To design and implement a proposal based on a self-adaptive software architecture that improves the system quality (in particular availability and reliability) and the applicability of services, by addressing the new challenges posed by the new dynamic computation systems.

In order to achieve the main objective we plan to address the following specific ones:

- To study and identify the main limitations of the current software architectural design approaches for mobile and ubiquitous systems.
- To study and identify the principal techniques and methods of Autonomic Computing to address the challenges posed by mobile and ubiquitous systems concerning software design.
- To review and analyse the existing proposals on dynamic service or data provisioning in dynamic environments, and to identify from the analysis the desirable features of a candidate solution.
- To propose, design and implement specific services and components that dynamically allow the software architecture adaptation according to context changes, thus providing better support for applications and users in dynamic mobile and ubiquitous systems.
- To evaluate and validate the feasibility of the proposal using tools able to model and simulate the dynamic nature of computer networks, and therefore to handle and evaluate dynamic models.

1.4 Structure of the Doctoral Thesis

This thesis is divided into three parts:

i. The first part of this thesis is composed of three chapters. In this first chapter, the context in which this research work is developed, the motivation, hypothesis, and objectives are presented. Chapter 2 introduces an overview of the concepts and technologies related to the design and development of self-adaptive architectures in dynamic environments. Chapter 3 presents a thorough review of existing approaches to address the problem domain. In this chapter, the desirable features of a candidate solution addressing the main objective are analysed.

ii. In the second part, the main contribution of this thesis, the Proteo architecture, is described. Chapter 4 introduces the design of Proteo architecture, providing an overview of its main components. In Chapter 5 the Proteo model is provided. This offers a detailed vision of Proteo architecture, its components and behaviour. Chapter 6 outlines the development and configuration of the proposal in a network simulator, this allows us to validate and evaluate it. Lastly, in Chapter 7, the results obtained during the evaluation of Proteo architecture are shown, analysed and discussed.

iii. The third part of this work provides an overview of work already in progress to continue expanding and improving Proteo architecture in Chapter 8. Finally, Chapter 9 summarises conclusions and the possible research lines for future work are outlined.

1.5 Publications

The articles published to date as a result of the research performed on this thesis are listed below.

Journals

- **A Context-Aware Architecture Supporting Service Availability in Mobile Cloud Computing**. G. Guerrero-Contreras, J. L. Garrido, S. Balderas-Díaz, C. Rodríguez-Domínguez. *IEEE Transactions on Services Computing* 2017 vol: 10 (6) pp: 956 - 968. [JCR 2016 Q1; IF 3.520] 10.1109/TSC.2016.2540629

- **A Collaborative Semantic Annotation System in Health: Towards a SOA Design for Knowledge Sharing in Ambient Intelligence**. G. Guerrero-Contreras, J. L. Navarro-

Galindo, J. Samos, J. L. Garrido. *Mobile Information Systems* 2017 vol: 2017 pp: 1-10. [JCR 2016 Q4; IF 0.849] 10.1155/2017/4759572

- **Using Actigraphy and mHealth Systems for an Objective Analysis of Sleep Quality on Systemic Lupus Erythematosus Patients**. S. Balderas-Díaz, M. P. Martínez, G. Guerrero-Contreras, E. Miró, K. Benghazi, A. I. Sánchez, J. L. Garrido, G. Prados. *Methods of Information in Medicine* 2017 vol: 56 (2) pp: 171-179. [JCR 2016 Q3; IF 1.772] 10.3414/ME16-02-0011

- **Self-adaptive deployment of services in mobile environments: a study of the communication reliability on the host election algorithm**. G. Guerrero-Contreras, S. Balderas-Díaz, C. Rodríguez-Domínguez, J. L. Garrido, A. Valenzuela. *Journal of Reliable Intelligent Environments* 2016 vol: 2 (4) pp: 197-207. 10.1007/s40860-016-0029-3

Book Chapters

- **Designing New Low-Cost Home-Oriented Systems for Monitoring and Diagnosis of Patients with Sleep Apnea-Hypopnea**. S. Balderas-Díaz, K. Benghazi, J. L. Garrido, G. Guerrero-Contreras, E. Miró. *ICTs for Improving Patients Rehabilitation Research Techniques* 2015. pp 210-221. 10.1007/978-3-662-48645-0_18

International Conferences

- **Integrating a Dual Method on a General Architecture to Self-Adaptive Monitoring Systems**. S. Balderas-Díaz, K. Benghazi, J. L. Garrido, G. P. M. O'Hare, G. Guerrero-Contreras. *Advances in Intelligent Systems and Computing* 2017 vol: 1 pp: 528-538. 10.1007/978-3-319-56535-4_54

- **Impact of Transmission Communication Protocol on a Self-adaptive Architecture for Dynamic Network Environments**. G. Guerrero-Contreras, J. L. Garrido, M. J. Rodríguez-Fórtiz, G. P. M. O'Hare, S. Balderas-Díaz. *Advances in Intelligent Systems and Computing* 2017 vol: 206 pp: 115-124. 10.1007/978-3-319-56538-5_12

- **BaaS-4US: A Framework to Develop Standard Backends as a Service for Ubiquitous Applications**. F. Carranza-Garcia, C. Rodriguez-Dominguez, J. L. Garrido, G. Guerrero-Contreras. *15th International Conference on Ubiquitous Computing and Communications and International Symposium on Cyberspace and Security (IUCC-CSS)* 2016 pp: 23-30. 10.1109/IUCC-CSS.2016.012

- **Trending Technologies and Standards Supporting the Development of Quality Collaborative Web Applications: The Case Study of VIRTRAEL**. C. Rodríguez-Domínguez, F. Carranza-García, G. Guerrero-Contreras, J. L. Garrido. *Ambient Intelligence and Smart Environments* 2016 vol: 21 pp: 94 – 103. 10.3233/978-1-61499-690-3-94

- **An Approach Addressing Service Availability in Mobile Environments**. G. Guerrero-Contreras, S. Balderas-Díaz, C. Rodríguez-Domínguez, A. Valenzuela, J. L. Garrido. *Workshop Proceedings of the 11th International Conference on Intelligent Environments*, Prague, Czech Republic, July 15-17, 2015. vol: 19 pp: 46-57. 10.3233/978-1-61499-530-2-46

- **An Introduction to Continuous Interaction**. C. Rodríguez-Domínguez, J. L. Garrido, G. Guerrero-Contreras, F. Carranza, A. Valenzuela. *Workshop Proceedings of the 11th International Conference on Intelligent Environments*, Prague, Czech Republic, July 15-17, 2015. vol: 19 pp: 82-92. 10.3233/978-1-61499-530-2-82

- **Dynamic Replication and Deployment of Services in Mobile Environments**. G. Guerrero-Contreras, C. Rodríguez-Domínguez, S. Balderas-Díaz, J. L. Garrido. *New Contributions in Information Systems and Technologies* 2015 vol: 353 pp: 855-864. 10.1007/978-3-319-16486-1_85

- **Self-adaptive Service Deployment in Context-Aware Systems**. G. Guerrero-Contreras, J. L. Garrido, C. Rodríguez-Domínguez, S. Balderas-Díaz. *Ubiquitous Computing and Ambient Intelligence. Personalisation and User Adapted Services*. UCAmI 2014. vol 8867, pp. 259–262. 10.1007/978-3-319-13102-3_42

- **Consistent Management of Context Information in Ubiquitous Systems**. G. Guerrero-Contreras, J.L., Garrido, S. Balderas-Díaz, C. Rodríguez-Domínguez. *Internet and Distributed Computing Systems - 7th International Conference*, (IDCS) 2014, Calabria, Italy, September 22-24, 2014. 10.1007/978-3-319-11692-1_16

- **Towards a Self-Adaptive Deployable Service Architecture for the Consistent Resource Management in Ubiquitous Environments**. G. Guerrero-Contreras, J. L. Garrido, K. Benghazi, S. Balderas-Díaz, C. Rodríguez-Domínguez. *Workshop Proceedings of the 10th International Conference on Intelligent Environments*, Shanghai, China, June 30 - July 1, 2014. vol: 18 pp: 206-217. 10.3233/978-1-61499-411-4-206

- **A service-based platform for monitoring and diagnosis of patients with SAHS symptoms**. S. Balderas-Díaz, K. Benghazi, J.L. Garrido, G. Guerrero-Contreras, E. Miró.

Proceedings of the 8th International Conference on Pervasive Computing Technologies for Healthcare, PervasiveHealth 2014, Oldenburg, Germany, May 20-23, 2014. pp: 290-293. 10.4108/icst.pervasivehealth.2014.255365

- **Designing a Service Platform for Sharing Internet Resources in MANETs**. G. Guerrero-Contreras, J.L., Garrido, C. Rodríguez-Domínguez, M. Noguera, K. Benghazi. Advances in Service-Oriented and Cloud Computing - Workshops of ESOCC 2013, Málaga, Spain, September 11-13, 2013. vol: 393 pp: 331-345. 10.1007/978-3-642-45364-9_27

Chapter 2

Foundations

Chapter Abstract

As introduced in the previous chapter, since Distributed Systems evolved to Mobile Systems, adaptation, specifically self-adaptation, has been a fundamental requirement to successfully and efficiently address the development of software systems. However, despite the importance of this requirement, still, there is no an architectural approach to address it. To this end, SOA has to be combined with methods and techniques from Autonomic Computing field, as well as complementary approaches, such as Event-Driven Architecture (EDA) and Microservices. Throughout this chapter, these concepts are examined and exposed from the perspective of self-adaptation and system autonomy.

Additionally, in this chapter the Leader Election Problem and some of the existing traditional election algorithms to solve it to address coordination in distributed systems are exposed. Moreover, the quality attributes treated throughout this work are defined (Agility, Interoperability, Reliability, Scalability and Service Availability). Finally, the tools used to support the development of this thesis are introduced: the ns-3 network simulator tool, which will be used to evaluate and validate our proposal, and SysML, a general-purpose modelling language for *"System of Systems"* (SoS).

Chapter Contents

2.1 Introduction

The purpose of this chapter is to briefly lay the foundations on which the work of this thesis is developed. The concepts covered in this chapter are studied from the perspective of the adaptation and system autonomy.

The chapter is organized as follows. In Section 2.2 the problematic and requirements for adaptation, or more specifically self-adaptation, of mobile and ubiquitous systems, are exposed, and how these still need to be addressed in IoT and MCC. Section 2.3 presents the software approaches aimed to successfully and effectively address the software system development in dynamic computation environments. Section 2.4 introduces the Leader Election Problem, and the traditional algorithms proposed to solve it. Additionally, Section 2.5 defines the system quality attributes that are treated, in one form or other, throughout this thesis. Finally, Section 2.6 introduces the tools that will support the development of this work.

2.2 Ubiquitous Systems

The introduction of the ubiquitous computing, not as a technology but as a new computation paradigm, brings which Weiser defined as **The Age of Calm Technology** [168]. In which, technology fades into the background of our daily lives, providing an unconscious human-computer interaction.

Ubiquitous computing can be considered an extension of mobile computing, which in turn can be considered an extension of distributed systems. Each extension brings new potential advantages, providing increasing liberty and flexibility to the user. However, they also pose new research challenges, in addition to inheriting existing ones from previous environments (Fig. 2.1).

Since the introduction of the mobile computing, as an extension of distributed computing, it has been clear that mobile clients have to be adaptive. As Satyanarayanan noted:

> Any viable approach to mobile computing must strike a balance between these competing concerns [resource poverty, low trust, and robustness]. This balance cannot be a static one; as the circumstances of a mobile client change, it must react and dynamically reassign the responsibilities of client and server. In other words, the mobile client must be adaptive. [142]

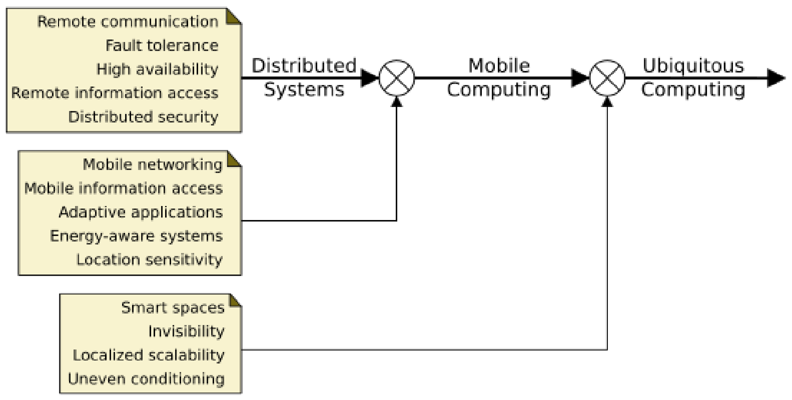


Fig. 2.1 *Taxonomy of computer systems research problems in ubiquitous computing. Extracted from [143].*

To this end, mobile and ubiquitous systems have to be context-aware, i.e., they need to know what it is happening and how in their execution contexts to reason what adaptations are necessary to perform.

2.2.1 Context-Aware Computing

There have been different definitions of what context is. However, one of the most accepted is that provided by Dey in 2001: *"**Context** is any information that can be used to characterise the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves"* [46]. On this basis, Dey defined what a **context-aware system** is: *"A system is context-aware if it uses context to provide relevant information and services to the user, where relevancy depends on the user's task"* [46]. A latter and simplified definition of context was given by Hirschfeld et al.: *"any computationally accessible information can be considered as context"* [67]. Therefore, in the design of a context-aware system, a fundamental step is to clearly identify what is the relevant context information that has to be taken into consideration.

The context information can be structured in two main categories, according to its source (Fig. 2.2) [144]: (1) human factors and (2) physical environment. Each of this categories can be divided into three subcategories:

- **Human factors**:
  - *Information of the user*: habits, emotions, bio-physiological conditions, etc.

  - *Social environment*: group dynamics, social interactions, co-location of other actors, etc.
  - *User's task*: activity, engaged tasks, goals, etc.

- **Physical environment**:
  - *Location*: absolute and relative positions, co-locations, etc.
  - *Infrastructure*: surrounding resources for computation, communication, task performance, etc.
  - *Physical conditions*: noise, light, temperature, etc.

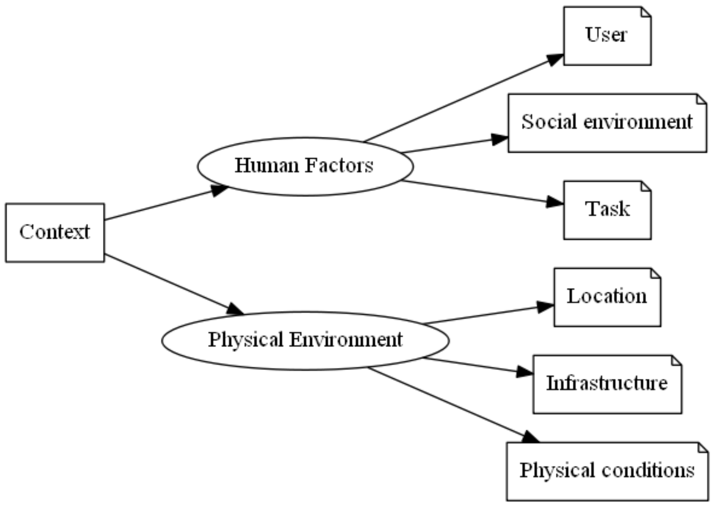


Fig. 2.2 *Context feature space. Extracted from [144].*

2.2.2 Advanced Systems

From Mobile and Ubiquitous Computing, new computation systems have arisen. IoT and MCC are of particular interest, due to their increasing popularity. These more specialized environments bring new advantages and challenges. However, they inherit the need of self-adaptation from Mobile and Ubiquitous Computing.

Internet of Things

Internet of Things (IoT) can be defined as *"the pervasive presence around us of a variety of things or objects [...] which, through unique addressing schemes, are able to interact with each other and cooperate with their neighbours to reach common goals"* [11].

IoT paradigm brings a promising vision. However, its realization is also challenging, owing to IoT encompasses a wide number of ICT fields (Fig. 2.3), taking their advantages as well as their research challenges. Among these ICT fields, as Wireless Sensor Networks (WSN) or RFID technologies, SOA have been claimed as the software architectural approach to realize IoT [102]. However, as stated by Borgia: *"as SOA standards were originally designed for connecting programs running on static computers, their direct application to IoT devices is not feasible but requires an adaptation to this context"* [22]. Specifically, services are expected to adapt autonomously to different context situations. According to [102] it is required: *"methods for discovering, **deploying** and composing **services at run-time** in a distributed fashion, supporting autonomicity within all phases of the service life-cycle."*

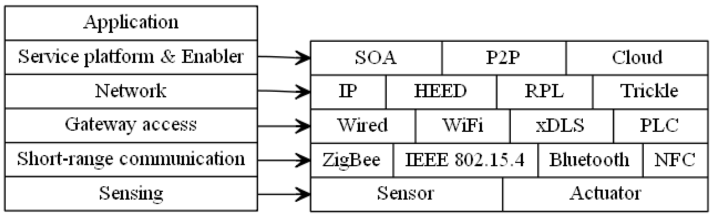


Fig. 2.3 *IoT layers. Extracted from [22].*

The main challenges that have to be addressed to realize IoT have been identified as [102]:

- *Device heterogeneity*. IoT will be composed of a wide set of different devices and technologies, with different capabilities. This issue is closely related to **Uneven Conditioning** of Ubiquitous Computing.

- *Scalability*. In IoT, scalability has to be considered at four levels: (1) naming, (2), data communication and networking, (3) information and knowledge management, and (4) service provisioning and management. Note that the second level, data communication, and networking, is closely related to the need of achieving **Localized Scalability** in Ubiquitous Computing.

- *Ubiquitous data exchange through proximity wireless technologies*. The need of ad-hoc networks requires for smart object the adoption of wireless medium communication technologies, which, together the high increase in devices inherit to the IoT, may lead to technical issues in the spectrum availability.

- *Energy-optimized solutions*. It is necessary to devise solutions able to **optimize energy usage**, prioritizing its use even against the system performance.

- *Localization and tracking capabilities.* Abilities to track the location of a smart object in its environment. Especially necessary for product management applications.

- *Self-organization capabilities.* Owing to the dynamicity and complexity of IoT scenarios, it is necessary to provide **autonomic capacities** to the things, to minimize the human intervention. The need of the IoT nodes to organize themselves in ad-hoc networks arises the need of providing them with capabilities to manage the data sharing and organization, adapting to the dynamic changes in context.

- *Semantic interoperability and data management.* Owing to the amount of data produced will be necessary to provide standardize formats, models, and semantics to allow its analysis and interoperability.

- *Embedded security and privacy-preserving mechanisms.* Security and privacy are considered key requirements to IoT, due to the deep impact that they can have on the daily lives of their users.

Mobile Ad-hoc Cloud Computing

Cloud computing [8] can help to address one of the main weaknesses of mobile systems: the lack of computational resources. The Cloud allows the delegation of some processing and storage tasks that must otherwise be carried out in nodes with limited resources. This opens up new possibilities for mobile systems, and as a result, the concept of Mobile Cloud Computing (MCC) has emerged, that is, *"an infrastructure where data storage and processing could happen outside of the mobile device"*, which can operate under three main schemes [52]:

- *A traditional scheme*, where the mobile node delegates part or all of its operation to the Cloud, through an Internet connection. This is performed under a client-server paradigm, where the mobile node acts as a regular client of the Cloud through an Internet connection. In turn, this scheme can be divided into different subcategories according to the degree of computational offloading on the Cloud.

- *A Local Mobile Cloud* or *Spontaneous Mobile Cloud*, where a set of mobile nodes make up a local Cloud between them through short-range connections. The storage or processing is carried out through a distributed process between the nodes that make up the local network. This schema is of particular interest in domains where a stable Internet connection cannot be established, and thus it is not possible to connect with an external Cloud.

- *A hybrid scheme*, where the mobile system is composed of an Ad Hoc Mobile Cloud, which is made up of the nodes of the mobile system, and, at the same time, it is supported by an external Cloud under a traditional scheme.

Currently, the first scheme is widely spread, as nowadays the majority of the current applications are offered through a SaaS ("Software as a Service") service model [8] (e.g., Google Apps: Calendar, Gmail or Drive). While SOA is a model for the design and development of software systems, Cloud and SaaS are models for their software implementation and delivery, respectively [31]. Thus, it is especially interesting to complement SOA and SaaS models to develop different services at different levels of abstraction [93], which can be accessed from mobile devices connected through the Internet.

The second scheme, Ad-Hoc Mobile Cloud, is required in domains where mobile applications also need to be supported by data storage and processing services provided by the mobile platform itself in a transparent and flexible way. This is of particular interest in the IoT, where the advantages of MCC can alleviate some of its limitations [132], such as those related with the *architecture technology*: MCC provides an efficient and flexible architectural model; and *ubiquitous communications*: MCC allows heterogeneous interoperability through services.

Nevertheless, this cloud approach is usually supported by a MANET, which poses new challenges [34] that must be conveniently addressed to guarantee the dependability (availability, reliability, safety, integrity, and maintainability) of the Mobile Cloud (Fig. 2.4). Shila et al. state that:

> An underpinning concept in cloud computing is resource virtualization. A network node can provision its physical resources to multiple applications through certain service level agreements. The significant challenge is to design proper adaptive resource allocation techniques for balancing the tradeoff between resource utilization efficiency and QoS guarantee. [150]

According to these authors, Autonomic Computing postulates as a promising solution to manage Ad-Hoc Cloud Computing systems [150].

2.3 Software Development Approaches

In this section, the techniques, methods and approaches for the design and development of software systems in dynamic computing environments are presented. As it will be described, these approaches are often related, and its combination will be crucial to successfully address the problematic presented in Ubiquitous environments.

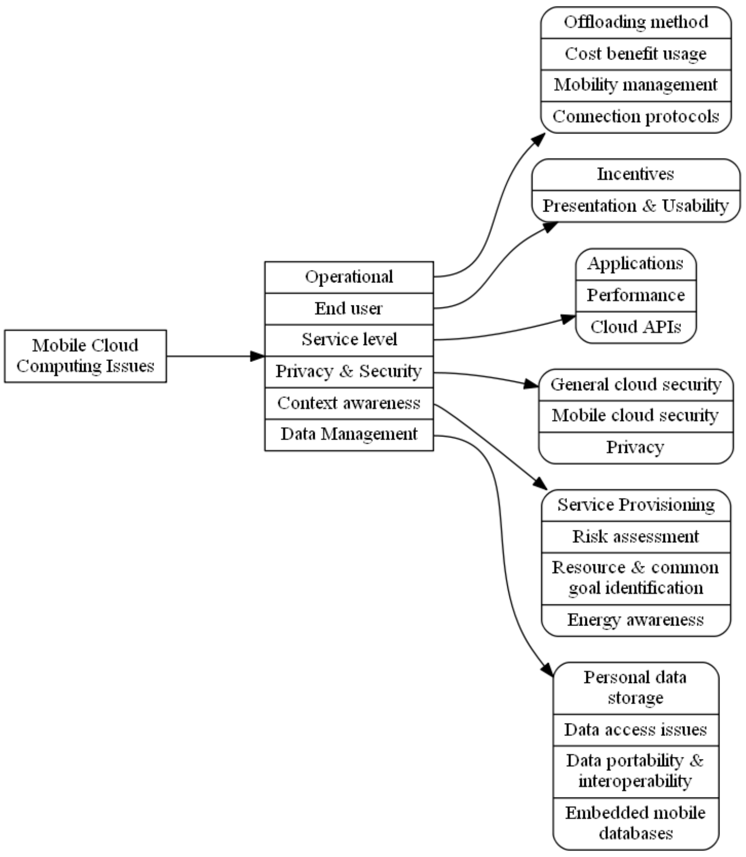


Fig. 2.4 *A taxonomy of issues in mobile cloud computing. Extracted from [52].*

2.3.1 Autonomic Computing and Self-adaptive Software Systems

Autonomic Computing arises in 2001 by the hand of IBM as a solution to address the increasing computing systems' complexity and an enabling technology for pervasive environments. The objective of autonomic computing is to provide computer systems with self-management and allow them to run at peak performance 24/7 without explicit human intervention. The system will be able to adjust its operation to changing conditions (demand patterns, workload or failures). Autonomic Computing (self-management) is achieved by means of four main self-* aspects [84]:

- Self-configuration: *"Automated configuration of components and systems follows high-level policies. System adjusts automatically and seamlessly."*
- Self-optimization: *"Components and systems continually seek opportunities to improve their own performance and efficiency."*

- Self-healing: *"System automatically detects, diagnoses, and repairs localized software and hardware problems."*
- Self-protection: *"System automatically defends against malicious attacks or cascading failures. It uses early warning to anticipate and prevent system wide failures."*

Self-adaptive software and autonomic computing terms are related and in many cases, they can be used interchangeably. However, a distinction can be made: while self-adaptive software mainly covers the middleware and upper software layers, autonomic computing can also cover layers below the middleware [141].

In ubiquitous computing, adaptation is defined as *"the reactive process triggered by a specific event or a set of events in the context, with an ultimate goal to improve the QoS perceived by the end-user"* [82].

Adaptation Control Loop

An autonomic system is composed of a collection of autonomic elements. These elements will manage their own behaviour and relations with others (machine or human). This is managed by the commonly accepted MAPE-K IBM's adaptation control loop (Monitor, Analyse, Plan, Execute and Knowledge) (Fig. 2.5).

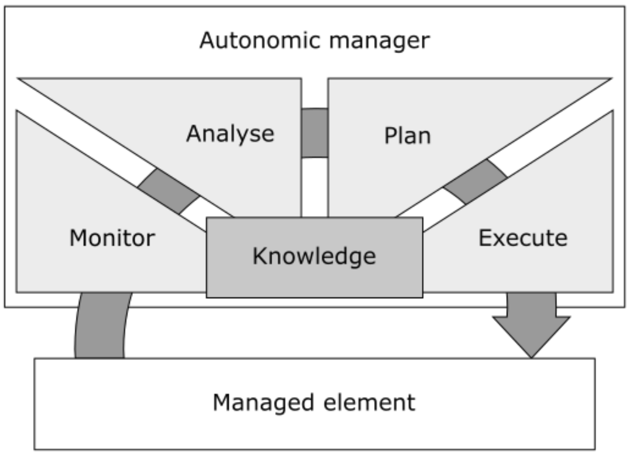


Fig. 2.5 *MAPE-K IBM's adaptation control loop. Extracted from [73].*

In general, the autonomic manager is responsible for the management and control the adaptation process on the managed element. A managed element can be a hardware resource (storage, CPU or device) or a software entity (database, document or a legacy system).

Monitoring becomes an essential feature to react to context changes, with self-optimization and reconfiguration purposes, and to know if the requested performance is currently meeting.

Kakousis et al. adapt MAPE-K loop (see Fig. 2.5) proposed by IBM to an adaptation loop for ubiquitous computing consisting in three consecutive phases:

- *Context sensing and processing*: the data of user context and system context are recollected and processed to produce high-level events that might trigger a system adaptation.
- *Adaptation reasoning and planning*: the system reasons about what is needed to be done and how to accomplish with the overall adaptation goal.
- *Adaptation acting*: the necessary adaptation mechanisms are used to implement the decision made in the system adaptation planning.

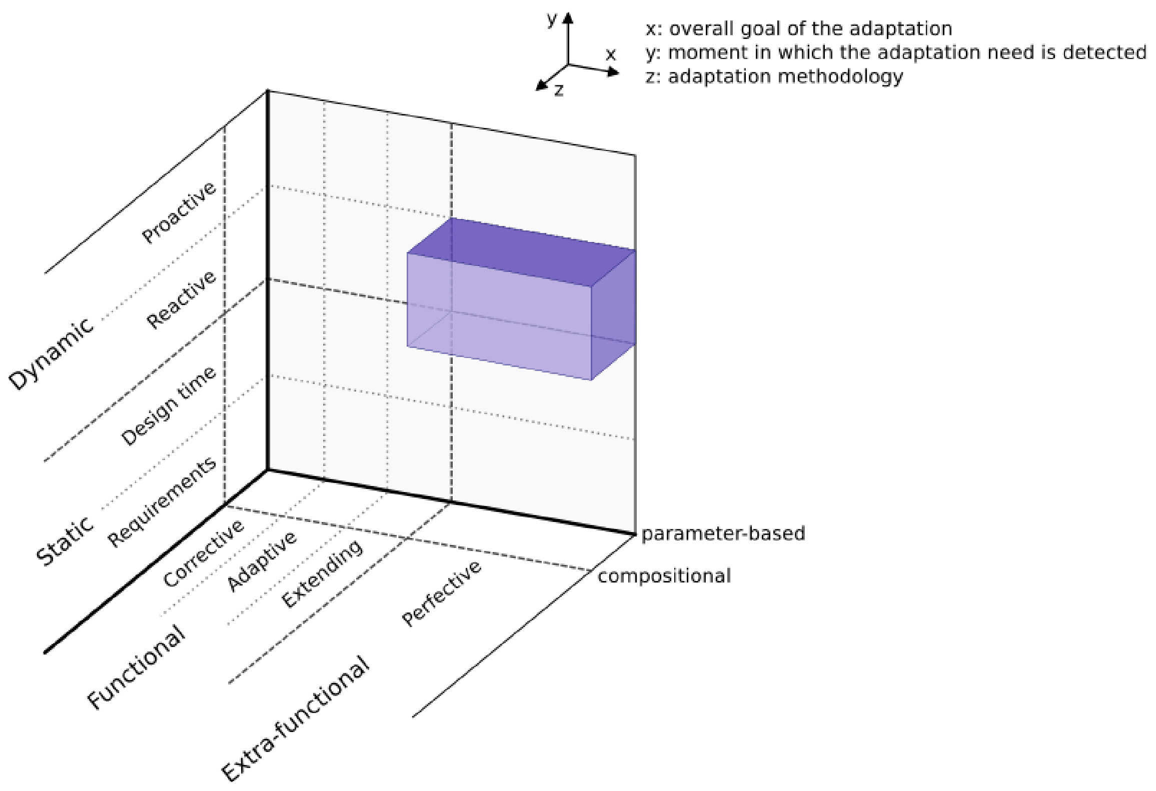


Fig. 2.6 *Software adaptation classification on the basis of (x) overall goal of the adaptation; (y) moment in which the adaptation need is detected; and (z) adaptation methodology. This figure is based and integrates the classifications proposed on [82][85][26][15].*

Software Adaptation Classification

Software adaptation can be classified in different categories (see Fig. 2.6), from the perspective of (1) the overall goal of the adaptation, (2) the moment in which the adaptation need is detected or applied, and (3) the adaptation methodology.

According to the overall goal of the adaptation, Kakousis et al. propose a coarse-grained classification: functional and extra-functional [82]. Whereas functional adaptation refers to system's functionality, extra-functional is related to QoS attributes of the system. Ketfi et al. propose a finer grained classification[85]:

- *Corrective adaptation*: a faulty part of the system is replaced to continue with the normal operating of the system.
- *Adaptive adaptation*: system reacts to events of the context that can affect to its behaviour.
- *Extending adaptation*: the system is extended with new functionality that has not been considered at design-time.
- *Perfective adaptation*: modifications are made to the system to improve it QoS attributes.

Regarding the moment in which the adaptation need is detected or applied, Kakousis et al. propose a coarse-grained classification again: dynamic and static adaptation. Whereas dynamic adaptation refers to modifications performed at run-time, static adaptation refers to modification performed at design or deployment time. Canal et al. and Becker et al. propose a finer classification [26][15]:

- *Requirements adaptation*: within the software development life cycle, this adaptation generally occurs in the analysis phase, where the requirements have to be modified owing to a system specification modification or extension.
- *Design-time adaptation*: refers to the necessaries modification to solve mismatch types that usually occur on component-based systems.
- *Dynamic adaptation*: refers to the modifications occurred at run-time. These can be divided into reactive and proactive:
  - *Reactive adaptation*: when the system reacts in response to any change in the execution context.
  - *Proactive adaptation*: system takes the initiative to suggest or adapt itself usually to accomplish in a better way user's needs.

Finally, software adaptation can be classified from the point of view of the adaptation methodologies. According to Kakousis et al. there are two main adaptation methodologies [82]: parameter-based and compositional. Whereas parameter-based adaptation refers to adaptation using variable modification or parameter configurations, compositional adaptation refers to adapt system's behaviour employing structural, geographical, interface or implementation modifications.

Reasoning and Acting Phases: Techniques and Methodologies

Kakousis et al. include a study of the most popular techniques and methodologies for the phases of *Adaptation reasoning* and *Adaptation acting*. Regarding *Adaptation reasoning* techniques they evaluate them on the basis of resource efficiency, coverage of the context value domain, ease of building, adaptation openness, adaptation coordination, evolvability, and traceability.

The techniques applied in this phase are [82]:

- *Action-based (rule-based) adaptation*: Based on states and actions and IF-THEN rules. This technique is widely applied in reasoning processes. This is because of the approaches based on this technique are not resource demanding and have high traceability. However, they may present low coverage of the context value domain, as they only consider binary decisions. Moreover, in complex systems, they need high development effort, as the designers have to define every possible situation to which the system will address. Additionally, dynamic modification and evolution of the rules are often complex.

- *Goal-based adaptation*: Instead of defining each event-reaction situation explicitly, the designers only define the overall goal of the system, and it is the system itself who decides the actions that have to be taken to reach that goal. This technique is closely related to Multi-Agent System. These approaches require a low knowledge of the system functionality from the designers. However, in resource-constrained situations, it can cause problems when the system is not able to fulfil all the goals simultaneously, or there are conflicting objectives.

- *Utility functions*: Adopted from the field of economics and AI, the utility functions are applied to map a situation or state to a real numeric value, according to a set of preferences usually represented as weights. In this way, an entity can compare different situations easily and select that what better fits the system preferences and the current context. According to Kakousis *"utility functions have been applied in several works, especially QoS-based, for measuring the suitability of adaptation alternatives in fluctuating environments"* [82]. Utility function based approaches present a good coverage of

Table 2.1 *Comparison of adaptation reasoning approaches based on their applicability for ubiquitous computing and mobile environments. Extracted from [82].*

		Evaluation Criteria				
		Action-based	Goal-based	Utility functions	Case-based	Reinforcement learning
Reasoning Approach	Resource Efficiency	High	High	Medium	Low	Low
	Context coverage	Medium	Low	High	Low	Medium
	Ease of building	Low	Medium	Low	Low	Low
	Adaptation openness	Low	Low	High	Medium	Medium
	Adaptation coordination	High	Low	High	Medium	Medium
	Evolvability	Low	Low	High	High	High
	Traceability	High	High	Low	High	Low

the context value domain, as they can produce numeric values on the basis of any context value; they can address easily different adaptation objectives, using the specialization and combination of different utility functions; and are easily adaptable, even at run-time. However, they may present a challenge for designers in complex systems and can offer poor traceability.

- *Case-based reasoning*: Based on the experience using a set of already resolved previous problems or situations, CBS tries to learn how to resolve similar problems. However, it is necessary a significant amount of previous knowledge. However, adaptation problems in ubiquitous computing present a wide set of variables, and sometimes, that, if not are well-represented in the case may lead to CBS-based approaches to low coverage of the context value domain. Although, CBS-based approaches are highly evolvable, as facilitates self-learning from the new resolved cases, improving their performance.

- *Reinforcement learning*. It is an unsupervised learning mechanism that performs a trial-and-error search. The main disadvantage of this technique is the high resource demand.

Table 2.1 shows the overview of the evaluation of the different adaptation reasoning techniques. Although the utility functions are one of the better valued, it must be emphasized that these approaches can be combined in multiple ways and that the election of one approach is determined by the execution context of the system and its overall objective.

Regarding *Adaptation acting*, Kakousis et al. include the following mechanism to implement dynamic software adaptation:

- *Code mobility*, defined as *"the process of migrating or moving running program instances, codes or objects from one host to another"* [82]. This is supported by Mobile Code Languages, divided into two categories: (1) strong if it allows the mobility of the execution state and code; and (2) weak if it only allows the mobility of the code and initialisation data. Code mobility can be executed in two ways: (1) code pushing, where an entity sends the code/process to a host, or (2) code pulling, where a host downloads the code/process from a remote entity. This can be implemented employing a Client/Server scheme, Code On Demand, Remote evaluation or Mobile agents.

- *Parameter adaptation* modifies the behaviour of the system through the customization of predefined configuration variables. It is a cheap and fast adaptation technique. However, it has to be in design time where the configuration variables have to been settled, thus if new adaptation requirements arise at run-time, these cannot be addressed through this technique. Moreover, in complex systems an excessive number of configuration variables can become difficult to manage.

- *Compositional adaptation* is based on the traditional component-based paradigm [155]. Szyperski defines a component as *"a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties"* [155]. In this way, it is possible to exchange dynamically (i.e., at run-time) parts of the system to improve its behaviour or performance and to address new adaptation requirements that can arise at run-time and be not expected, without interrupting the execution of the system. Within dynamic compositional adaptation, it can be found two types [82]: tunable software, which only allows extra-functional adaptations; and mutable software, which allows altering the business logic of a program.

Distributed Adaptation and Related Areas

According to [82], distributed adaptation is one of the main challenges of dynamic adaptation in pervasive systems. The adaptation can be performed by a centralized entity (centralized adaptation), or the different entities implied in the adaptation have to collaborate and agree to achieve a global adaptation (distributed adaptation). Although decentralized approaches introduce complexity, owing to the requirement of negotiation among the distributed entities, the robustness and fault-tolerance of the system are highly increased, as any node is essential. This is of great interest in dynamic contexts, where the nodes or communication links can fail or disappear.

Finally, it worthwhile mentioning that service-oriented computing (Section 2.3.2) and grid systems are approaches that could support the design and development of autonomic systems. However, autonomic elements can be both service consumers and providers [84]. This is highly related with SOA 2.0 (Section 2.3.5). Moreover, Autonomic Computing is highly related with Multi-Agent Systems (Section 2.3.6), specifically when the autonomic elements are goal-oriented designed.

2.3.2 Service Oriented Architecture

Service-Oriented Computing (SOC) is understood as a representation of a new generation distributed computing platform. It comprehends its own design paradigm and principles, design patterns catalogues, pattern languages, architectural model, concepts, technologies, and frameworks. According to T. Earl, service-oriented computing:

> represents a new generation computing platform that encompasses the service-orientation paradigm and service-oriented architecture with the ultimate goal of creating and assembling one or more service inventories. [50]

Otherwise, Service-Oriented Architecture (SOA) is defined as:

> an architectural model that aims to enhance the efficiency, agility, and productivity of an enterprise by positioning services as the primary means through witch solution logic is represented in support of the realization of strategic goals associated with service-oriented computing. [50]

It must be highlighted that although SOA and service-oriented computing are often used as synonyms, SOA is seen as a form of technology architecture and its implementation is specific for each enterprise. Therefore, according to T. Earl:

> building a technology architecture around the service-oriented architectural model establishes an environment suitable for solution logic that has been designed in compliance with service-orientation design principles. [50]

Service

In both cases, and in general, in a service-oriented solution, the service is the most fundamental unit of logic. Services are defined as:

> physically independent software programs with distinct design characteristics that support the attainment of the strategic goals associated with serviced oriented computing. Each service is assigned its own distinct functional context and is comprised of a set of capabilities related to this context. Those capabilities suitable

> for invocation by external consumer programs are commonly expressed via a published service contract (much like a traditional API). [50]

According to the definitions provided, services are considered as an independent unity of software. This allows coordinating a set of aggregate services to answer different needs using software composition. In this way, a service can be classified as *Utility*, *Entity* or *Task* service. *Utility* services are basic units, usually independent of the application and the domain that fulfil basic functionality. These services are composed to provide functionality to Entity services. *Entity* services are mean to represent basic entities, especially in the business area. Although these services can be considered independent of the application, they are designed for a specific domain. Finally, *Task* services are high-level services with a functionality directly associated with a specific task or process. These services are generally positioned as the controller (orchestra conductor) of a composition or orchestration of other Entity or Unity services. Because of this dependency, they are often placed within close geographic proximity to the services of which it is composed.

Ideally, a service fulfils a set of design principles, within the service-orientation paradigm: standardized service contract, loose service coupling, service abstraction, service reusability, service autonomy, service statelessness, service discoverability and service composability.

Service Autonomy Design Principle

Specifically, service autonomy is understood as the ability of a service to self-govern at runtime as one key design [50]. This is the degree to which the service can act independently. The fulfilling of this design principle involves enhanced reliability and predictability, owing to the independence of the service of external components.

To fulfil this principle, it is necessary an execution and deployment environment where the service can have a significant amount of control (including sometimes even the data models). Therefore, it should be designed and deployed an infrastructure able to support the service to be isolated, composed or moved as required.

It can be mainly found two forms of autonomy: (1) runtime autonomy and (2) design-time autonomy. Runtime autonomy refers to *"the level of control a service has over its processing logic at the time the service is invoked and executing"* [50]. However, due to service composition, a service can exist as part of an aggregate of services that may also be part of another service composition. Thus, the autonomy of a service is determined by the collective autonomy of the services that participate in the composition. In fact, if in this composition some agnostic service is participating, the autonomy can be considered lost.

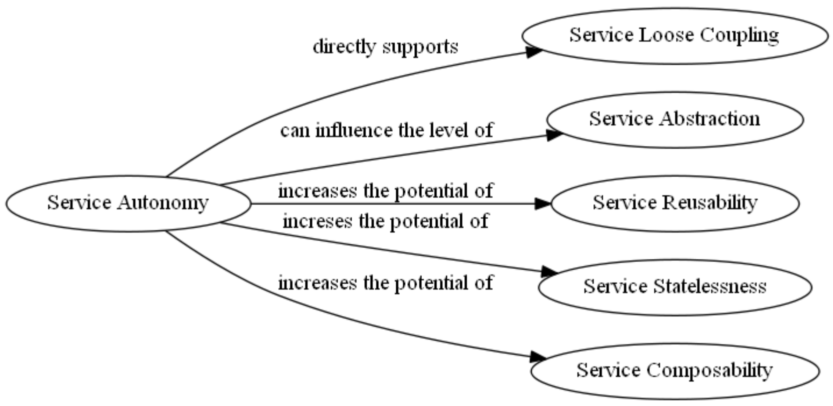


Fig. 2.7 *Relation of Service Autonomy design principle with other service-orientation design principles. Extracted from [50].*

Design-time autonomy can be defined as *"the level of freedom [service owners] [...] have to make changes to a service over its lifetime"* [50]. However, a problem arises when a consumer binds itself to a service's contract, and thus the service can no escape to this obligation. This could generate some problems related with: (1) to scale a service to satisfy a higher usage demand, (2) modify or enhance the hosting environment, and (3) upgrade or replace the technology to give response to new requirements.

The autonomy of a service can be categorized into four levels:

- *Service contract autonomy*: the contracts of the services in the same inventory do not present overlapped functionality.
- *Shared autonomy*: logic and resources are shared between different entities.
- *Service logic autonomy*: service logic is independent, but data resources are shared. This involves unpredictable levels of concurrent data access, inconsistency, prolonged query execution times and low scalability.
- *Pure autonomy*: logic and data resources are independent. Within this category, we can find functional isolation, where the services share the physical host device and absolute isolation, where the service, service components and data are on a dedicated host. Although especially referring to data and other resources, pure autonomy can be hard to achieve, the use of replication can be an effective approach to address it [50].

Service Autonomy design principle is closely related to other design principles (Fig. 2.7).

The service autonomy involves some challenges as [50]: (1) misjudging the service scope; (2) wrapper service and legacy encapsulation; and (3) overestimating service demand. This last on is related to what the resources are that should be assigned to a service. Physically isolate services in dedicated host devices can lead to a high cost, whereas the usage demand or the relevance of the service can vary along the life cycle of the system.

Enterprise Service Bus

The Enterprise Service Bus (ESB) it one of the most common compound patterns in service-oriented computing [51]. The primary objective of this pattern is to provide a middle layer to facilitate the scalability and the communication of heterogeneous services, including legacy entities, in a system. This pattern is usually made up of a service broker, an asynchronous message queuing system, and an intermediate routing system. However, it only provides a set of message functions to establish a connection between services and resources. Because of this, upon the ESB the Canonical Schema Bus pattern can be built [51], which aims to enforce contract-level standardization, in addition to the functionality provided by the ESB.

Although the ESB has had a great impact and it has been adopted as one of the main patterns in SOA, it also has been defined as a "spaghetti box" [158]. In the way that, although ESB tries to centralize and homogenise the communication between the services, there is still a set of services that are directly connected, but instead by an independent connection, this communication goes through the ESB together other similar cases. This often makes to ESB to present integrity issues, in addition to a single point of failure. On the other hand, centralized controls improve the system security.

2.3.3 Microservices

Although standardization can help to improve the interoperability of a system, it reduces the flexibility. Additionally, in an SOA approach, services often possess several capabilities, which makes hard to replace or modify them at run-time. To address this issue, in the last years a different approach has arisen, the Microservice Architecture. This can be defined as *"a style of engineering highly automated, evolvable software systems made up of capability-aligned microservices"* [106]. As an SOA implementation relies on services as the basic entity, a microservice architecture relies on microservices. They are defined as *"an independently deployable component of bounded scope that supports interoperability through message-based communication"* [106]. As with traditional services, microservices are characterized by a set of well-defined features: small in size, messaging enabled, bounded by contexts, autonomously developed, independently deployable, decentralized, and built and released with automated processes. However, two are the most important characteristics: decentralization and autonomy.

It is worth to highlight that although in this approach, the (micro)services tend to be simpler, the architecture goes more complex. Some of the technical benefits of this architectural style are [106]: supporting multiple technologies/languages/frameworks enables graceful degradation of service and facilitates to replace or envolve part of the system at run-time, greater agility, higher composability, independent service deployability, replaceability. When referring to small in size, the general assumption is that a microservice only has one responsibility, only performs one well-defined function. While in SOA, a service usually implies a considerable quantity of standardization (e.g., contracts), a microservice is supposed to be a more agile piece of software, easily deployable and replaceable. Usually, it offers its functionality over an HTTP resource API.

2.3.4 Event-Driven Architecture

Event-Driven Architecture (EDA) can be defined as an architecture in which *"a notable thing happens inside or outside your business [or system], which disseminates immediately to all interested parties (human or automated). The interested parties evaluate the event, and optionally take action"* [101]. EDA represents a complete set of elements, which together are capable of disseminating and evaluating events and take actions according to the nature of these events. Another definition can be *"EDA is an approach to enterprise architecture that enables systems to hear events and react to them intelligently"* [157].

Event

An event usually represents a change in state. This is a change in a data value [157]. In EDA, an event has to be machine-understandable and poses three levels of detail: (1) event notification, this is, the event has occurred or not; (2) event definition, what about the event is; and (3) event detail, what has happened within the event context. It must be highlighted, that in EDA anything can be or trigger an event.

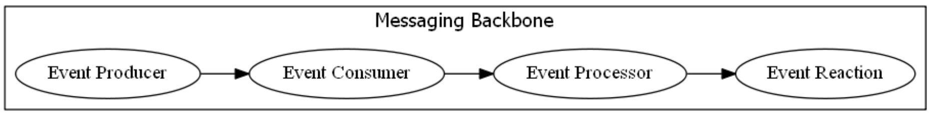


Fig. 2.8 *The paradigmatic EDA. Extracted from [157].*

A basic EDA is composed of the following entities: (1) Event Producers, also known by publishers, which creates and disseminated the events; (2) Event Consumers, also known by listeners, which can differentiate an event by its definition from others and it can interpret

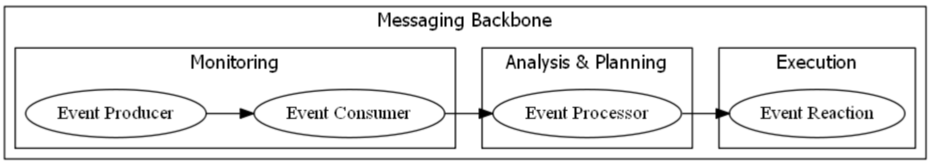


Fig. 2.9 *The paradigmatic EDA within Autonomic Computing MAPE-K loop.*

the meaning of the event. Usually, in a basic implementation of EDA, the consumer only can receive the events that it is meant to hear; (3) Event processors, which can determine the event impact and the next action to perform according to the information provided by the event (occurrence, definition, and detail); (4) Event Reactions, these can range from automated actions to human interventions; and (5) Messaging Backbone, which has to provide a communication infrastructure between the entities of the architecture, in a high level of decoupling. The basic event processing flow can be depicted as in Fig. 2.8.

It is worth emphasising how EDA event processing flow and entities have a straightforward relationship with the MAPE-K loop of Autonomic Computing (Fig. 2.9).

It is true that Planning stage has a blurred boundary between Event Processor and Event Reaction. Especially when in Event Reaction it is necessary the human intervention. However, in a more automated EDA, the Planning stage is more identified with the Event Processor. Regarding event processing, it can be found three basic patterns: (1) simple processing, where an Event Producer produces an event with a non-periodically pattern, and this event is always processed; (2) stream processing, where the event production is periodic, and the event consumer only reacts when specific criteria in the event information is met; and (3) complex event processing (CEP), where not only an event but a set of them are analysed under multiple logical conditions to obtain high-level information. CEP is becoming the main event processing technique, and future trends are including AI techniques to recognize and react to patterns of events.

Finally, EDA can be classified into two types [157]: explicit and implicit. Whereas in the explicit EDA, the relation between the Event Producers and Consumers is ad-hoc, in the implicit EDA, there is no dedicated connection between these entities. This makes, implicit EDA a more agile, scalable and maintainable approach, especially when is combined with CEP.

2.3.5 SOA 2.0

However, while SOA had still a great presence in software system design, it started to talk about SOA 2.0. Also known as advanced SOA o event-driven SOA, Steve Harris, vice president of Oracle Fusion middleware in 2006, defined it as *"the term that we're using to talk about the combination of service-oriented architecture and event-driven architecture"* [90]. In SOA 2.0, services are not just passive entities, but also they can receive and generate events proactively, thus getting the benefits of both approaches: interoperability, platform independence, flexibility and a modular design from SOA, and low coupling between components of the system, thanks to event distribution system from EDA.

Some authors refer to the combination of SOA and EDA, or an EDA implementation of SOA as *"the ablest paradigm to deliver the optimal architecture"* [157], regarding flexibility and cost-effectiveness.

SOA 2.0 represents a step more in the direction of service autonomy. Allowing to services, through the event spreading, react to changes in the context of the system, and therefore adapt the software system and the behaviour of the services as necessary to maintain or even increase the performance of the system.

2.3.6 Multi-Agent Systems

Multi-Agent Systems (MAS) is a set of agents that interact between them, in a distributed and concurrent way, to achieve some goal. An agent can be defined as *"a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its design objectives"* [170]. By definition, MAS are a subclass of concurrent systems [170]. However, there are two fundamental differentiating aspects:

- Synchronization and coordination take place at run-time.
- The interaction between agents is economic-driven. This is, in contrast with distributed/concurrent systems, where it is implicit that all entities share a common goal, in multiagent systems, agents act, first of all, in their own welfare (i.e., on behalf of the user).

Usually, an intelligent agent possesses three main features [170]:

- *Reactivity*. Agents are capable of sensing their environment, and react according to the changes that occur within it.
- *Proactiveness*. Besides, to react to changes in its context, intelligent agents can take the initiative to accomplish their objectives.
- *Social ability*. Intelligent agents can interact with other entities (other agents or humans) to achieve their objectives.

Table 2.2 *Multi-Agent System and Service Oriented Computing, similarities and differences. Source: Own elaboration.*

Multi-Agent Systems	Service Oriented Computing	Self-adaptive Services
Approaches to build **complex software systems**		
Autonomy		
Agents act individually fulfilling their individual goals	There are no dependences between services	
Adaptive		
in response to changing requirements, services and exceptions		
Sociability	**Interoperability**	
Intelligent society	Interface level	
Communicating agreed-upon protocols		
Distributed	**Distributed/Centralized**	
There is no a central control entity	Choreography/Orchestration	
Rationality		
Reactivity	**Encapsulation**	**+ Reactivity**
Proactivity		**+ Adaptability**
Highly formal specification	**Availability/Discovery**	
	Public registries	

At this point, agents and services can be considered interchangeable entities. In fact, there is two main point of view about the difference between an agent and a service:

- *An agent is the same that a service.* This is based on the idea that anything that is an agent could be deployed as a service and any service can be conceived as an agent.

- *Services are simpler things that agents.* From this point of view, services can be used by agents (as clients), and agents can support services: *"While, in principle, autonomic functionalities can thus be implemented by using agents to create high-level behaviours coordinating the system's functional layer, it is important to provide the latter with proper interface and reflective functionalities"* [116].

From the point of view of this work, although an agent and a service are similar to software entities, some differences must be taken into consideration in the design of software architecture. The main difference between an agent and a service is that in the agent-oriented world there is no 'invoked a method' concept [170]. If an agent A request to agent B to perform an action β, it cannot be assured that the agent B will perform such action, as perform action β may not be in the best interest of agent B.

Therefore, although MAS and SOC are close approaches that share several design features (Table 2.2), even with the inclusion of Self-adaptive services to SOC, which completes SOC with reactivity and adaptability capabilities, an agent will present selfish behaviour, in order to accomplish its own objectives (on the behalf of its user).

2.4 Leader Election Problem

Distributed Computing arises several challenges in comparison with Centralized Computing [156], especially those related with synchronization, coordination or duplication of responsibilities. In this context, the Leader Election Problem responds to the necessity of electing in a distributed system a unique process or node to perform some task. In this way, a distributed system can rely on a *"centralized"* entity to simplify some of the tasks mentioned above.

Although different variants of the Leader (or Coordinator) Election Problem can be found, the most general is defined as: *"[in a group of distributed processes] each process eventually decide that either it is the leader or it is not the leader, subject to the constraint that exactly one process decides that it is the leader"* [10].

This problem is usually addressed by a leader election algorithm, and formally, the algorithm has to meet three requirements to correctly solve it [57]:

- *Termination*: The algorithm has to converge to a solution in a finite time or number of steps.
- *Uniqueness*: Once the election is resolved, there has to be exactly one leader, even if there are several concurrent elections in the same group [42].
- *Agreement*: The rest of the processes or nodes of the group have to know who the leader is.

Two election algorithms are the traditional referents in this context [42]:

- **The algorithm of Chang and Roberts** [30]: an asynchronous algorithm where processes arranged in a logical ring transmit a token message to perform the election.
- **The Bully algorithm** [53]: asynchronous algorithm devised to allow processes to crash during an election.

Although both algorithms were designed to operate in reliable communication environments, the algorithm of Chang and Roberts is particularly vulnerable in this context, owing to the loss of the token message will result in a fatal failure of the algorithm. Despite the detection and recovering of failure (i.e., detection of the token lost and regeneration) are possible, these can be costly, especially in large-scale systems [7]. For this reason, this work will focus on the Bully algorithm and their later improvements, which are detailed in the following sections.

2.4.1 Bully Algorithm

Bully algorithm is based on three kinds of messages:

- *Election message*: to start and announce an election.
- *Answer message*: to respond to an election message.
- *Coordinator message*: it is sent by the elected node to announce its election.

Election in Bully is performed on the basis of the ID of the node. It is assumed that each node has a unique ID and the IDs of the nodes of the network are known.

On this basis, it performs in the following way:

(A) **Given a node *N* that starts the election:**

- **If *N* is the highest node:** *N* becomes the elected node and sends the *Coordinator* message to announce its election to the rest of the nodes of the network.
- **If *N* is not the highest node:**
  1. *N* broadcasts an *Election* message only to these other nodes with higher IDs than itself.
  2. *N* **waits for an *Answer* message**:
     - **If *N* does not receive any *Answer* message**: *N* becomes the elected node and sends the *Coordinator* message to announce its election to the rest of the nodes of the network.
     - **If *N* receive an *Answer* message**: *N* gives up the election and waits for a *Coordinator* message.

(B) **Given a node *P* that does not start the election**:

- **If *P* receives an *Election* message from** *N*: In such case, since *Election* messages are sent to nodes with higher ID, *P* will have a higher ID than *N*. Therefore, *P* will reply with an *Answer* message to *N* and will take over the election process from step (A).
- **If *P* receives a *Coordinator* message from** *N*: *P* knows now that *N* has won the election and it is the new coordinator.

2.4.2 Kordafshari Algorithm

Since the proposal of the Bully election algorithm, different improvements have been suggested, of which that proposed by Kordafshari et al. can be highlighted [89]. The advantage of Kordafshari proposal over other variants of the Bully algorithm is that Kordafshari el al. proposed an effective method to reduce the number of coordination messages and concurrent elections without make additional assumptions.

In addition to the messages used in the Bully algorithm (see Section 2.4.1), Kordafshari variant introduces a new kind of message: *Grant message*. This message is used by the node that starts the election to inform the elected node that it has won the election.

On this basis, the algorithm operates as follows:

(A) **Given a node *N* that starts the election**:

- **If *N* is the highest node**: *N* becomes the elected node and sends the *Coordinator* message to announce its election to the rest of the nodes of the network.
- **If *N* is not the highest node**:
  1. *N* broadcasts an *Election* message only to that other nodes with higher IDs than itself.
  2. ***N* waits for an *Answer* message**:
     - **If *N* does not receive any *Answer* message**: *N* becomes the elected node and sends the *Coordinator* message to announce its election to the rest of the nodes of the network.
     - **If *N* receive an *Answer* message or more**: *N* selects the node with highest ID that have replied and sends it a *Grant* message. After this, *N* waits for *Coordinator* message.

(B) **Given a node *P* that does not start the election**:

- ***P* receives an *Election* message from *N***:
  1. Since *Election* messages are sent to nodes with higher ID, *P* will have a higher ID than *N*. Therefore, *P* will reply with an *Answer* message to *N*.
  2. ***P* waits for a *Grant* or *Coordinator* message**:
     - **If *P* receives a *Grant* message from *N***: *P* has been elected as leader and it will send a *Coordinator* message to announce its election to the rest of the nodes of the network.

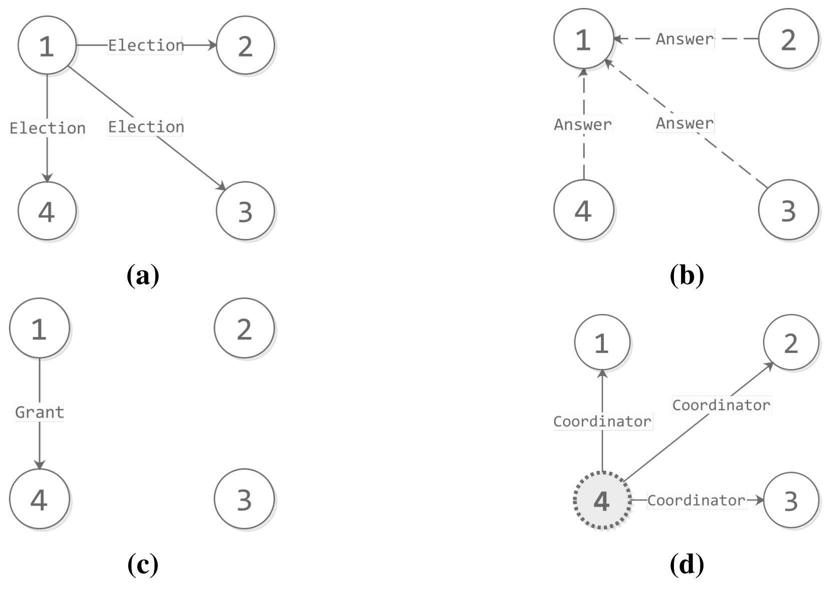


Fig. 2.10 *Kordafshari election algorithm example.*

· **If** *P* **receives a *Coordinator* message from** *Q*: *P* knows now that *Q* has won the election and it is the new coordinator.

- *P* **receives a *Coordinator* message from** *Q***:** *P* knows now that *Q* has won the election and it is the new coordinator.

Fig. 2.10 depicts the general operating of Kordafshari algorithm. In the election depicted, the Node 1 starts the election (Fig. 2.10a) and the Node 4 (Fig. 2.10c), as the node with highest ID, is elected as leader by means of a *Grant* message.

Moreover, in order to reduce the number of concurrent elections, Kordafshari proposes a mechanism in which, if there are multiple nodes that sends *Elections* messages, i.e., multiple nodes that are carrying on simultaneously an election, only the *Election* messages of the nodes with the lowest ID will be replied with *Answer* messages. In other words, if there are multiple concurrent elections, the election initiated by the lowest ID node will continue over the other elections.

An example where Nodes 1, 2 and 3 (Fig. 2.11a) initiate a concurrent election is depicted in Fig. 2.11. In this case, Nodes 2 and 3 give up their elections and *Answer* messages are only send to Node 1 (Fig. 2.11b), the lowest ID node, who declares Node 4 as the leader (Fig. 2.11c).

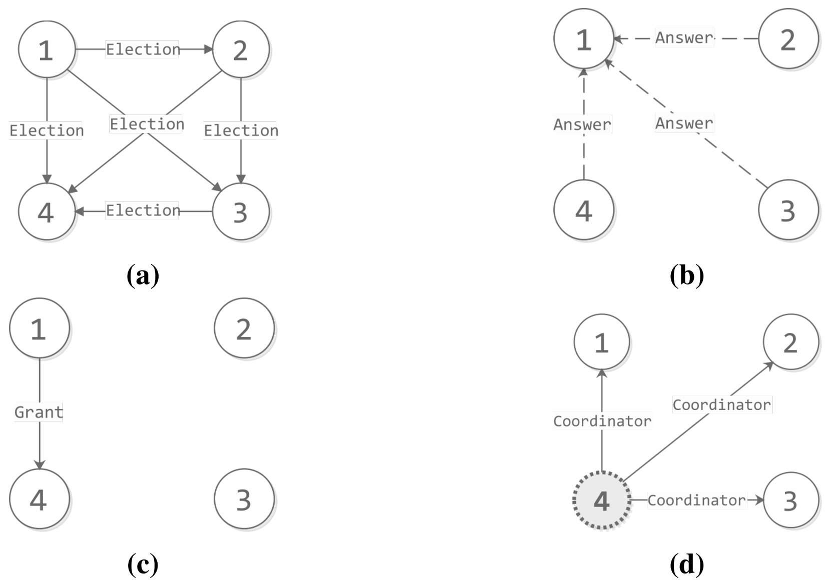


Fig. 2.11 *Kordafshari election algorithm example with multiple concurrent elections.*

2.5 System Quality Attributes

System Quality Attributes are used to evaluate the performance of a system. They are considered as accomplished **non-functional requirements** [115]. This section does not intend to provide an exhaustive list of the existing quality attributes, but only to define those related in some way with this thesis.

2.5.1 Agility

Satyanarayanan defines the ability of an application or system to promptly adapt to changes in the context as **agility** [142]. However, he highlights that a highly agile system may suffer from instability, as the system could consume its resources reacting to minor and irrelevant changes in the context.

2.5.2 Interoperability

Interoperability can be defined as *"a situation where two o more software components must work together to perform a function. Usually, interoperation involves two o more software applications exchanging data, processing those inputs, and sharing outputs with one or more applications"* [157].

2.5.3 Reliability

The **reliability** of a system can be defined as *"the ability of a system to perform as designed, with-out failure, in an operational environment, for a stated period of time"* [159]. Thus, in order to measure the reliability of a system, or usually, of a system component, it is firstly required to define when it fails.

MTBF

Mean Time Between Failures (MTBF) is calculated as the average time between failures of a system [159]. It is generally used to predict the elapsed time between failures and the reliability of the system.

2.5.4 Scalability

Scalability can be defined as the ability of a system to *"function well (**without degradation of other quality attributes**) when the system is changed in size or in volume"* [115]. Scalability presents diverse dimensions, among which are [65]:

- *Functional*: the capacity of adding new features to the system easily.
- *Geographical*: the capacity of the system to move for a local to a global environment, increasing distance and number of users.
- *Workload*: the capacity of the system to afford, generally, an increase of operational demand, adding, removing or modifying its components to accommodate the changes in demand.

2.5.5 Service Availability

In simple words **Service Availability** can be defined as *"the ability of services to be accessible as needed, whenever and wherever they are required"* [74]. In IT systems where there is a service agreement, Service Availability is often calculated as in Equation 2.1.

$$Availability = \frac{Agreed\ Service\ Time - Down\ Time}{Agreed\ Service\ Time} \times 100\% \tag{2.1}$$

This measure is often related to reliability measures such as MTBF or maintainability measures such as Mean Time to Recover Service (MTRR).

2.6 Tools

In this section, the tools used during the development of this thesis are introduced. These are two: the ns-3 network simulator tool, which will be used to evaluate and validate our proposal, and SysML, as a general-purpose modelling language for *"System of Systems"* (SoS).

2.6.1 Network Simulator 3 (ns-3)

The series of ns simulators, which stands for Network Simulator, was initiated in mid-nineties at Lawrence Berkeley National Laboratory. These arise as discrete-event network simulators, specially devised for educational and research purposes. The second version of the simulator, ns-2, was released in 1996 and it was maintained until 2011, when the last version, ns-2.35, was released. Its third version, ns-3 [114], was first released in July 2008.

ns-2 has had, and it still has, a profound impact in scientific research. Using the Scopus[1] database of peer-reviewed literature it can be found 2863 documents that have the words "ns-2" and "simulator" in their title, abstract or keywords, 148 documents the last year (2017). Although, as it can be seen in Fig. 2.12, the number of documents that mention ns-2 still surpasses the number that mentions ns-3, since ns-3 first release, the scientific papers that use ns-2 is being reduced. Moreover, the total number of papers that reference ns-2 or ns-3 remains stable.

In Fig. 2.13 it is shown the use in scientific papers of ns simulators (ns-2 and ns-3) in comparison with other popular network simulators [108] (NetSim [109], OPNET [118], OMNET++ [117], QualNet [133] and JSim [81]), according to the same procedure. As it can be seen, ns simulators are the most used simulation tools, not only together but also individually.

The simulator ns-3 has been built as a C++ library. It has been built from scratch (although some models have been ported from ns-2) and does not provide backward compatibility with ns-2. This library supplies a set of network simulation models, using object orientation provided by C++ and wrapping them over python. Nodes in ns-3 are designed after Linux networking architecture. In this way, key elements, such as sockets and net devices, are adjusted with those in a Linux system. This is an essential feature since it improves the realism of the models and makes them comparable with real systems. Moreover, ns-3 core supports both IP and non-IP networks, and provide models for WiMAX, Wi-Fi, or LTE and a variety of static or dynamic routing protocols. Finally, it should be noted that it is licensed under the GNU GPLv2 license, which allows users to study and change the source code of the own simulator.

[1]https://www.scopus.com

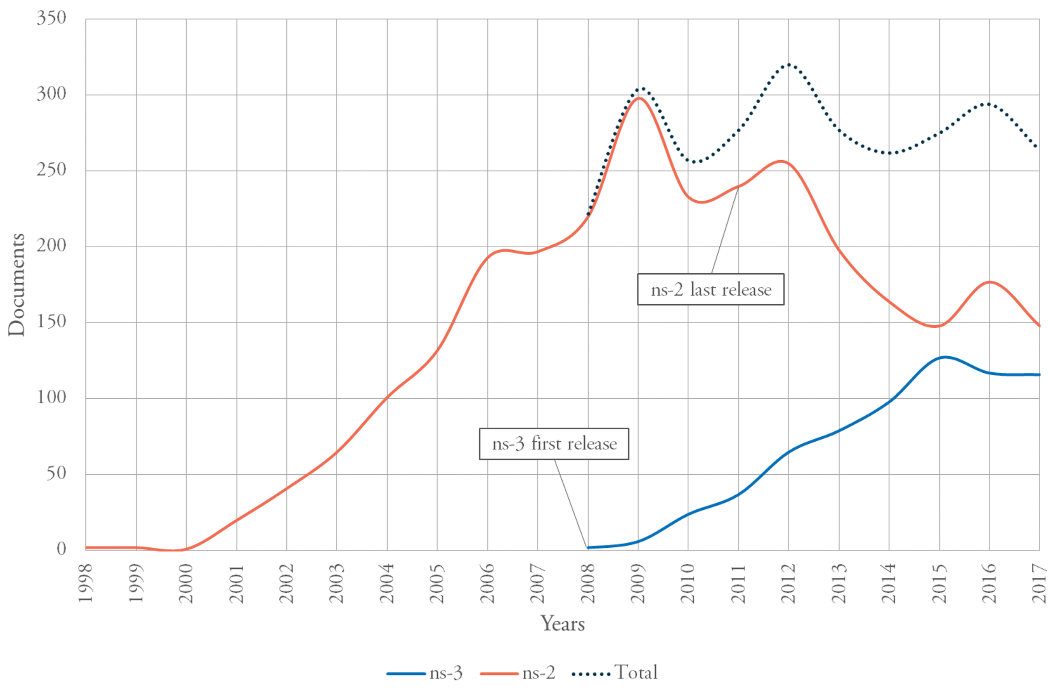


Fig. 2.12 *Documents that mention ns-3 or ns-2 in their title, abstract or keywords on the Scopus database of peer-reviewed literature. Source: Own elaboration.*

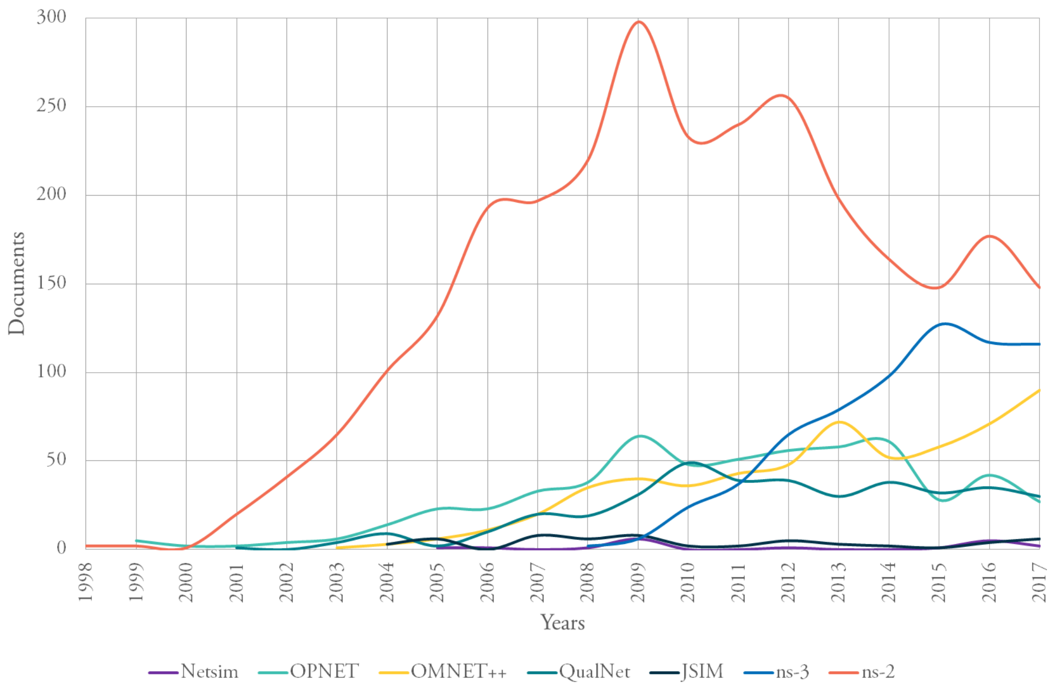


Fig. 2.13 *Comparative showing the number of scientific documents that mention Netsim, OPNET, OMNET++, JSIM, ns-3 or ns-2 in their title, abstract or keywords on the Scopus database of peer-reviewed literature. Source: Own elaboration.*

BonnMotion

The support provided by ns-3 to generate mobility scenarios under different mobility models is limited. In its current version (ns-3.28), it supports the following models: ConstantPosition, ConstantVelocity, ConstantAcceleration, GaussMarkov, Hierarchical, RandomDirection2D, RandomWalk2D, RandomWaypoint, SteadyStateRandomWaypoint and Waypoint. These models, although interesting, do not allow us to simulate group mobility or obstacle restrained scenarios. To overcome this limitation, it is possible to use external tools to generate mobility trace files that ns-3 can read and apply.

Currently, one of the most complete and utilized tools is BonnMotion [9][21]. BonnMotion is a Java-based software which allows to create and analyze a wide variety of mobile ad hoc network mobility scenarios. It also allows exporting the generate trace files to a variety of formats, readable by ns-2 or ns-3, GloMoSim/QualNet, COOJA, MiXiM, or ONE. It supports mobility models, such as Manhattan Grid, Random Waypoint or Reference Point Group Mobility (RPGM). BonnMotion is jointly developed by the University of Bonn (Germany), the Colorado School of Mines (USA) and the University of Osnabrück (Germany).

2.6.2 SysML

Architecture description languages (ADLs) emerged in the 1990s as a formal tool to model and represent the system architectures [36]. According to ISO/IEC/IEEE 42010 (Systems and software engineering—Architecture description) [77], an ADL *"is any form of expression for use in architecture descriptions"* and it has to meet the following requirements (clause 5 in ISO/IEC/IEEE 42010):

- Architecture description identification and overview information.
- Identification of the system stakeholders and their concerns.
- A definition for each architecture viewpoint used in the architecture description.
- An architecture view and architecture models for each architecture viewpoint used.
- Applicable architecture description correspondence rules, architecture description correspondences and a record of known inconsistencies among the architecture description's required contents.
- Rationales for architecture decisions made.

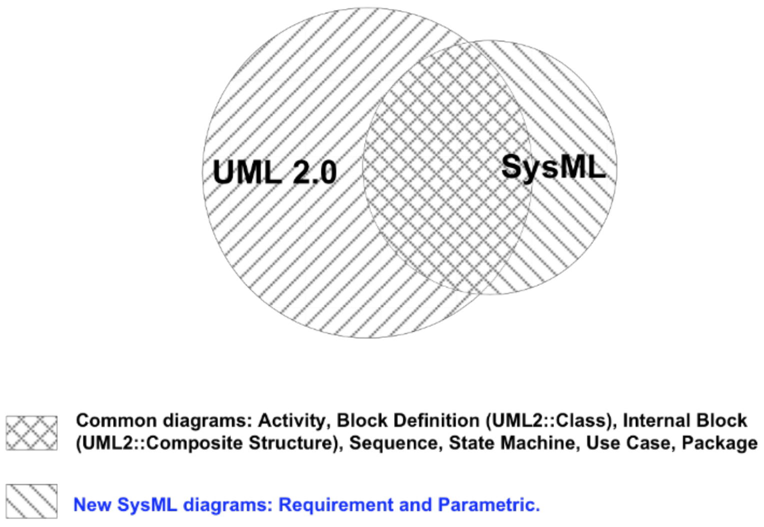

UML 2.0
SysML
Common diagrams: Activity, Block Definition (UML2::Class), Internal Block (UML2::Composite Structure), Sequence, State Machine, Use Case, Package
New SysML diagrams: Requirement and Parametric.

Fig. 2.14 *Relationship between SysML and UML. Extracted from [154].*

Between the varieties of ADLs proposed in the decade of 2000, π-ADL highlights as it was specifically designed for specifying static, dynamic and mobile architectures. Flavio Oquendo proposed π-ADL on 2004 [120].

However, despite the efforts and the variety of ADLs proposed in the decade of 2000, there is a lack of industrial adoption of ADLs [97]. The reasons exposed for this fact are: (1) ADLs are not practical, it is necessary to reduce their learning curve; and (2) most ADLs do not provide multiple views and cross-views consistency, which is essential when there are different stakeholders involved. As a result, formal ADLs lack mature tools to help the software architecture designers and generally are not included in the life-cycle software development [125].

For this reason, Flavio Oquendo, author of π-ADL, has begun to develop SysADL, a Systems Modeling Language (SysML) profile for software architecture description [94][119][121].

SysML is defined as *"a general-purpose architecture modeling language for Systems Engineering applications"* [154], and it was adopted by the Object Management Group (OMG) in July 2006. The last version, OMG SysML 1.4, was released in August 2015.

SysML supports the specification, analysis, design, verification, and validation of systems and systems-of-systems. These include hardware, software, information, processes, personnel and facilities systems. SysML is defined as a lightweight UML 2 Profile (Fig. 2.14). It reuses the majority of the UML diagrams, some of them without modifications, e.g., Use Case, Sequence and State Machine diagrams; and others with minor customizations, e.g., renaming Classes as Blocks and adding syntax and semantics for item flows. It also adds two new diagrams: Parametric diagrams, extending Composite Structure and UML Class diagrams; and Requirements diagrams, extending UML Class diagrams. It also worth noting that SysML is an enabling technology for Model-Based Systems Engineering.

Moreover, although SysML provides a semi-formal modelling, according to the ISO/IEC/IEEE 42010, SysML is considered, per se, an ADL:

> The Systems Modeling Language (SysML) is built upon UML. SysML defines several types of diagrams: Activity, Sequence, State Machine, Use Case, Block Definition, Internal Block, Package, Parametric, and Requirement diagrams. In terms of this International Standard [ISO/IEC/IEEE 42010:2011], each SysML diagram type is a model kind. SysML provides first-class constructs for Stakeholders, Concerns, Views, and Viewpoints so that users can create new viewpoints in accordance with this International Standard [ISO/IEC/IEEE 42010:2011]. [77]

After review the different options available about modelling languages: π-ADL, SysADL, and SysML, in this work we have opted for SysML. As exposed, the lack of adoption, documentation and support tools of π-ADL discarded it as a maintainable and flexible solution. Otherwise, the lack of maturity and proposal completeness of SysADL leaves SysML as the best option. SysML is a standard architecture modelling language, supported by the OMG, well documented and with mature tools to support the system modelling and design.

2.7 Summary

This chapter has introduced the software design challenges that mobile and ubiquitous system present, and how these have to be addressed in IoT and MCC environments as well. As a result, and to successfully address these challenges, self-adaptation has become a requirement for the software systems devised to operate in these environments. To this regard, the techniques, methods, and approaches for the design and development of software systems in dynamic computing environments have been revised.

SOA stands a promising architectural approach for software system design in this context. It proposes a modular distribution of the functionalities of a system through services and provides the foundations to build an interoperable and scalable system thanks to the use of standard protocols and service composition. However, to provide a complete solution, it has to be combined with methods and techniques from Autonomic Computing field, as well as complementary approaches, such as Event-Driven EDA (i.e., SOA 2.0) and Microservices.

Autonomic or self-adaptive systems are managed by the commonly accepted MAPE-K IBM's adaptation control loop. From this loop, it has been seen that reasoning and acting phases are of particular relevance. To this respect, the most widely used techniques in these phases have been revised. Utility Functions and Compositional adaptation techniques highlight in reasoning and acting phases, respectively, owing to its simplicity and potential.

Moreover, in distributed systems synchronization, coordination or to avoid duplication of responsibilities can be challenging. To this end, election algorithms arise as a response for a distributed system to rely on a *"centralized"* entity to simplify some of these tasks. This distributed election process can be considered as a self-adaptive approach and can be of special interest in the dynamic replication and deployment of service replicas. To this regard, two algorithms have been presented: the Bully algorithm and the improvement on it, introduced by Kordafshary et al.

Generally, the ultimate goal of self-adaptation is to improve the quality attributes of the system, according to the changes in its operational context. In this chapter, the definitions for Agility, Interoperability, Reliability, Scalability and Service Availability quality attributes have been provided.

Finally, the tools used to support the development of this thesis have been introduced: the ns-3 network simulator tool, which will be used to evaluate and validate our proposal, and SysML, as a general-purpose modelling language for *"System of Systems"* (SoS), which is also considered an ADL according to the requirements established by the ISO/IEC/IEEE 42010:2011 standard.

Chapter 3

Related Work

Chapter Abstract

This chapter reviews the existing proposals related to the dynamic service or data provisioning in dynamic topology environments. The research articles revised are divided into three categories: (1) approaches to enhance data/service QoS through dynamic replication, (2) leader election algorithms in dynamic environments, and (3) cluster head election algorithms. From the proposals studied, it can be highlighted that the majority explicitly assume reliable connections or say nothing about this assumption. The majority use a network simulator to evaluate their proposal. The cost measure taken as reference to evaluate the proposals is varied, but energy consumption, bandwidth usage and response time predominate. Finally, from the approaches to enhance data/service QoS through dynamic replication, only three ones partially address the synchronization of the replicas.

Chapter Contents

3.1 Introduction

In this chapter, the current proposals related to the dynamic service provisioning in dynamic topology environments are reviewed. These proposals are based on autonomic schemes, whose cornerstone is an autonomic control loop. As Section 2.3.1 introduced, it is accepted that a control loop must include, at least, the following phases: observation, analysis, planning, and execution.

In this context, the planning phase has received particular attention in the literature because it is the most important phase in the behaviour of the autonomic system. In this phase, two main questions have to be addressed: when to replicate and where to deploy the replica [153]. The answers will determine the capability of the system to react to context changes, the service quality attributes (e.g., both availability and reliability are deeply influenced by the allocation of the service), and the cost of the solution provided.

Whereas the decision about when to replicate is usually implemented by rules that are triggered by context events, the decision about where to deploy a replica is implemented by an election algorithm. In general terms, an **election algorithm** is defined as a distributed algorithm *"for choosing a unique process to play a particular role"* [42] (Section 2.4).

Therefore, two types of approaches are analysed in this chapter: (1) **service replication algorithms**, and (2) **general purpose election algorithms**. In the former, an election algorithm is used in combination with other techniques such as code mobility or service hibernation, to elect the most suitable node to host a service replica according to some defined strategy to enhance its quality attributes. The latter, as previously noted (Section 2.4), performs an election in order to elect a leader on the basis of some defined evaluation criteria. However, the objective of the election could be undefined or be related to different purposes.

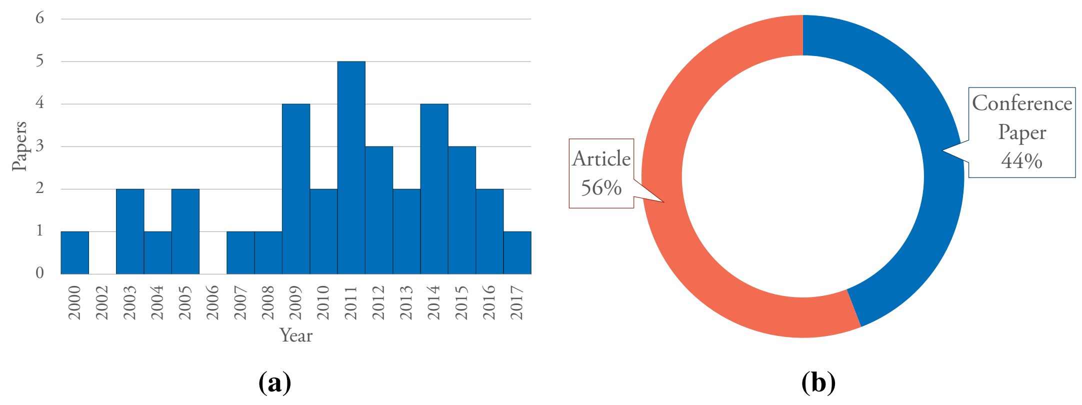


Fig. 3.1 *Distribution of the articles revised according to publication year and source type.*

The first search of articles was done according to the research objective defined (Section 1.3). From the articles obtained were excluded those not focused in the distributed election in dynamic environments, early proposals, or those where an adequate description of the research process was not carried out. It should be noted that the approaches that can be found for the dynamic provisioning of data in dynamic environments are also applicable in the dynamic provisioning of services. Hence, **in this review both terms service and data have been taken into consideration**. The total final selected articles down to 34. In Fig. **??**, the distribution of the articles revised according to publication year and source type is shown.

These proposals are presented and divided into three main groups: approaches to enhance data/service QoS through dynamic replication (Section 3.2), leader election algorithms in dynamic environments (Section 3.2) and cluster head election algorithms (Section 3.4). Finally, Section 3.5 presents a discussion over the literature review performed. This section also shows three comparative tables that summarised the main features of proposals revised.

3.2 Approaches to enhance data/service QoS through dynamic replication

The primary objective of the proposals presented in this section is to enhance the QoS (especially availability) of replicated data/service deployed on dynamic network topologies. This objective is directly related to the objective of this thesis. Some of the main features that will be analysed in these proposals are: Are the systems proposed reactive to context changes? If so, what events are taken into consideration?; What features are taken into account to select the host node? Are these features easily configurable?; Is the knowledge used to take the replication decisions global to the network or local to a node or group?; Are the communication channels assumed reliable?; What is the cost of the solutions provided?

3.2.1 Hamdy et al.: Service Distribution Protocol

In [59] [58] [60], the *Service Distribution Protocol* is presented as an approach to increase the **service availability** in MANETs. This protocol takes into consideration the interest that exists in the network on a specific service to replicate it. The interest of a service is measured as the number of request regarding service during a predefined time interval. The replication mechanism works as follows: the client node counts the request over a specific service, if it reaches a specific threshold, it will ask the server for a replica of the service. Moreover, Hamdy et al. also proposed a **hibernation mechanism**, in order to hibernate a replica when the interest in its use decreases. In the evaluation of the proposed protocol, it is used the Random Waypoint (RWP) mobility model. The synchronization management of the service/replicas is not considered. In [60] a comparative study of the proposed protocol with and without hibernation mechanism is performed. The results obtained in the study show that the success ratio, defined as the number of the successful service requests, is slightly higher without the hibernation mechanism, although this difference disappears for large networks, where the network partitions are reduced owing to the increment of nodes and links. However, the election of the host node is only performed on the basis of the interest, so it is not taken into consideration any of capabilities of the nodes or if the service is available on some neighbour node. Moreover, to provide service or replicate a service any cost has not been considered (e.g., energy, bandwidth or storage), therefore the availability of the service is incremented, but in the study has not been taken into consideration any measure of the cost and thus the efficiency of the solution cannot be measured.

3.2.2 Sun et al.: Minimum Access Cost replication strategy

The dynamic *Minimum Access Cost* (MAC) replication strategy is presented in [153]. This strategy is designed to operate on hierarchical unstructured overlay P2P networks, consisting of two types of nodes, peers, and super peers. Super peers are responsible for administrating the peers of its domain. The replication is made on the basis of the access cost: if the average response time for any Logical Resource exceeds the threshold set, the super peer will create more replicas of the Logical Resource in an appropriate location. In this work, a **utility function** is used, called *Replication Factor*, to know the suitability of the peer to host a replica of the Logical Resource. It is defined in Equation 3.1.

$$RF = RV \cdot (\frac{Memory}{filesize}) \cdot AvgBW \cdot SR \tag{3.1}$$

Where *RV* is the reliability of the peer, which is calculated as shown in Equation 3.2.

$$RV = \frac{online\ time}{total\ time} \tag{3.2}$$

where:

Memory	= represents the available memory of the node
filesize	= represents the size of the Logical Resource to be replied
AvgBW	= represents the average bandwidth of the peer
SR	= represents the service ratio of the peer, defined as $\frac{the\ requested\ processed}{all\ the\ requests\ received}$

This information is mainly local to the peer (node) and could be easily obtained in a MANET to know how adequate a node to host a Logical Resource is. In the simulation, the Grid Simulator OptorSim [16] is used with a wired and fixed network of static nodes. The super peer performs the replication process, and it follows a bi-directional-linked list based replica location scheme (BLL-RLS) to get a global replica view and to obtain the replication cost globally to design the optimal replication strategy. The proposed replication strategy is focused on wired, static P2P networks, and on the basis of global knowledge about the network topology and behaviour. Moreover, the network is considered stable and reliable.

3.2.3 Ahmed et al.: Distributed Adaptive Service Replication

In [2], a distributed service replication scheme for MANETs is proposed, named *Distributed Adaptive Service Replication* (DAR), which is later improved in [3] and renamed as *Highly Distributed Adaptive Service Replication* (HDAR). The scheme proposed is based on clustering. The clusters are formed by one-hop neighbours, where the node with the minimum moving speed is elected as the cluster head, while the rest of them are designated as cluster members. The nodes that are neighbours to other clusters are designated as gateways. Each server can serve multiple clusters. Once that clusters are created, the cluster head (i.e., the node with the minimum movement speed) receives all the service requests from the member of its cluster. Servers exchange some information among them in order to know the current service demand in the network. In dynamic network topology environments, to exchange this information among all the servers of the networks is not suitable. Therefore, HDAR introduces, with regards to DAR, an r-level coverage confirmation mechanism (where r refers to the number of hops), in order to limit the communication between distant servers. Both DAR and HDAR base its replication on the service interest. This is, the replication of a service in a determinate cluster will depend on the number of request from the cluster and in the number of active nodes in the cluster. This information is maintained for each server. Additionally, HDAR introduces a new factor: the trade-off between communication cost and replication cost. Therefore, a service will be replicated in a cluster if (1) the service is highly demanded in that cluster; or (2) the replication cost is less than communication cost to access to the resource remotely. The architecture proposed is simulated using the QUALNET simulator and under two mobility models: RWP and Realistic mobility model generated by MobiREAL[1]. The replication trigger is not dynamic, it is statically determined. This means that the replication scheme is not dynamic or reactive to run-time events (e.g., changes in the network topology). If a service

[1]http://www.mobireal.net/

is replicated to a cluster, it is allocated in the cluster head. Additionally, the cluster head is also a node router for the requests from cluster members to services that are allocated out of the cluster. Thus, a cluster head can become a bottleneck easily and its resources (energy and bandwidth) can be easily depleted. The cost of the replication scheme is measured in terms of energy. The synchronization of the replicas is not addressed in the proposal.

3.2.4 Bellavista et al.: REDMAN middleware

The REDMAN (Replication in Dense MANETs) middleware is proposed in [17], in order to manage replicas of data/service components in a dense MANET. In this work, a dense MANET is defined as a MANET that *"includes a large number of mobile wireless devices located in a relatively small area at the same time* [...] *(and) has a node density, defined as the average number of wireless nodes at single-hop distance from any dense MANET participant, that is almost invariant during a long time"* [17]. To determine if a node belongs to a dense MANET, it is proposed a decentralized protocol, **in which each node determines autonomously if it is in a dense MANET by mean of the number of one-hop connections**. In this way, if this number is greater than a predefined threshold it will consider itself to be within a dense MANET. According to the authors, a threshold between 10 and 20 single-hop connections is enough to assure enough number of alternative connectivity links. Once the dense MANET is established, a Replica Manager node is elected, as a central entity to manage the replica distribution and management. The objective is to select a node that minimizes the number of hops required to reach the farthest node from its position. To this end, it is proposed a protocol that explores as candidates only a subset of the nodes of the MANET. The basic idea is, once evaluated one node, to select as the next node to be evaluated that which is in the direction of its farthest node. According to this heuristic, the exploration protocol will evaluate the nodes in direction to the centre of the MANET, where the optimal node is located, according to the selection criteria. This election process is triggered each time that the current manager node realizes it is going to exit or abruptly fail. The protocol is designed for dense MANET, where, according to the authors, the frequency of node entering/exiting dense regions and their speed is low. Therefore, it is assumed that all nodes placed along the path from the node that it is being currently evaluated to its farthest nodes remain within the communication range during the protocol round. Once the Replica Manager node has been elected it start to distribute the replica along the MANET. The replication distribution is performed on the basis of pre-established target replication degree and replication hops. In this way, the replicas are distributed according to a gossip-based strategy. This consists in randomly send a replication token message, and if the nodes that receive it can host a replica of the service, the node will host it, the remaining replica count will decrease, and the message will be forwarded until the replication degree is committed. To evaluate the proposal the ns-2 simulator has been used, and the nodes have been statically deployed randomly in a square area. The author assumes that the replicate resources are read-only, thus to address the synchronization of the replicas is not necessary.

3.2.5 Dustdar et al.

In [49], a system for replication and synchronization of stateful Web services[2] in MANETs is presented. The system is based on a copy-primary replication system of stateful Web services, synchronizing the replica set by means of the *Simple Replicator Protocol* (SRP). The replication mechanism is based on a global view of the network, where the nodes know all about the other nodes in the network (their properties and their hosted services). Moreover, a **hibernation mechanism** is proposed, where the replicas that are no longer needed are disabled instead of deleting them completely. In the system proposed, the Monitor plays a fundamental role. The most powerful nodes are elected as monitors, although the specific features considered for a node to be powerful are no defined. These are responsible for partitioning the network, by node IDs, and check them periodically. The periodic checks are performed to know if they are still available, their state has changed and what properties they have. In the work, this technique is named by the authors as *active monitoring*. The communication necessary for monitoring is done by SRP. After each monitoring event, a replication event is performed, where it is checked if it is necessary to deploy or remove service replicas. This process is divided into two phases: (1) leader election and (2) placement logic. The leader election is done on the basis of the information stored by the monitors and gathered during the monitoring process. If it is necessary to deploy only a replica service, this will be deployed in the own monitor node. If it is necessary to deploy more replicas, these are deployed in popular leaders. The proposed system has been evaluated in networks consisting of up to 140 nodes. The simulator used is not determined in the paper. The nodes were statically deployed and disabled randomly, but in a balanced way, in order to simulate failures. The synchronization is not contemplated in the evaluation. In this work, monitor nodes can become easily a bottleneck, as they are in charge of monitoring the network periodically and host the service replicas when there is only one replica to deploy.

[2]Stateful services are those which store or maintain persistent data along time [51].

3.2.6 Kim et al.: Scalable Replica Allocation scheme

In [86], a *Scalable Replica Allocation* scheme (SRA) is proposed to address the communication cost in MANETs. The replica allocation scheme consists in two phases: (1) first, the nodes are grouped according to their speed and in groups of one-hop distance; (2) the node with less speed is elected as the cluster head. This node will allocate the data items according to its frequency access, based on a greedy approach. The replicas will be allocated, first, in the cluster heads and then in the rest of the nodes of the group following a round-robin approach according to the IDs of the nodes (the lowest ID node first). Data items are not updated and the replicas are reallocated periodically. It is assumed that each node is aware of its own speed and that to calculate this has not any additional cost. Additionally, a mobility-based allocation method is proposed, in which the nodes with speed greater than a certain threshold are excluded as host candidates. The communication and replication cost are measured in number of hops. The system has been tested on ns-3 (v3.10), and the nodes follow the RWP mobility model. **In this work, the link problems, as fading, multipath effects, congestion or interference are taken into consideration**. The data availability is measured, as the ratio of successful queries and the communication cost as the number of hops. However, the proposal is not reactive to changes in the context, the replicas are reallocated periodically, but no when a node crashes, leaves or joins the network or the topology changes.

3.2.7 Choi et al.

In [80][138], a set of data replica allocation techniques for MANETs with **selfish nodes** are presented. It should be noted that selfishness is only considered in data storage, not in data forwarding. They consider three types of nodes in the network: *nonselfish*, *fully selfish* and *partially selfish* nodes (i.e., they only provide a part of its available memory to allocate data for other nodes). The strategy proposed is made up of three phases: (1) detecting selfish nodes; (2) building a *self-centred friendship* tree (SCF) and (3) place data replicas. The degree of selfishness of a node is calculated as an approximation of the total storage space of a node against the space provided to store data replicas for other nodes. Once that each node calculates the degree of selfishness, each node builds a self-centred friendship three of deep d, where d represent the hop distance of the tree. This tree is built on the basis of selfishness degree, removing paths containing nodes with high selfishness degree. Regarding replica allocation, four schemes are proposed:

- *SCF-tree-based replica allocation*: a *greedy-based* replica method.

- *SCF-tree-based replica allocation with degree of selfishness*: the more frequently accessed data items are allocated on less selfish nodes.
- *SCF-tree-based replica allocation with closer node*: data replicas are allocated in the closer nodes in the tree.
- *Extended SCF-tree based replica allocation*: selfish nodes are included in the tree, although, the replica allocation grants preference to non-selfish nodes. This last technique shows the best performance of query delay.

The different proposed techniques are simulated, but none simulator is specified. The movement model used is the RWP mobility model. There are measured four performance metrics: *overall selfishness*, *communication cost* (hop count), *average query delay*, and *data availability* (authors refers to this term as *data accessibility*). However, although selfishness is taken into consideration in this work, it is calculated with data that the own nodes have to provide, so a selfish node can lie about its available memory.

Lately, in [139], an improvement on the previous proposal [80][138] is described, where the update of the replicas is taken into consideration. It is assumed a known and fixed updated period for each data item. The update only can be performed by the owner of the data item, and when the data item is updated, all of its replicas become invalid. The update value is taken into consideration to allocate the replica and to build the tree (the SCF tree is renamed as USCF), adding this value to the formula together the selfishness degree and hop distance. In this way, for each node to allocate near it the data items more frequently accessed and with a low data update period is proposed. To consider the data update frequency to allocate replicas is an interesting approach. However, only static and known data update frequencies have been considered, which could be challenging to translate into a real-world scenario. Moreover, with dynamic data update frequencies, the USCF tree will have to be rebuilt continuously, according to the data update frequencies change.

3.2.8 Kumar et al.

In [91], an improvement is presented on the replica allocation algorithm proposed on [80]. The original replica allocation algorithm is based on an SCF Tree. However, according to the authors of [91], the construction of this tree is expensive, especially on high dynamic MANETs. Therefore, **it is proposed to use the information gathered by the routing protocol** (i.e., the information on the routing tables) **to obtain information about the network topology** and allocate the replicas avoiding to build the SCF tree. In this work, the *Fisheye State Routing* *protocol* (FSR) [129] is used. The data items are allocated taking into consideration the hop distance, provided by FSR, and the selfishness of the nodes in a similar way that in [80]. The data items are not updated, and the replicas are relocated in specific periods. In this work, to evaluate the proposal, a mathematical analysis has been performed using MATLAB. However, the mobility model used is not specified.

3.2.9 Zhang et al.

In [178], a study about data replication in mobile tactical networks is presented. The work highlights how the motility models in this context are strongly defined and how this affects the data replication strategies. There are presented three common tactical manoeuvres in battlefields:

- *Repeated Traversal (RT)*: in which nodes are organized as squads, which advance in a staggered way.
- *Bounding Overwatch (BO)*: which presents a more restricted mobility, where several nodes compose one or two squads.
- *Pincer (PI)*: which presents a high dispersion degree between squads.

Assuming the authors that data are not updated, (i.e., the synchronization of the replicas is not taken into consideration), four data replication schemes has been studied over the proposed mobility models:

- *Best-Location Intra-Group Data Replication*: the best location to place the replica is that in which the data redundancy and data access cost are minimized. This is calculated as the node with the higher access frequency to the data in the group, against the replication cost for that node (how the replication cost is obtained is not specified).
- *Greedy Data Replication* (Inter-Group Data Replication): each node replicates the most accessed data.
- *Pairing Cooperation Data Replication* (Inter-Group Data Replication): each node collaborates with its neighbour to agree in which data should be replicated.
- *Reliable Neighbouring Data Replication* (Inter-Group Data Replication): as in the *Pairing Cooperation Data Replication*, but increasing the number of collaborators. The replication decision is made on the basis of data access frequency.

As metrics that can be remarked, we can find the Average Availability of the data and the Average Access Delay. The authors have developed a simulator based on CSIM 19 to evaluate the different proposals. The communications are considered reliable while the nodes are within a communication range. After the performing of the simulations, it is concluded:

- Pairing scheme (3) performs better in the scenario of RT, because of the stability of the small squads.
- Neighbouring scheme (4) performs better for BO scenario, because of its major stability (bigger squads, i.e., one or two).
- Greedy scheme (2) for PI, because of its major dispersion.

This work underlines the broad impact that the mobility model can have in the performance of the replication strategy. However, the only parameter to take into consideration to elect a host is the interest of the node in use the replica.

Later, in [177], *Greedy Data Replication*, *Pairing Cooperation Data Replication* and *Reliable Neighbouring Data Replication* schemes are evaluated under Random Walk, RWP, Manhattan Grid and Reference Point Group Mobility (RPGM) mobility models. The work is also extended, considering a dynamic assignment of memory from the nodes to store replicas according to link stability with the neighbouring. The more stable is the link, the more memory is dedicated.

3.2.10 Barolli et al.

In [13], a data replication system based on a fuzzy logic approach is proposed. The objective is to guaranty the **QoS requirements** of the data replicated. The metrics that are determined in this work, to be measured in order to establish the efficiency of the system are: Local Node Density (LND), Maximum Data Accessibility Hop Number (MDAHN) (this is not addressed in [13], [43] nor [75], and the authors leave it for future work), Bounded Retrieval Time (BRT) and Minimal Number of Replicas (MNR). The main component of the data replication system is the Fuzzy Logic Controller (FLC). It is composed of the *fuzzifier*, inference engine, Fuzzy Rule Base (FRB) and the *defuzzier*. The evaluation of the proposed system has been done with MATLAB. This work has been subsequently continued in [43] and [75]. The replication system is applied to **opportunistic networks on IoT environments**, and in the node election node speed and node density are take into consideration. The probability of a node to be selected increases with the node speed and is reduced with high node density, as there are more nodes in the vicinity to act as leaders.

3.2.11 Xia et al.: ComPAS data replication method

An extended version of ComPAS, a data replication method to improve data accessibility in ad-hoc social networks, is presented in [173]. The replication method is based on a greedy approach. For the placement of the replicas, the storage space and the cost of reading/writing operations have been considered. This cost is set to 0 if the data replica is in the same community (group) and to 1 if this data is in a different community (group). The proposed replication method has been evaluated with the Gephi[3] tool, which is used for visualization and analysis of wide graphs. The replica is allocated in the node with most storage within a community (group).

3.2.12 Shi et al.: RHPMAN data replication scheme

In [149], a data replication scheme, denominated RHPMAN (*Replication in Highly Partitioned Mobile Ad-hoc Network*), is presented. It is assumed that there is not communication fading, therefore, if two nodes are within their communication radio, they can communicate. In the RHPMAN scheme is devised for networks where there is a set of stable large network groups and nodes with high mobility that travels across these groups. While the data replicas should be located in the nodes of the large partitions (i.e., nodes with a stable neighbouring topology), **the nodes with high mobility can be used as a transporter to carry information/replicas from one network partition to another**. Authors state that, the *"movement based on human decisions and socialization behaviours improve connecting opportunities between a pair of nodes in the intermittently connected network"*. They do not consider in this work selfish or malicious nodes. The nodes, periodically, calculates and exchanges its delivery probabilities based on local profile information. In this work, the **change degree of connectivity** is considered in order to know if a node has high mobility and can be used as a carrier to get the replicas to the other network partitions. Moreover, this degree is also used for the election of a replica host node, in order to know if it belongs to a stable network group. A node can become a host if it is the most powerful (energy and storage space) and stable of its neighbourhood. Each node compares its attributes with his neighbours, limited by h hops. **The routing protocol *DSDV* [130] is used to get the information about the network topology**, by means of the routing tables. The owner of the data to be replicated is who initiates the replica distributing process. The status of the replicas is not maintained, owing to trace and maintain this information can cause network overhead, according to the authors. Regarding the synchronization and consistency of the data, they only consider a weak consistency model, where only the owner of

[3]https://gephi.org

the original copy of the data can update the data items. They have used the OPNET simulator to evaluate the replication scheme. Regarding the mobility model, they have divided the mobility area into four sub-areas. In each of them, there is a group of nodes that follow a Random Walk Model. Moreover, there is a group of nodes, which are able to move in all the area (they travel between groups) that also follow a Random Walk Model. In this work, it is assumed that the network topology does not change while a data item is being transferred from a node to another. This is a strong assumption that can be difficult to achieve in real-world scenarios. The replication is performed periodically. This is, the proposal is not reactive to events in the context. Finally, the data availability is measured, but not data replication or replica host costs are taken into consideration.

3.2.13 Hirsch et al.

In [66], it is presented an adaptive cooperative caching approach to reduce the energy consumption and traffic congestion. This approach denominated *Tidal Replication* is based on a flooding scheme, in which the replication is performed in the network zones where the data becomes popular, and it is gradually de-allocated when its popularity decreases. The host candidates are selected according to the bandwidth utilization. Additionally, a distribution approach, named Magnetic Distribution, is proposed, which pretends to dynamically redistribute cached items in a more centralized host within the network according to its demand. In the work, both the replication and distribution of the cached items are taken into consideration, but cache consistency is not. The replication decision is made using global network information about the data access, and according to the authors, this is the main advantage of the work. The decision about the replication of the item is controlled by the Data Item Control Node (DICN). However, is not specified how the DICN is elected. The replication of the item is made towards the dominant requesting path. When a node decides to replicate an item, this sends a replication request to the first node in the Dominant Request Path (DRP). This node will evaluate if it can allocate the item (i.e., to check its storage availability), or according to the request matrix redistribute that replica to the next node in the DRP. The algorithm will stabilize when in a node all request paths have a balanced request count. If the demand for this data item is persistent in time, the DICN will delegate its responsibility in a new node, more centralized within the zone of the network from which the demand comes. The proposals have been simulated employing their own simulator. However, communications have been assumed reliable. Regarding the mobility model, a modified version of the RWP mobility model has been used. The efficiency measures taken are: Mean Query Response Time, Mean Query Hop Count, Energy Savings and Mean Storage Utilization Ratio.

3.2.14 Hara et al.

In relation to data consistency in MANETs, in [63] a study of different consistency levels and protocols to achieve them is presented. The protocols are applied to a structured network divided into peers and proxies. The proxies are used as region headers, responsible for managing the peers of its region. It is assumed that the proxies have limited movement and never leave their own region. Moreover, the role of each node (peer or proxy) is assumed, i.e., these roles are assigned statically, and every proxy knows replicas held by peers in its region. Under *Global Consistency Protocol*, proxies are responsible for coordinating the write operations performed by the peers over the network. Regarding the replication strategy, to use *square-root allocation* (SRA) and inversed SRA are proposed. These are strategies used in unstructured P2P networks, where the data are replicated on the basis of query frequencies [37]. However, there are significant differences between P2P systems and MANETs. Specifically, P2P systems are generally designed for the Internet, where there is an efficient IP routing mechanism, and the peers are frequently static, with sufficient resources and highly reliably links [99]. Therefore, such replication strategies are based on global knowledge of the status of the network and its topology, which can be easily obtained in stable P2P networks but it is hard and expensive to obtain in dynamic topology and resource-constrained environments as MANETs. The consistency protocols have been tested under the RWP mobility model and Random Walk mobility model. In this work, the creation of the structured network is preconfigured, and dynamic configurations, i.e., dynamic election of proxies and region configurations are avoided.

Related to previous work, the same authors of [63] present some data replica allocation methods in ad-hoc networks [62]. These methods are based on access frequencies to data items from each host and known update periods for each data. Three different methods are proposed:

- E-SAF, where a node hosts a data replica according to the average number of access requests. This value is denominated PT.
- E-DAFN, where the data replication between neighbours is removed, i.e., if two neighbours have the same data replica, the one with lower PT will remove it.
- E-DCG, where data replication is removed in stable node groups.

The node groups are calculated as biconnected components of the network (i.e., a group of nodes that is not divided if an arbitrary node of the group is removed). Regarding the result obtained in the work, E-SAF provides the lowest traffic, although a higher replication of the data exits, while E-DCG provides the highest data availability. The data relocation periods are fixed and determined by a system administrator. Finally, it should be noted that in this work it is assumed that there are no changes in the network topology in the executions of the E-DAFN and E-DCG methods.

3.3 Leader election algorithms in dynamic environments

In this section, election algorithms of general purpose are revised. These election algorithms can be adapted or, in some cases, directly applied to elect a host node in a service replication process. The traditional election algorithms presented in Section 2.4 were devised to operate in distributed systems where no communication failures were assumed to happen. The proposals selected in this section are devised to operate in dynamic environments.

3.3.1 Malpani et al.

In [98], two leader algorithms for MANETs are presented. The algorithms are based on TORA routing protocol [128]. In the first of the algorithms, that only can occur a single topology change at a time is assumed. This is, the network is completely recovered when a link changes before another link does it. For the second of the algorithms, this assumption is not taken. **The changes in the network topology are detected through TORA routing protocol**. In the proposed election algorithm, the node that detects a partition elects itself as the leader of the new network partition. It transmits this information to its neighbours (one-hop distance nodes), who propagate it. If two or more network partitions merge, the leader with the smallest ID will become the leader of the merged network. The correctness of the first of the algorithms is formally demonstrated, understanding as correctness: (1) eventually there is a leader in the network; and (2) there should never be more than one leader. However, it is assumed for both cases that the links are bidirectional, FIFO and reliable.

3.3.2 Vasudevan et al.

Secure Extrema Finding Algorithm (SEFA) and *Secure Preference-based Leader Election Algorithm* (SPLEA) are proposed as secure leader election in MANETs [163]. Whereas SEFA is based on a *Common Election Algorithm* (CEA), in which all the nodes use the same **utility function**, i.e., all of them will obtain the same value for a candidate node, SPLEA uses a specific utility function for each node, which represents the preference of the individual node. In this way, SPLEA can evaluate features as the distance from the leader that cannot be taken into consideration in SEFA. Both algorithms are based on a voting approach. However, for these algorithms is assumed, that the communications are secure and reliable and that the nodes are reliable. Moreover, that the network topology remains static for the duration of the election is assumed. There are a third algorithm, *Asynchronous Extrema Finding Algorithm* (AEFA), that unlike SEFA and SPLEA, is a non-secure election algorithm that allows changes in the topology during the election. However, bidirectional and FIFO links and reliable node-to-node communications are assumed. The election is triggered when a node is disconnected from its leader. Such election is based on the termination-detection algorithm for diffusing computations of by Dijkstra and Scholten [47].

In [164], the work presented in [163] is improved, proposing a diffusing computation-based leader election algorithm for MANETs able to operate in dynamic network topology environments. The objective of the work is to elect a leader in a network partition according to some system-related capabilities (such as battery life or computation power). In the work, bidirectional and FIFO links and a weak reliability in the node communications are assumed. This is, *"the message delivery is guaranteed only when the sender and the receiver remain connected for the entire duration of the message transfer"*[164]. Thus there have not been considered other features that can affect the message delivery, such as the bandwidth saturation or channel fading. The algorithm is based on the construction of a **spanning three**, to create a hierarchical communication structure. Its construction is started by the first node that detects the departure of its current leader. This node will remain as the root of the spanning tree, and the growing process will be propagated to its neighbours. Once the structure is constructed, the root node will select the most valuable node and will spread this election to the rest of the nodes through the spanning tree. The algorithm has been simulated with GloMoSim, under the RWM model. In [96], a formal analysis of the election algorithm proposed in [164] is carried out, where it is concluded that owing to the mobility of the nodes during the discovering process and the temporary disconnections that this causes, some nodes can be left out of the election protocol.

Vasudevan algorithm

The algorithm proposed by Vasudevan et al. is based on three main kinds of messages:

- *Election message*: to start the election and grow the spanning tree. The node that initiates the election is defined as the root.
- *ACK message*: is used to reply to *Election* messages once that the tree has completely grown. They are used to shrink the tree and gather information about the best node in the tree or subtree.

- *Leader message*: is used by the root node to announce the leader, once that the tree has completely shrunk, and retransmitted by its child until the leaves nodes.

On the basis of this messages, the algorithm performs an election as follows:

(A) **Supposing a node *N* that initiates an election**:

1. *N* send, as root node, an *Election* message to all of his direct neighbours (one hop distance), now designated as children.
2. *N* waits for the *ACK* replies of its children nodes.
3. Once that *N* receives all the *ACK* messages it compares the information received in each of them to discover who it is the best node in the tree.
4. *N* announces the winner of the election by means of a *Leader* message, which is sent to its children nodes.

(B) **Supposing a node *P* that does not initiate the election**:

– ***P* receives an *Election* message from *Q***:
  1. *P* knows now that an election process has started. Thus, it designates *Q* as its parent node in the tree.
  2. *P* retransmits the *Election* message to its direct neighbours.
  3. *P* waits until it has received all the *ACK* replies.
  4. Once that *P* has receives all the *ACK* replies, it checks who is the best node in its subtree and replies to *Q* with an *ACK* message adding this information.
  5. *P* waits for a *Leader* message.

– ***P* receives an *Election* message from *Q* and *P* has not child nodes**: *P* is a leaf node, it replies directly to *Q* with an *ACK* message adding its own information as best node of this subtree.

– ***P* receives an *Election* message from a node that is not its parent**: This message is usually sent by a sibling node. Therefore, *P* replies immediately with an ACK message with no additional information.

– ***P* receives a *Leader* message**: *Leader* message contains the information about the node that has won the election (this node could be *P* itself). Thus:
  1. *P* sets the winner node as the current coordinator.
  2. *P* retransmits the *Leader* message to its children nodes, if there are any.

Additionally, to this basic messages, Vasudevan defines periodic messages to know the state of the network, such as *Heartbeats* messages from the leader, in order to know when the leader fails; and *Probe* and *Reply* messages between the nodes in order for a parent to know when a child is no longer available and do not wait for its *ACK* message.

Moreover, *Election* messages have a unique ID, identifying the election. In this way, to avoid spread concurrent elections, Vasudevan proposes that if a node receive *Election* messages from different elections, the node will spread only that with higher ID, i.e., the newest one.

3.3.3 Park

A variation of the Invitation algorithm of Garcia-Molina [53] is presented in [127]. It is aimed to resolve the election problem in asynchronous distributed systems. The main difference of this proposal with the original algorithm is that in the original algorithm any node can trigger the election if it detects that the current leader has crashed. However, in the new proposal, the algorithm can be only triggered when all the nodes agree that the leader has crashed. The communications between nodes are assumed to be reliable. The algorithm is not simulated, but its properties (safety and liveness) are formally demonstrated correct.

3.3.4 Brandner et al.

An election algorithm based on a probabilistic approach is presented in [24]. The idea is that when a node needs to another node of the network to perform some task, it will send a request message to all the nodes of its neighbourhood. Then, the nodes will respond with a probability p_{ij}, where i is the requesting node and j the time slot. If only one node replies the request, it will be the elected node, if not, the slot will continue advancing until a non-colliding message happens.

3.3.5 Raychoudhury et al.

A k-leader election algorithm for MANETs is proposed in [134]. In its work, **the author claims that there are no k-leader election algorithms for MANETs or algorithms based on weights** (i.e., all proposed election algorithms takes IDs as election criteria). The connections are assumed to be reliable, always that two nodes are within the connection range and it is assumed that the nodes behave correctly. The k-leaders are elected on the base of a weight, that represents the node capabilities, and each node can have different weights, according to the desired capability in the election. The weight ties are broken using the IDs of the nodes.

To perform the election, the nodes with highest weights among their two-hop neighbours are elected as RED nodes. The rest of the nodes of the group are denominated WHITE nodes. The RED nodes start a diffusing computation building a diffusion tree, which collects the weights of all the nodes in the network. Each diffusion tree only contains one RED node. Once all weights are collected, the RED nodes choose the K highest weight nodes as leaders. The election algorithm is simulated under the RWP mobility model. The simulator is not specified, and the weights are randomly assigned at the beginning of each simulation. In the simulation are evaluated the fraction of time without k-leaders, the election rate (the average number of election in which a node participates in per unit of time), the election time and the message overhead (average number of messages sent by a node per election). The architecture also is evaluated in a testbed, but with a static network.

3.3.6 Sabat et al.

In [140], a leader election algorithm for MANETs in the presence of selfish nodes is presented. The objective of the election is to run an *Intrusion Detection Service* (IDS)[4]. The authors have proposed an adaptive energy aware reputation system model. In this way, when a node acts as server, it is monitored periodically by some nodes of the network. If this node runs correctly the IDS, the node will gain reputation. If the reputation of a node falls from certain threshold, the node will not be served. Therefore, all the nodes will need to act as servers, in order to be able to be served when they become clients. In this way, it is guaranteed that nodes will avoid the selfish behaviours or lie about their energy levels. **Moreover, the nodes adapt its connection range to the size of the cluster, in order to save battery**. The proposed election algorithm is cluster dependent, but the clusters are assumed to be already formed. The utility function takes into consideration the energy level of the node and its reputation, to allow the nodes with low energy levels to be elected and obtain reputation. The system is tested on ns-2, with the RWP model. However, the simulation time is only of 30 seconds.

3.3.7 Mohammed et al.

In [103], two possible election algorithms are proposed: *Cluster-Independent Leader Election* and *Cluster-Dependent Leader Election*. In the first one, the leader is elected by the votes of its neighbour nodes (two-hop). In the second one, the leader is elected after the creation of the cluster in the network. In this work, synchronized clocks between the nodes are assumed. The

[4]IDS are used in network-based systems (e.g., Cloud) to detect anomalies or known attacks in the system and notify network administrators [61].

system is tested under the ns-2 network simulator, for a static network and a dynamic network under the RWM. The election models follow a voting approach, and the energy level of the node is taken as election criteria.

3.4 Cluster Head election approaches

In large-scale systems [34] scalability is one of the primary objectives for architects. Node cluster methods [175] are applied to turn a distributed network into a set of interconnected local clusters that can be handled individually, like a centralised network. In this way, the management of the network is simplified, achieving scalability under a *“divide and rule"* approach. The use of a cluster helps to reduce the redundant message transmissions. However, the major challenge in the implantation is how to select the cluster heads (CH) dynamically.

3.4.1 Wu et al.

In [172] an approach to constructing a cluster-based two-layer hierarchy is proposed, to perform a consensus protocol. In this work is assumed that number of hosts is static and also known by all the hosts of the group. Moreover, the communications are assumed to be performed under a reliable channel, where there is no message loss. The election of the CHs is performed through the flush algorithm [35], in which the host selection is performed on the basis of trust, to elect the most reliable host. To evaluate the approach, its authors have used their own simulator. The simulations have been performed under the Random Walk mobility model and the RWP mobility model.

3.4.2 Torkestani et al.

A cluster formation method based on the mobility of the nodes in a MANET is presented in [4]. The main contribution of this method is that an indeterministic behaviour about the mobility of the nodes is assumed. It proposes a learning automata-based weighted clustering algorithm, in which each node bases its decisions from the local information received from the one-hop neighbours. The node elected as CH is the one with the minim expected relative mobility, concerning its neighbours. The clustering algorithm is simulated under ns-2, and it is assumed that each node is aware of its mobility information using a reliable positioning system. The cluster lifetime and the control message overhead are used as measures of efficiency. The RWP mobility model is used to evaluate the proposal.

3.4.3 Venkanna et al.

A CH election algorithm, denominated TEA-CBRP, is presented in [165]. Its objective is to improve a *Cluster Based Routing Protocol* (CBRP) routing algorithm for MANETs [161]. To elect a node as CH the following parameters are taken into consideration: trust value, remaining energy level and time of availability (calculated as a prediction of disconnection, taking into consideration, connection range, speed, and direction). Once the clusters are formed, the node with lowest ID is elected as temporal CH. This node will evaluate the nodes of its cluster and will elect the best-ranked node. This node will be elected as main CH, and the second best-ranked node will be elected as second CH. The objective of the second CH is to support the maintainability of the cluster. The primary objective of the TEA-CBRP routing protocol is to avoid elect malicious nodes as CH, and therefore jeopardizing the routing. The proposed routing algorithm has been tested on the ns-2 simulator, under the RWP mobility model. The metrics that have been measured are packet delivery ratio, average end-to-end latency, routing packet overhead and the number of CH changes.

3.4.4 Benkaouha et al.

In [18], an implementation of a *Stable Storage* (SS) system is introduced, in the context of a check-pointing protocol for MANETs. Specifically, the protocol ARISS (*Area Repartition for Implementing a Stable Storage*), based on a geographical clustering of the nodes, is proposed. The work assumes that each node has a GPS system, the channels of communication are reliable and that the network cannot be partitioned. ARISS uses a mobile agent based zone repartition, where a node gathers information about the position of the rest of the nodes (global knowledge of the network) and calculates the gravity areas using the Euclidian distance for each zone. ARISS works with only four zones. This gravity point is recalculated periodically by the servers of the zones. As the server for each zone, the closest node to the gravity point is elected. This server will help the other nodes, by hosting a mobile agent to build their SS. If the server node moves away, the mobile agent will move to the new server of the zone. The protocol has been tested on the ns-2 network simulator, under the RWP mobility model.

3.5 Discussion

From the the proposals above studied, three tables (Tables 3.1, 3.2 and 3.3) have been created as an overview of their main features. Table 3.1 summarizes the approaches to improve data/service QoS through dynamic replication (Section 3.2), Table 3.2 presents the leader

Table 3.1 *Overview of the approaches to improve data/service QoS through dynamic replication and their features. Source: Own elaboration.*

	Objective (1)	*Reliable Communications* (2)	*Simulated* (3)	Simulator	*Mobility Model* (4)	*Clustering* (5)	*Election Criteria* (6)	*Reactive* (7)	*Knowledge Scope* (8)	*Cost Measure* (9)	*Synchronization* (10)
[59][58][60]	$+a$	?	✓	?	RWP	✗	Int.	✓	🏠	✗	✗
[2]	$+a$	?	✓	QualNet	RW/Map-based	Pos.	Int.	✗	🌐	🔌	✗
[3]	$+a$	?	✓	QualNet	RW/Map-based	Pos.	Int./RC./CC.	✗	🏠	🔌	✗
[17]	$+a$	✓	✓	ns-2	St.	✗	G.	✓	🌐	✗	✗
[13][75]	$-rt$	?	✗	MATLAB	?	✗	Dens./RT.	?	🌐	⏲⚖	✗
[43]	$-rt$	?	✗	MATLAB	?	✗	Dens./RT./Spd.	?	🌐	⏲⚖	✗
[86]	$-cc$	✗	✓	ns-3	RWP	Spd.	Spd.	✗	🏠	📶	✗
[80][138]	$+a$ $-rt$	?	✓	?	RWP	✗	Self./G./Dist.	✗	🌐	📶	✗
[139]	$+a$ $-rt$	?	✓	?	RWP	✗	Self./G./Dist./Upd. Freq.	✗	🌐	📶	✗
[91]	$+a$ $-rt$	?	✗	MATLAB	?	✗	Self./Dist.	✗	🌐	✉	✗
[178]	$+a$	✓	✓	CSIM	RT BO PI	Phy.	Int.	?	🏠	⏲	✗
[177]	$+a$	✓	✓	CSIM	RW RWP MG RPGM	Phy.	Int.	?	🏠	⏲	✗
[49]	$-rt$	?	✓	?	St.	ID	Perf.	✓	🌐	📶	✓*
[173]	$+a$	?	✗	Gephi	RPGM	?	Sto.	?	🌐	✗	✗
[149]	$+a$	✓	✓	OPNET	RW	✗	Perf./Spd.	✗	🏠	✗	✗
[66]	$-e$ $-nt$	✓	✓	?	RWP	✗	BW/Int.	✗	🌐	🔌⏲🗄	✗
[62][63]	$+a$	✓	✓	?	RWP RW	St.	Int./Pos.	✗	🌐	✗	✓**

1 ($+a$) To increment the availability; ($-rt$) To reduce the response time; ($-cc$) To reduce communication cost; ($-e$) To reduce energy consumption; ($-nt$) To reduce network traffic.

2 (✓) communication is explicitly assumed reliable; (✗) communication is explicitly assumed not reliable; (?) nothing is said about communication reliability.

3 (✓) the proposal is simulated; (✗) the proposal is not simulated (note that MATLAB and Gephi are not considered simulation tools)

4 (RWP) Random Waypoint mobility model; (St.) nodes are static; (RW) Random Walk mobility model; (RT) tactical Repeated Traversal mobility model; (BO) tactical Bounding Overwatch mobility model; (PI) tactical Pincer mobility model; (MG) Manhattan Grid mobility model; (RPGM) Reference Point Group Mobility model

5 (✗) clustering is not used; (Pos.) position-based clustering; (Spd.) speed-based clustering; (Phy.) cluster are physically delimited; (ID) ID-based clustering; (St.) statically predefined clusters.

6 (Int.) interest in use the resource; (RC) replication cost; (CC) communication cost; (G.) greedy-based approach; (Dens.) replica density; (RT) response time; (Spd.) node speed; (Self.) node selfishness; (Dist.) hop-distance to the resource; (Upd. Freq.) update frequency of the resource; (Sto.) storage capacity of the node; (Perf.) node performance; (BW) node bandwidth; (Pos.) position of the node in the network topology.

7 (✓) the replica deployment scheme is revised when changes in the context happens; (✗) the replica deployment scheme is not revised when changes in the context happens; (?) nothing is said about the reactivity of the solution.

8 (🏠) local to the node or its vicinity; (🌐) global to the network.

9 (✗) none cost measure is taken into consideration; (🔌) energy consumption; (⏲) access delay to the resource; (⚖) to use the minimum number of replicas; (📶) bandwidth usage; (✉) message usage; (🗄) storage usage.

10 (✓) the synchronization between the replicas of the resource is addressed; (✗) the synchronization between the replicas of the resource is not addressed or only are considered read-only resources.

* In [49] a copy primary replication scheme synchronized under SRP is considered. However, the synchronization is not contemplated in the simulation/evaluation.

** In [62][63] data updates of the original copies in known and fixes periods are contemplated, but no modifications on data replicas.

Table 3.2 *Overview of the leader election algorithms in dynamic environments and their features. Source: Own elaboration.*

	Reliable Communications [(1)]	*Simulated* [(2)]	*Simulator*	*Mobility Model* [(3)]	*Election Criteria*
[98]	✔	✖	-	-	The node that detects the partition
[163]	✔	✖	-	-	Undetermined utility function
[164]	✖*	✔	GloMoSim	RWP	Undetermined utility function
[127]	✔	✖	-	-	Lowest ID
[24]	?	✖	-	-	Probability, influenced by the capabilities of the node
[134]	✔	✔	?	RWP	Undetermined utility function
[140]	?	✔	ns-2	RWP	The node with highest energy level and lowest reputation level
[103]	?	✔	ns-2	St. RWP	Node with highest energy

[1] (✔) communication is explicitly assumed reliable; (✖) communication is explicitly assumed not reliable; (?) nothing is said about communication reliability.
[2] (✔) the proposal is simulated; (✖) the proposal is not simulated.
[3] (RWP) Random Waypoint mobility model; (St.) nodes are static.
[*] [164] considers a weak reliability, i.e., *"the message delivery is guaranteed only when the sender and the receiver remains connected for the entire duration of the message transfer"*, thus there have not been considered other features that can affect the message delivery, such as the bandwidth saturation or channel fading.

Table 3.3 *Overview of the cluster head election approaches revised and their features. Source: Own elaboration.*

	Reliable Communications [(1)]	*Simulated* [(2)]	*Simulator*	*Mobility Model* [(3)]	*Election Criteria*
[172]	✔	✔	?	RW RWP	Reliability of the node
[4]	?	✔	ns-2	RWP	The node with minimum expected relative mobility
[165]	?	✔	ns-2	RWP	Utility function
[18]	✔	✔	ns-2	RWP	The closest node to the gravity point of the geographical zone

[1] (✔) communication is explicitly assumed reliable; (✖) communication is explicitly assumed not reliable; (?) nothing is said about communication reliability.
[2] (✔) the proposal is simulated; (✖) the proposal is not simulated.
[3] (RWP) Random Waypoint mobility model; (RW) Random Walk mobility model.

election algorithms for dynamic environments (Section 3.3) and Table 3.3 shows the cluster head election approaches (Section 3.4).

In general, it can be seen that the majority of the total of the proposals (29) explicitly assume **reliable connections** (12 of 29) or say nothing about this assumption (15 of 29). Only two proposals, [86] and [164] consider no reliable connections in their proposals. Although [164] only assumes a weak reliability, i.e., *"the message delivery is guaranteed only when the sender and the receiver remain connected for the entire duration of the message transfer"*. Therefore there have not been considered other features that can affect the message delivery, such as the bandwidth saturation or channel fading. This can be considered as a strong assumption, since data or message loss is an inherent property of mobile and wireless ad-hoc networks, especially in highly dynamic and large-scale environments. This highlights the fact that the solutions proposed in this topic generally are focused on objectives such as load balancing, energy consumption, and scalability, while system reliability is a secondary consideration [174], although fundamental for the applicability of the proposal in real-world scenarios.

Regarding the **evaluation** of the proposals, the majority (21 of 29) use a network simulator to perform the evaluation. Of these 21 proposals, 7 of them use ns-2 or ns-3, 6 of them use other simulators, and 8 of them do not specify which simulator have used to evaluate their proposals. This makes it difficult to reproduce the validation performed and the possible verification of the results that the authors have obtained. In respect to the **mobility model** used in the evaluation, the majority of the proposals use the RWM model, whereas only [178] and [177] use additional mobility models as Manhattan Grid or RPGM. In fact, in their work, [178] and [177] underline how deep can affect the mobility model in the replication strategy, and thus the importance of evaluating the proposal under different mobility models.

Focusing on the approaches to improve data/service QoS through dynamic replication (Table 3.1), the **objective** of the majority of the proposals studied is to enhance the availability of the data/service, whereas a few also consider the response time in accessing the resource. 13 of a total of 17 are **resource-aware**. The cost measure taken as reference is varied, but energy consumption, bandwidth usage and response time predominate. Although energy consumption could be considered a good cost measure, the lack of standardization in the evaluation tools makes it dependent on the tool used and difficult the comparison of proposals. Instead, measures such as network traffic and the number of messages sent are more independent of particular tools or technologies, and therefore they are more appropriate measures to compare and evaluate different proposals.

Regarding the **scope of the knowledge** used to perform the host election, it can be seen (Table 3.1) how only 6 of 17 proposals use local information to the node or the node group, whereas the majority, the other 11, use global knowledge to the network to support their decisions. To obtain and maintain a global knowledge of the network is expensive in dynamic multi-hop ad-hoc networks [34]. In this respect, approaches that use local information are more efficient and scalable in resource-constraint environments. Moreover, **cross-layer interaction** approaches [39], such as [98], [149] and [91] are of particular interest. These proposals suggest to use the routing protocol to obtain information about the network topology, and they arise as more flexible designs for managing the dynamics of mobile environments and reducing the resource consumption.

Another important aspect to take into consideration is the **synchronization** of the replicas. Although the increment in replicas supposes a direct increment in the availability of the resource, it also leads to an increment in the messages need to synchronize a higher number of replicas. Additionally, in a collaborative system, the replica of a resource could easily become outdated or invalid if synchronization is not taken into consideration. From the 17 proposals, only [49] and [62][63] address the synchronization of the replicas. However, they only do it partially. In [49] a copy primary replication scheme synchronized under SRP is considered. However, the synchronization is not contemplated in the simulation/evaluation. In [62][63], data updates of the original copies in known and fixes periods are contemplated, but no modifications on data replicas.

Only 7 of the 17 works apply **clustering techniques**, getting scalability. Of these, only 3 apply dynamic clustering techniques, based on node positions [2][3] and node speeds [86]. [166] demonstrates that considering node speeds and direction as the criterion for node clustering results in more stable node clusters.

Finally, Tables 3.1, 3.2 and 3.3 show that there is a wide variety of host/leader/CH **election criteria**. It should be noted that the election criteria are dependent on the objective of the system. Thus, for example, [18], [4], [86] or [149] aim to elect stable nodes, in order to reduce the number of elections. Hence, their election criteria are based on the speed of the node. Other proposals determine as election criteria the access frequency, the battery life or the degree of selfishness of the node. However, the requirements of a system could vary according to different application scenarios, or even in the same scenario over time. Therefore it is important to provide flexible mechanisms that make easy to customize the election criteria, not only at design-time but also at run-time. To this respect, proposals such as [153], [163][164], [140], [134] and [165], perform the election on the basis of a utility function. Usually, this utility function evaluates different features of the candidate node, returning a score that represents how suitable is that node. The main advantage of this solution is that it is easily adaptable. This is, by introducing a simple change in the utility function, the criteria on which the host is elected is modified. However, generally, all the nodes have to use the same function for a correct operating, which could make difficult a distributed modification at run-time.

3.6 Summary

This chapter has provided a review of the existing proposals related to the dynamic service provisioning in dynamic environments. These have been divided into three categories: (1) approaches to enhance data/service QoS through dynamic replication, (2) leader election algorithms in dynamic environments, and (3) cluster head election algorithms.

From the proposals studied, two main drawbacks can be concluded: (1) these are frequently ad-hoc solutions have been developed for specific scenarios. Therefore, they are difficult to apply in other scenarios or with other requirements than those for which they were devised; and (2) they often assume reliable communication channels. This can be considered as a strong assumption, since data or message loss is an inherent property of mobile and ad-hoc wireless networks, especially in highly dynamic and large-scale environments.

As desirable features that a system should posses to efficiently address the dynamic service provisioning in dynamic environments the following can be highlighted:

- To be reactive to context changes, in order to modify the deployment scheme when, for example, a node leaves the network, or a network partition occurs.
- To base its decisions on local information, owing to in dynamic environments to obtain global information of the system is usually costly. To this end, cross-layer approaches can be useful to gather information about the network status and avoid to duplicate tasks in the system.
- To provide a resource-aware solution, supporting a balance between the quality attributes of the system and the resource consumption.
- To base the election in a Utility Function. This would facilitate to modify the election criteria according to the requirements of the application scenario.

Regarding the evaluation of the proposal, currently, the majority use network simulators. However, in this evaluation, a major diversity of mobility models should be taken into consideration. Additionally, as cost measures to asses the efficiency of the proposal, criteria such as network traffic and the number of messages sent should be used. These are more independent of particular tools or technologies than energy consumption, and therefore they are more appropriate measures to compare and evaluate different proposals.

Finally, the synchronization of the replicated resource (service or data) is currently relegated to a secondary plane. However, the increment in the number of resource replicas implies an increase on synchronization messages and, thus, reducing the efficiency of the approach. This may invalidate some of the proposals that perform an extensive replication of the resource to increment its availability.

Part II

Proteo Architecture: Design, Modelling, Simulation, and Evaluation

Chapter 4

Proteo: A Self-adaptive Software Architecture

Chapter Abstract

This chapter introduces the Proteo architecture. The primary objective of Proteo is to provide a self-adaptive and resource-efficient solution to support the availability of services in dynamic topology environments, through replication and self-configuration techniques.

The chapter starts presenting a motivating scenario and providing the system model under which the proposal will be developed. Then, the structural design of Proteo architecture is shown, together with the responsibilities of each of its main components. These responsibilities are closely related to the phases of MAPE-K autonomic loop of Autonomic Computing. Furthermore, how to evaluate in Proteo the feasibility of a node to host an active service replica is presented. This evaluation is based on a cross-layer approach to efficiently approximate the position of a node in the network topology.

In Proteo an election algorithm is used. To this end, two new election algorithms are proposed: Consensus and Voting algorithms in order to dynamically establish what node will be elected to host the active replica. These are devised to operate in mobile environments with highly dynamic network topologies and under unreliable communication channels. In addition to this, three existing proposals of election algorithm (Bully, Kordafshari, and Vasudevan algorithms) have been selected to be adapted and integrated into Proteo. Finally, the performance of these algorithm proposals is theoretically analysed, revealing the existing trade-off between message complexity and turnaround time.

Chapter Contents

4.1 Introduction

This chapter introduces the main components of the self-adaptive software architecture Proteo, which will be later detailed in Chapter 5. The main objective of Proteo is to provide a self-adaptive and resource-efficient solution to support the availability of services in dynamic environments. The proposal is based on **service replication techniques** together with **a self-configuration approach** for the activation/hibernation of the replicas of the service depending on relevant context information from the system.

This chapter is organized as follows: first, a motivating scenario is exposed in Section 4.2, which, simply, introduces the problematic intended to be addressed by Proteo. Then, Section 4.3 presents the system model in which the proposal is framed and the list of assumptions under which the proposal will be developed and studied. Section 4.4 will introduce the structural design of the self-adaptive software architecture. Additionally, the approach proposed and followed by Proteo architecture to evaluate the feasibility of a node to host an active service replica is shown in Section 4.5. Finally, Section 4.6 will present the proposed Consensus and Voting election algorithms. Moreover, for the sake of a performance comparison between the proposed algorithms and other existing ones, we introduce the adaptations that have been made to the Bully, Kordafshari, and Vasudevan algorithms, to make them applicable to the context of the proposal. Additionally, a theoretical performance study and analysis on message complexity and turnaround time of the algorithms are presented.

4.2 Motivating Scenario

In this section, an example scenario that reflects real needs for dynamic data and service replication is described (Fig. 4.1). In the scenario, three members of a team are working together in a remote area. Two kinds of devices can be found: (1) a laptop located in a car; and (2) three mobile devices (one per team member). Different functionalities provided by the corresponding services could be required. For example, lets suppose an Image Repository service. Each team member can take several pictures of an element with his/her mobile device; other team members can take additional photos for the element, or the same member can take other photos for related elements. Moreover, annotations to these pictures can be added, which can be related to other pictures. Therefore, the Image Repository service must keep an ordered and consistent set of this information, and at the same time, it must provide high **availability**, to allow team members to access and share pictures.

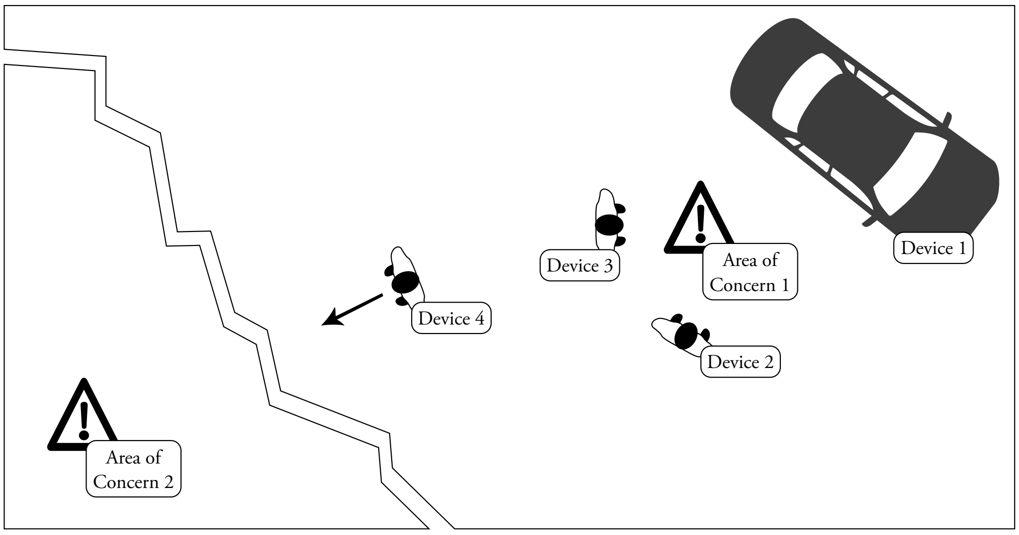


Fig. 4.1 *A hypothetical scenario of a collaborative work team in a remote area.*

In this scenario, the proposed self-adaptive architecture is intended to improve the availability of the Image Repository service as well as to provide a resource-aware solution. This is highlighted in the following situation:

1. Initially, all team members are near the *Area of Concern 1*. All the mobile devices are connected to the laptop located in the car. Therefore, the active service replica is the one deployed in the laptop (*Device 1* in the Fig. 4.1), as it presents better computational features.

2. The team member using the *Device 4* will move to the *Area of Concern 2*. This area is out of the laptop coverage. The self-adaptive architecture detects this situation, and the adaptation process starts to carry out the deployment and activation of a new replica of the Image Repository service. Since the only device available is itself, it will host an active replica of the service. At this point, the network is partitioned into two groups: (1) the *Devices 1*, *2* and *3*, where the *Device 1* hosts the active replica; and (2) the *Device 4*, in which the new service replica was deployed.

3. Later, the team members of the *Devices 2* and *3* move to the *Area of Concern 2*. At a given time the battery of the *Device 4* is under the 25%. The adaptation process will be started again to decide where to host a new service replica, with *Device 2* and *3* as candidates. At this moment, there are no any clients using the service replica deployed in

the laptop (*Device 1*) because of the network partition. Therefore, that service replica will be disabled in order to save resources.

To successfully address this scenario to provide a **context-aware** solution able to detect the events (e.g., network partitions, server disconnections or low level of battery charge) that could affect to the quality attributes (e.g., availability) of the service deployed is necessary ; a **self-adaptive** solution, able to self-configure the system to maintain or even improve its quality attributes in a transparent and unaware way for the users; a **resource-aware** solution, with low resource requirements (e.g., battery) and able to provide an efficient configuration of the system; and a **distributed** solution, where no device is indispensable for the correct operating of the own self-adaptive solution.

Additionally, **it should be noted that the architecture proposed, Proteo, is not limited to this example scenario**. The solution has been designed as an **adaptable** solution, with the intention of being applied in different application scenarios with different requirements or requirements that could vary over time.

4.3 System Model

The network in which Proteo is contextualized consists of a set of nodes, represented by V, which are assumed to have a unique identifier. Each pair of nodes can be connected by a link, depicting as E the complete set of links of the network. In this way, the network can be represented as a graph $G = (V,E)$, where the nodes represent the vertices, and the links the edges.

While the connection between two nodes is being established, the link could be **not bidirectional**. However, it is assumed that this link become bidirectional in a finite time after the connection is established.

Communication channel is **unreliable** and there is **channel fading** caused by the distance. Buffer size is **limited** and **network congestion** is possible.

Multi-hop communication is allowed, and the routing protocol manages it at the network layer. It is assumed a proactive routing protocol to be implemented. If not, a discovery service would be necessary.

The nodes of the set V can be or not mobile. If they are mobile, their movement cannot be controlled or influenced.

A node can fail or leave the network at any moment without notice.

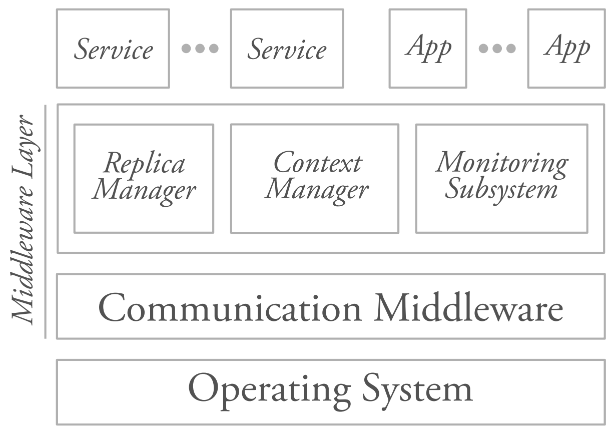


Fig. 4.2 *High-level Proteo layered architecture.*

4.4 Architectural Design Proposal

Proteo architecture is based on microservices, and it is composed of three main elements (Fig. 4.2): (1) *Monitoring Subsystem*, (2) *Context Manager Service*, and (3) *Replica Manager Service*. These elements will be replicated in each node of the network with enough capabilities to act as server, and their union and coordination will provide a resource-efficient solution to satisfy the availability of the services deployed in a dynamic network topology.

Proteo bases its communication on a *Communication Middleware* (Fig. 4.2), which allows communication between the different entities (nodes and services) of the system under two different communication paradigms: (1) a *Publish-Subscribe*; and (2) a *Request-Response*. The combination of both paradigms is usually known as SOA 2.0 [90] (a.k.a. advanced SOA), in which services are not just passive entities, but are also able to receive and generate events proactively. Additionally, in order to perform the coordination of the distributed nodes that compose the network, a *message passing* paradigm will be used in Proteo.

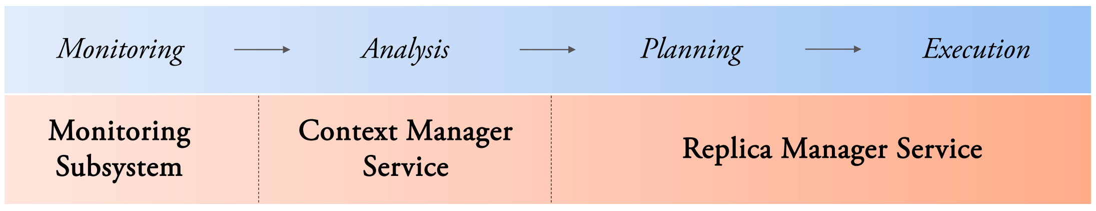


Fig. 4.3 *Responsibility mapping between Proteo elements and MAPE-K autonomic loop phases.*

The responsibility of Proteo services are closely related with the phases of MAPE-K autonomic loop of Autonomic Computing (Fig. 4.3): the *Monitoring Subsystem* sense the context information in relation to the node in which are deployed; the *Context Manager Service*, in addition to process this information, will also be responsible for analysing it to decide when a new system adaptation is necessary; and the *Replica Manager Service* will be responsible for coming to an agreement with the rest of the *Replica Manager Services* deployed in the system to establish what replica will be activated. This coordination will be performed on the basis of a distributed **host election algorithm** (Section 4.6), which will carry out the election according to of a dynamic score obtained by means of a **utility function** (Section 4.5).

4.4.1 Monitoring Subsystem

The *Monitoring Subsystem* encompasses a set of supervising services, which sense the context information concerning the node in which are deployed, to detect potential events that could affect the availability of a service deployed in the node. The features to monitor will depend on a particular scenario. Nevertheless, battery and network topology are transversal features that will be present in the majority of application scenarios for which Proteo is devised. Therefore, without prejudice to add new monitoring services, in this work both the node battery and the network topology are monitored. Note that, in this work, a service has been designed for each context feature to monitor. This has been done to comply with established good practices in service-orientation design [50][106]. In this way, service reusability and autonomy are increased.

To monitor the context is a costly operation in terms of energy and bandwidth. Nevertheless, the response time of a self-adaptive system depends on the precision of the monitoring action. The information about the node capabilities (e.g., battery) is local knowledge that usually can be obtained easily, while a dynamic network topology requires continuous monitoring because of constant changes that it suffers. Further, the cost increases exponentially with each hop in the network [34], since the dynamicity of the network and effects of topology changes are improved.

However, the information managed and gathered by the routing protocol can be reused with the objective of monitoring and detect changes in the network topology. Although according to the OSI model (Open Systems Interconnection) the routing protocols are designated as network layer responsibility, it has been proved that a more flexible design is required for managing the dynamics of mobile environments, where the network status is obtained from a **cross-layer interaction** [39].

Therefore, in this work we propose to use the information gathered by the routing protocol with two objectives: (1) to monitor and detect changes in the network topology; and (2) to use this information as a substitute of a discovery service. In this way, it is possible to provide a more resource efficient solution, avoiding to duplicate network monitoring tasks in the system.

4.4.2 Context Manager Service

The *Context Manager Service* will receive information from the monitoring services of its node. The information received by this service is basic information, but unrelated a priori. Its responsibility consists of processing, contextualizing and storing it. This information will be used by the *Replica Manager Service* in order to adjust the configuration of the activation/hibernation of the replicas according to the changes produced in the context.

Additionally, the *Context Manager Service* is responsible for detecting changes in the context that could affect the quality attributes of the service, and thus decide when the adaptation process has to be initiated. An example that illustrates this can be the following one: the network monitoring service detects that a node is no longer reachable. Then, it triggers an event to spread this information. The monitoring service unknowns the role or importance of the disconnected node. However, the *Context Manager Service*, when receives this information, detects that the disconnected node is indeed the current server and hence to elect a new server is necessary. The *Replica Manager Service* will be the responsible for performing the adaptation (Section 4.4.3).

Moreover, the *Context Manager Service* will be responsible for evaluating the feasibility of the node in which it is deployed to host an active service replica. This evaluation will be performed through a utility function, which will evaluate the node features according to a set of defined weights. In this way, the node only shares the score obtained through the utility function (which will be detailed in Section 4.5), whereas node specific features, such as node battery level, will remain as local information to the node.

4.4.3 Replica Manager Service

The *Replica Manager Service* runs the election algorithm (see Section 4.6) to achieve an efficient configuration for the activation/hibernation of the replicas of the service of which Proteo is supporting its quality attributes. When a change in the context is detected that could affect the quality attributes of the service; it will be responsible for agreeing with the rest of the *Replica Manager Services* deployed in the system. The objective of this coordination is to know if there is a better activation/hibernation scheme of the replicas and if so, to establish what replica will be activated. It should be emphasized that any node can initiate the election process. This is because each node has local information about its own features that is not accessible to the rest (e.g., battery level).

4.5 Host Feasibility Evaluation

The feasibility of the node to host an active replica of the service is evaluated through a utility function. This function will return a score in the 0-1 interval, where 1 represents the best feasibility. Usually, the function will be represented by a weighted average.

The node features and resources evaluated in the function will depend on the particular application scenario and the application requirements. For example, in some cases, within the same application scenario, it could be desirable to prioritize the battery of the devices. In such case, the weight assigned to battery level in the function would be higher.

As exposed in Section 4.4.1, in this work, without prejudice to add other features, both the node battery and the network topology are taken into consideration to evaluate the nodes.

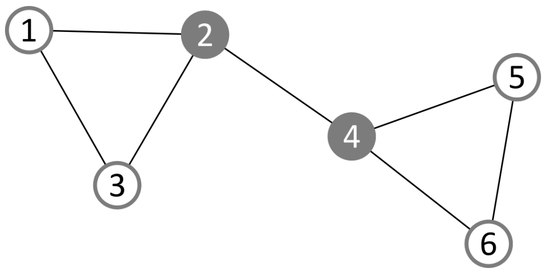
The position of the node in the network topology is a fundamental feature to take into consideration in the service deployment process, due to the availability, reliability, and performance of distributed applications are critically conditioned by the placement of the services in the distributed system [95].

As introduced in Section 4.4.1, to efficiently monitoring and evaluate the position of a node in the network, in this work, a cross-layer approach to use the information gathered by the routing protocol is proposed. Routing protocols for dynamic network topologies usually try to make an easy topology representation by considering only the best path between two nodes [1].

The routing protocol builds and maintains in each node a routing table with information about the reachable nodes and for each of them the gateway and the number of hops. Through the direct connections of a node, i.e., one hop connections, the system can approximate its position within the network topology. Although it does not supply the exact position of a node, it can provide enough information to approximate the network topology successfully.

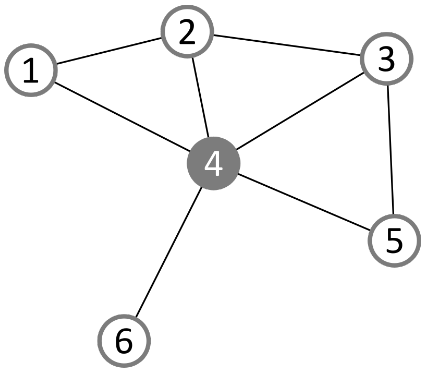
It should be noted that this approach requires proactive routing protocols [1], which periodically update routing tables according to the network topology changes.

Fig. 4.4, Fig. 4.5 and Fig. 4.6 show three examples of ad-hoc networks and how, employing the information provided by the routing protocol in the form of routing table, then to identify the most centric nodes in the network topology is possible. It is worth highlighting the example of Fig. 4.4, in which, according to heuristic proposed, both nodes 2 and 4 are in a good position within the network topology, which is true. In this case, the tie will be broken by other capabilities taken into consideration.



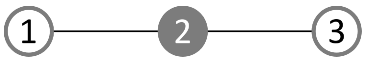
Node	1	2	3	4	5	6
Number of direct connections	2	3	2	3	2	2

Fig. 4.4 *Example 1 of the heuristic used to approximate the position of a node within the network using the information provided by the routing table.*



Node	1	2	3	4	5	6
Number of direct connections	2	3	4	5	2	1

Fig. 4.5 *Example 2 of the heuristic used to approximate the position of a node within the network using the information provided by the routing table.*



Node	1	2	3
Number of direct connections	1	2	1

Fig. 4.6 *Example 3 of the heuristic used to approximate the position of a node within the network using the information provided by the routing table.*

The general utility function to evaluate the feasibility of a node to host an active service replica can be implemented by the equation 4.1.

$$score = \frac{W_0 \cdot f_0 + W_1 \cdot f_1 + \cdots + W_n \cdot f_n}{n}, \text{ for } W_i, f_i \in [0,1], \sum_{i=0}^{n} W_i = 1 \tag{4.1}$$

where:

f_i = normalized value of feature i
W_i = weight assigned to feature i
n = number of features considered

In the case of taking into consideration the battery and direct connections of a node (i.e., position in the network topology), the utility function can be implemented by the equation 4.2.

$$score = \frac{W_0 \cdot bl + W_1 \cdot dcm}{2}, \text{ for } W_i \in [0,1], \sum_{i=0}^{n} W_i = 1 \tag{4.2}$$

where:

bl = battery level in a $[0-1]$ range
dcm = normalized direct connections of the node, which is given by $\frac{direct\ connections}{nodes\ in\ the\ routing\ table}$
W_i = weight assigned to feature i

4.6 Election Algorithms

In Proteo the election algorithm is executed in a distributed manner by the set of *Replica Manager Services* (Section 4.4.3) to establish what node will be elected to host the active replica. This election is based on the scores of the nodes (Section 4.5), and it will be the best-ranked node available which should be elected.

This section attempts to introduce an overview of the basic operating of the two new election algorithms proposed (Section 4.6.1) as part of the Proteo architecture proposal. Moreover, the adaptations proposed on existing election algorithms, to integrate them in Proteo architecture, are presented in Section 4.6.2. Finally, a theoretical performance study and analysis on message complexity and turnaround time of the algorithms are presented in Section 4.6.3. It should be noted that a detailed description of the election algorithms introduced in this section will be provided in Chapter 5.

4.6.1 Proposed Election Algorithms

In this section, two new election algorithms are proposed: **Consensus** and **Voting**. These election algorithms are devised to operate in mobile environments with highly dynamic network topologies and under unreliable communication channels.

It should be highlighted that although these election algorithms were initially designed to operate as host election algorithms, they can also be used to elect a coordinator node in a distributed manner for other purposes.

Consensus Election Algorithm

The Consensus election algorithm is based on three basic types of messages:

- *Score message*: Through this message, a node communicates its score, obtained through the utility function.
- *Server Request message*: This message is used when the requesting node wants to establish a client-server connection with the receiver node, where the requesting node will act as the client and the receiver node as the server.
- *Server Acceptance message*: Used to respond to a *Server Request* message. Through this message, the receiver node of a *Server Request* accepts the request to act as the server for the requesting node. Therefore, when the sender node of a *Server Request* message receives the *Server Acceptance*, it will establish the client-server connection, and it will start to use the service provided.

On the basis of these messages the basic operation algorithm is as follows:

(A) **Given a node *N* that starts the election process**:

(1) *N* calculates its own score, using a defined utility function.

(2) *N* broadcasts its score by means of a *Score* message.

(3) *N* waits for the *Score* messages of its neighbours.

(4) After *N* receives all the *Score* messages, it calculates the best ranked node:

- **If *N* is the best ranked node**: *N* becomes the elected node and waits for *Server Request* messages. Whenever *N* receives a *Server Request* message it replies with a *Server Acceptance*.
- **If the best ranked node is another node *P***:

(1) *N* sends to *P* a *Server Request* message and waits for the *Server Acceptance* reply.

(2) When *N* receives the *Server Acceptance* message from *P*, *N* sets *P* as server.

(B) **Given a node *Q* that does not start the election process**: When *Q* receives the first *Score* message, it will become aware of that an election process has started. Thus, it will proceed in the same way that the node that has started the election (step A.1).

As can be seen, there is no difference in the procedure between a node that starts the election and a node that participates in that election. In this way, instead of avoiding concurrent elections, we use them to spread the *Score* messages.

Score messages are also used to know what nodes are available to be elected. That is, if a node does not send its score, it cannot be elected. A node could not send its score by two main reasons: (1) the node has failed and thus is not available; or (2) the node is not suitable for election. In this way, the Consensus algorithm can be used in heterogeneous networks, where some nodes could not have enough computational resources to serve to other nodes (e.g., sensor nodes).

In Fig. 4.7 an example of the operating of the Consensus election algorithm is shown. As it has been exposed, it is irrelevant what node starts the election, since all of them broadcast its score. After that, the nodes request service to the best-ranked node, in this case, as it is assumed that $ID = score$, is the Node 4. This node will reply with an *Acceptance* message.

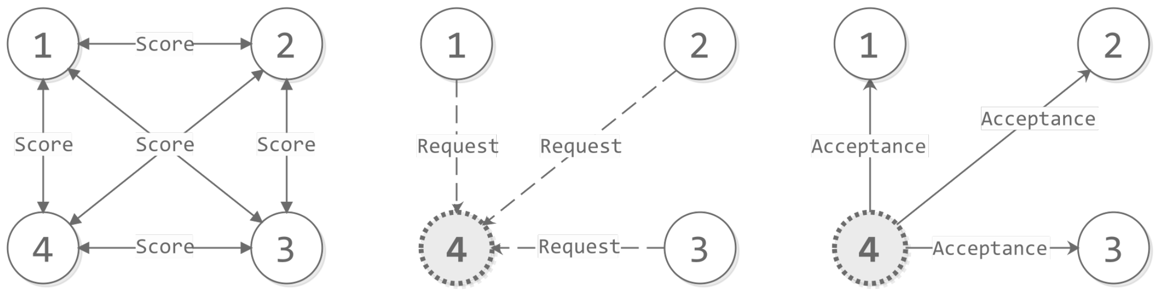


Fig. 4.7 *Consensus election example. Known IDs and highest ID as election criteria.*

Additionally to the three basic messages presented, the Consensus algorithm uses two additional messages devised to avoid concurrent servers elections in a group of nodes:

- *Server Rejection message*: If the request is made to the wrong node, i.e., a node that is not serving, this message will be used to respond. Through this message, the receiver node of a *Server Request* rejects the request to act as the server for the sender node. This message will include information about who the current server is.

- *Server Bare Rejection message*: This message has the same function that a *Server Rejection* message. However, it has no additional information about the server, because the requested node has not a known server established.

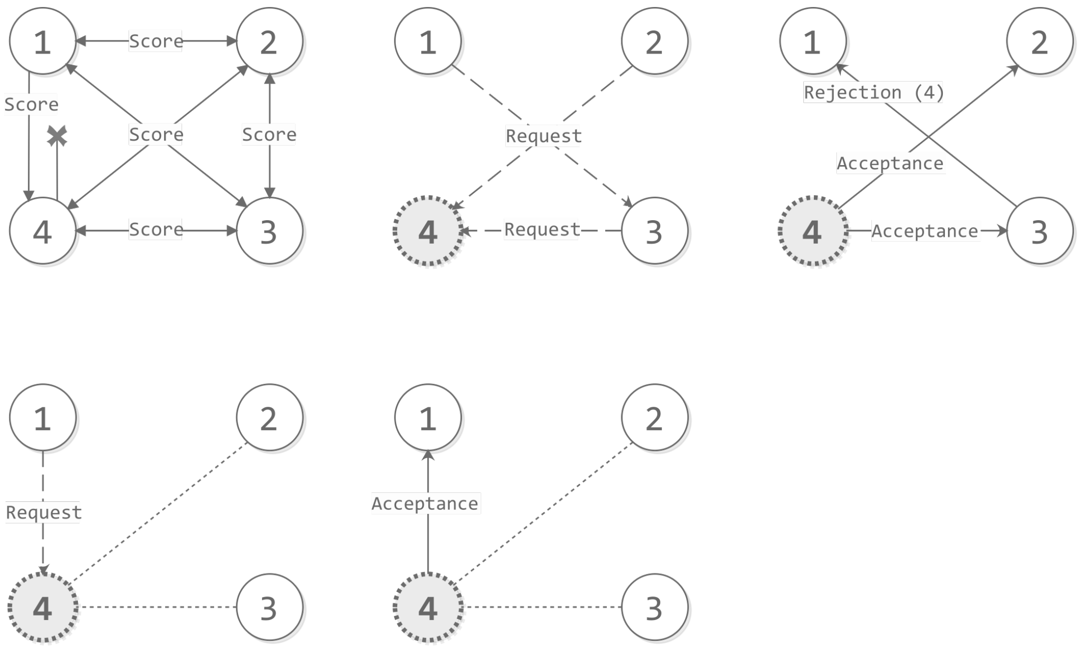


Fig. 4.8 *Consensus election example with message loss. Highest ID as election criteria.*

With these messages, specifically the *Rejection* message, the algorithm can recover from message loss. For example, in the scenario shown in Fig. 4.8 the *Score* message from the Node 4 to the Node 1 is loss. Thus, Node 1 will request service to Node 3, highest ID node known alive from its point of view. However, Node 3 knows the availability of Node 4; therefore it will answer to Node 1 with a *Rejection* message, indicating Node 4 as the elected node. Consequently, after Node 1 has received this information, it will request service to Node 4, which will reply affirmatively, recovering from the message loss.

Voting Election Algorithm

The Voting election algorithm is devised to provide a more robust algorithm against *Score* messages loss. This algorithm is based in three types of messages:

- *Score message*: This message is identical, in function and structure, to the *Score Message* used in the Consensus election algorithm (see Section 4.6.1).

- *Vote message*: It is used by a node to emit a vote to the best-ranked node participating in the election.
- *Elected Coordinator message*: It is used by a node to announce its victory in the elections when it receives at least the half plus one of the votes.

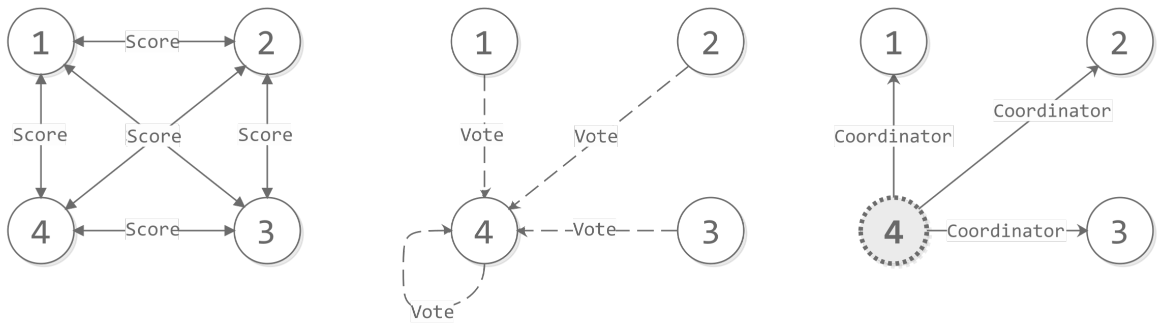


Fig. 4.9 *Voting election example. Known IDs and highest ID as election criteria.*

The election process operates as follow (Fig. 4.9):

(A) **Given a node *N* that starts the election**:

(1) *N* calculates its own score, on the basis of a defined utility function.

(2) *N* broadcasts its score by means of a *Score* message.

(3) *N* Waits for the *Score* messages of its neighbours.

(4) After *N* receives all the *Score* messages, it calculates the best ranked node:

∗ **If *N* is the best ranked node**:

· *N* increments the number of votes received in one (i.e., the node casts a vote to itself)

· *N* waits for more vote messages.

∗ **If the best ranked node is another node *P***:

· *N* sends to *P* a *Vote* message.

· *N* waits for the *Coordinator* message, which will be sent by the elected leader.

(B) **Given a node *Q* that does not start the election process**: When *Q* receives the first *Score* message, it will become aware of that an election process has started. Thus, it will proceed in the same way that the node that has started the election (step A.1).

(C) **Given a node T, regardless of whether it started the election or not**:

- **If T receives the $\frac{n}{2}+1$ *Vote* messages**: T declares itself as the elected leader and announces this by means of broadcasting a *Elected Coordinator* message.
- **If T receives a *Elected Coordinator* message from P**: T sets P as server. If T did not send its *Vote* message before to receive the *Elected Coordinator* message, it cancels the sending of the message, since this *Vote* message is already unnecessary.

As in the Consensus algorithm (Section 4.6.1), Score messages are used both to know what nodes are available and to perform the leader election on the basis of a dynamic utility function. The advantage of this algorithm concerning the previous is that *Score* messages are not essential. In the Voting algorithm, it is only necessary for $\frac{n}{2}+1$ nodes in the group to have the correct scores to perform the correct election, and therefore the *Score* message loss is not as critical as in the Consensus algorithm.

4.6.2 Proposed Adaptations on Existing Election Algorithms

In order to compare the performance of the new two proposed election algorithms with existing approaches, the election algorithms of Bully (Section 2.4.1), Kordafshari (Section 2.4.2), and Vasudevan (Section 3.3.2) have been selected.

The Bully algorithm has been selected because it is one of the more popular algorithms for dynamically electing a leader from a group of distributed computer processes since it was proposed in 1982. Since then, different improvements have been proposed on the Bully algorithm, of which that proposed by Kordafshari et al. can be highlighted [89]. The advantages of the Kordafshari proposal over other variants of the Bully algorithm is that Kordafshari el al. proposed an effective method to reduce the number of coordination messages and concurrent elections without make additional assumptions. Finally, the Vasudevan proposal highlights because it is one of the few proposals that does not assume reliable communication channels (see Chapter 3). However, it does assume a weak reliability, this is, *"the message delivery is guaranteed only when the sender and the receiver remain connected for the entire duration of the message transfer"*[164].

To make applicable these proposals in the context of this work, it is necessary to perform some adaptations on them.

In the Bully algorithm the coordinator election is performed on the basis of the node ID, i.e., the available node with highest ID will be elected as coordinator. These IDs are assumed to be static and known. Therefore, the Bully algorithm, per se, is not applicable to scenarios where the election is performed by some calculated score that changes over time, as it is common in dynamic systems.

Hence, to make possible to apply the Bully algorithm in these cases, the following modifications are introduced:

- *Election* and *Answer* messages will now include the score of the sender node.
- A node *N* will use the last known score of its neighbours to send the *Election* messages, which will be sent to those with higher scores.
- If a node is disconnected from the group, its last known score will be set to 0, to avoid to take as reference excessive outdated scores.
- Now, a node *N* could receive an *Election* message from a node *P* with higher score than *N*, owing to *P* ignores the actual score of *N*. In such case, *N* waits for the *Coordinator* message, because it now knows that exists a node *P* with higher score available.

These modifications are introduced to allow the applicability of the Bully algorithm in dynamic scenarios, reducing the excessive use of *Election* and *Answer* messages that would be necessary if the last known score would not be used as a filter.

Since the Kordafshari algorithm is based on the Bully algorithm, these modifications are applied to the Kordafshari proposal as well.

Regarding the Vasudevan algorithm, it can operate with an election based on dynamic scores without any modifications. However, because of its integration on Proteo, alive messages (*Heartbeats*, *Reply* and *Response* messages) are no longer necessary. This is because, in Proteo, the reachability of the neighbours or discovery is managed by the network monitoring service. This service is based on the information provided by the routing protocol on the routing table, following a cross-layer approach and thus reducing the necessity of additional messages from OSI application layer.

4.6.3 Theoretical Analysis of Message Complexity

In this section, a theoretical analysis of the selected (Bully, Kordafshari, and Vasudevan) and the proposed (Consensus and Voting) election algorithms is performed. This theoretical analysis will help to reach a better understanding of the behaviour of the algorithms in a simplified environment, owing to that neither the mobility of the nodes nor message loss will be taken into consideration. In should be noted that for the cases of the Bully and Kordafshari algorithms, both the original proposals (presented in Section 2.4.1 and Section 2.4.2, respectively) will be analysed, as well as the adapted versions to operate with dynamic scores presented in Section 4.6.2, in order to shown how the complexity of the algorithms changes because of the adaptations.

The algorithms will be analysed according to the indicated by Coulouris et al. [42], who establish that the performance of an election algorithm is measured by its total use of network bandwidth (the total number of messages sent during the election, i.e., message complexity), and the turnaround time (given by the serialized messages transmission time). For each algorithm its best and worst case scenarios will be analysed. Finally, the results obtained will be compared and discussed.

Standard Bully Algorithm

For the standard Bully election algorithm, this is, the presented in Section 2.4.1, the message complexity of the algorithm to elect a new coordinator is defined as:

Best Case

The node that starts the election is the node with highest ID. In this case, only $n-1$ *Coordinator* messages are sent, where n is the number of nodes involved in the election. Thus, the Bully algorithm complexity in the best case is $n-1 \in \Omega(1)$ messages, and the turnaround time is of one message (t_{msg}), where t_{msg} is the time required to send and process a message.

Worst Case

The node that starts the election is the node with lowest ID, and every node tries become the leader starting an election.

This scenario is depicted in the Fig. 4.10. Note that after Node 1 in Fig. 4.10a starts the election, the elections started by the Nodes 2 (Fig. 4.10c), 3 (Fig. 4.10e) and 4 (Fig. 4.10g) can happen concurrently but always with a slight anticipation of the Node 2 over the Nodes 3 and 4, and of the Node 3 over the Node 4. Otherwise, if any of the Nodes with higher IDs anticipates, nodes with lower ID will not try to dispute the election.

In this case, the first node will send $n-1$ *Election* messages, the second one $n-2$ and so on. Thus, in the worst case:

- $\sum_{i=1}^{n}(n-1)$ *Election* messages will be sent.
- Each of these *Election* messages will be replied with an *Answer* messages, therefore $\sum_{i=1}^{n}(n-1)$ *Answer* messages will be sent as well.

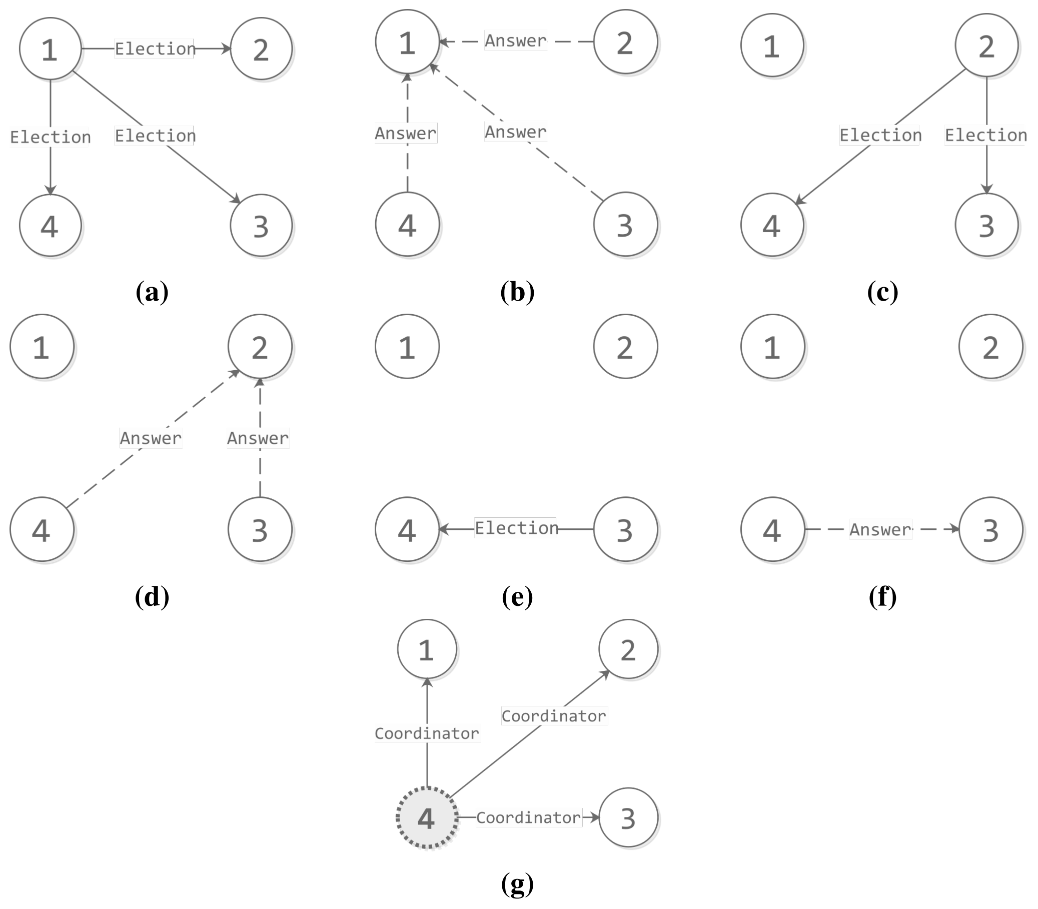


Fig. 4.10 *Bully election worst case scenario example. Known IDs and highest ID as election criteria.*

- Finally, when the turn arrives to the highest ID node, it will send $n-1$ *Coordinator* messages.

Consequently, the message complexity in the worst case scenario is defined by $2 \cdot [\sum_{i=1}^{n}(n-1)] + (n-1)$, which simplified becomes $2n^2 - 1 \in \mathcal{O}(n^2)$ messages.

If we pay no attention to the slight time difference between the concurrent elections, the turnaround is of three messages ($3t_{msg}$), due to *Election*, *Acknowledge* and *Coordinator* messages have to be sequential.

Adapted Bully Algorithm

However, as introduced in Section 4.6.2, the standard version of the Bully algorithm is not applicable to scenarios where the election is performed by some dynamic score, instead of known IDs. Therefore, to introduce some adaptations in the algorithm has been necessary. Consequently, the message complexity is increased, both for the best and the worst case, when introducing the adaptations presented in Section 4.6.2.

Best Case

Although the election is initiated by the best ranked node, this will send $n-1$ *Election* messages, due to it ignores the score of its neighbours. Since this is the best node, there will not be *Answer* messages, and the node will proclaim itself as new coordinator, sending $n-1$ *Coordination* messages. Consequently, the message complexity in the best case will now be given by $2 \cdot (n-1) \in \Omega(1)$.

In this case, the node will wait for the *Acknowledge* messages a time t_{wait} before proclaims itself as coordinator, where $t_{wait} > t_{msg}$. Thus, the turnaround time will be of $2t_{msg} + t_{wait}$, because there are two serialized messages: *Election* and *Coordination*.

Worst Case

In the worst case, none of the nodes know the score of their neighbours and all of them start the election concurrently. In this case:

- $n \cdot (n-1)$ *Election* messages will be sent.
- Since *Answer* messages are sent as a reply when the source of the *Election* message has a lower score, $\sum_{i=1}^{n}(n-i)$ *Answer* messages will be sent.
- Finally, the winner of the election process will sent $n-1$ *Coordinator* messages.

Therefore, in this case the message complexity will be delimited by $n \cdot (n-1) + [\sum_{i=1}^{n}(n-i)] + (n-1)$ messages, simplified $\frac{3n^2-n-2}{2} \in \mathcal{O}(n^2)$.

The turnaround time will be the same that in the best case: $3t_{msg} + (t_{wait} - t_{msg}) = 2t_{msg} + t_{wait}$. This is because, although there will be 3 messages serialized, the node with highest score will be waiting in parallel (t_{wait}) whereas *Acknowledge* messages are sent (t_{msg}).

Standard Kordafshari Algorithm

For the standard Kordafshari election algorithm, presented in Section 2.4.2, the message complexity of the algorithm to elect a new coordinator is defined as:

Best Case

As in the Bully algorithm, the best case is when the node that starts the election is the node with highest ID. In this case, only $n-1$ *Coordinator* messages are sent. Thus, the Kordafshari algorithm complexity in the best case is $n-1 \in \Omega(1)$, and the turnaround time is the time taken to send and process one message (t_{msg}).

Worst Case

Each node, except that with the higher ID, starts a concurrent election, as in the case depicted in Fig. 2.11 in Section 2.4.2. In this case:

- $\sum_{i=1}^{n}(n-i)$ *Election* messages will be sent.
- Thanks to the approach to reduce concurrent elections introduced by Kordafshari, only the lowest ID node will have replies to its *Election* messages, hence only $n-1$ *Answer* messages will be sent.
- The lowest ID node will send a *Grant* message to the highest ID node available.
- Finally, the highest ID node will send $n-1$ *Coordinator* messages.

Consequently, the message complexity in the worst case scenario is defined by $2 \cdot (n-1) + \sum_{i=1}^{n}(n-i) + 1$ messages, simplified $\frac{n^2+5n-2}{2} \in \mathcal{O}(n^2)$.

The turnaround time will be of $4t_{msg}$, because now the *Grant* message is added as a new serialized message. It should be noted, that although the Kordafshari algorithm reduces the number of messages in the worst case with regard to the Bully algorithm, it worsens the response time.

Adapted Kordafshari Algorithm

As in the case of the Bully algorithm, the Kordafshari algorithm is devised to operate with known IDs. Introducing the modifications presented in Section 4.6.2, to make possible to base the election of Kordafshari on unknown dynamic scores, the message complexity is increased, both for the best and the worst case.

Best Case

The same as in the Bully algorithm (see Section 4.6.3), which presents a message complexity of $2 \cdot (n-1) \in \Omega(1)$, and a turnaround time of $2t_{msg} + t_{wait}$

Worst Case

In the worst case, none of the nodes know the score of their neighbours and all of them start the election concurrently. In this case:

- $n \cdot (n-1)$ *Election* messages will be sent.
- $n-1$ *Answer* messages are replied only to the lowest ID node.
- The lowest ID node will sent a *Grant* message to the elected node.
- The elected node will sent $n-1$ *Coordinator* messages.

Therefore, the message complexity will be delimited by $n \cdot (n-1) + 2 \cdot (n-1) + 1$ messages, simplified $n^2 + n - 1 \in \mathcal{O}(n^2)$. The turnaround time will be of $4t_{msg}$.

Vasudevan Election Algorithm

The Vasudevan algorithm, presented in Section 3.3.2, unlike the Bully and Kordafshari algorithms, can operate with an election based on a dynamic score without any modifications, as introduced in Section 4.6.2. It should be noted that, because of its integration on Proteo, Vasudevan alive messages (*Heartbeats*, *Reply* and *Response* messages) are not longer necessary, because it will be the *Network Monitoring Service*, the responsible for monitoring the network and thus the alive and reachable nodes. In Proteo, the reachability of the neighbours or discovery is managed by the network monitoring service (Section 4.4.1).

Unlike in the case of the Bully and Kordafshari algorithm, the best and worst case scenarios are not dependent on who initiates the election but on the topology of the network, due to election is not performed on the basis of the ID of the nodes but on some unspecified score.

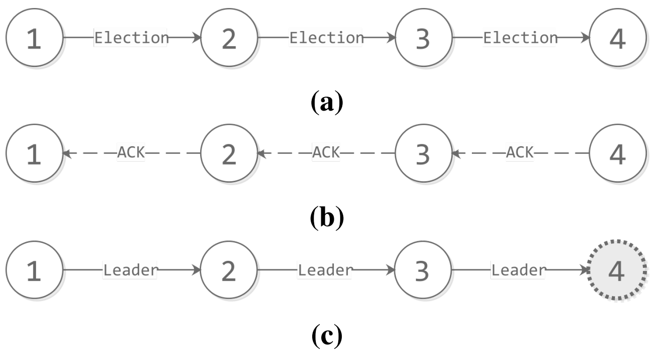


Fig. 4.11 *Vasudevan election example. This example depicts the best case scenario. Known IDs and highest ID as election criteria.*

Best Case

As messages between sibling nodes are unnecessary for the election but inevitable in the grown of the tree, the best case for this algorithm to reduce the message complexity is that in which the network topology is narrow and deep, as the example depicted in Fig. 4.11. In such example, the election is initiated by the Node 1, who grows the tree towards Node 4 (Fig. 4.11a). Once that the tree shrinks by means of *ACK* messages (Fig. 4.11b), Node 4 is elected leader by Node 1 (Fig. 4.11c).

In this case, $n-1$ *Election*, *ACK* and *Leader* messages are sent. Thus, for Vasudevan election algorithm the message complexity for the best case is given by $3\cdot(n-1) \in \Omega(1)$.

The Vasudevan algorithm presents a higher serialization of messages than the other algorithms. In the best case, where all nodes are in a line, all the messages sent will be serialized. Therefore the turnaround will be of $3\cdot(n-1)\cdot t_{msg}$, and it will be dependent on the number of nodes involved in the election.

Worst Case

The worst case scenario happens when the network topology is short and wide, i.e., a mesh topology, as depicted in Fig. 4.12. In this example, the election is initiated by the Node 1 (Fig. 4.12a). All the nodes of the network are siblings, and how it can be seen, the message exchanges in the steps 2 (Fig. 4.12b) and 3 (Fig. 4.12c) are unnecessary for the leader election, since the Nodes 2, 3 and 4 do not exchange any useful information.

Therefore, for a mesh topology, the following messages are sent:

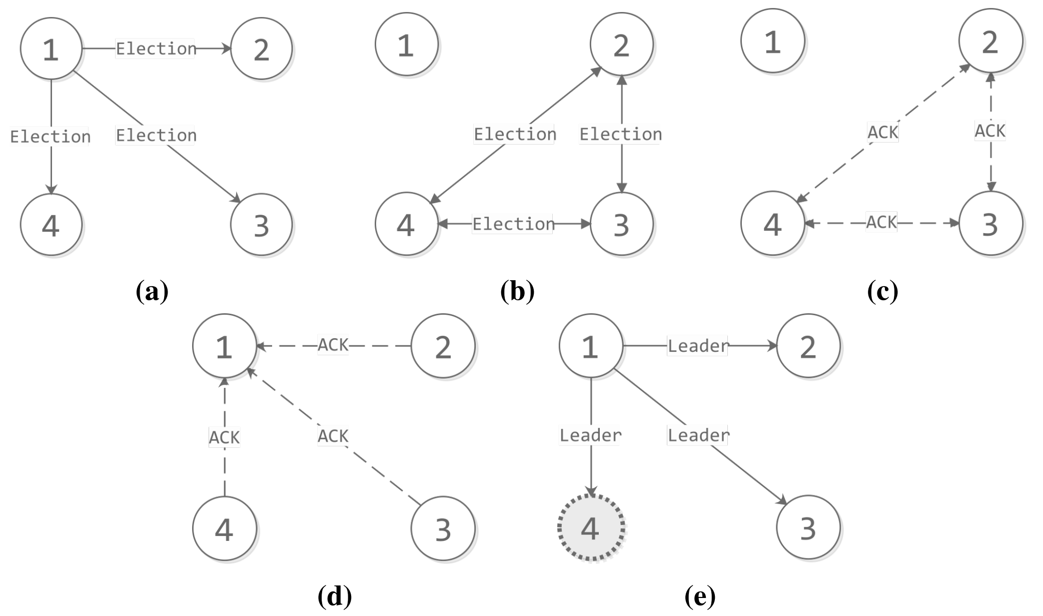


Fig. 4.12 *Vasudevan algorithm election example. This example depicts the worst case scenario. Known IDs and highest ID as election criteria.*

- $(n-1)+(n-1)\cdot(n-2)=(n-1)^2$ *Election* messages: root node sends to all nodes except itself; children nodes send to all nodes, except themselves and their parent.
- $(n-1)^2$ *ACK* messages: children nodes send to all nodes, including the parent node, except themselves.
- $(n-1)$ *Leader* messages.

Consequently, for the Vasudevan election algorithm the message complexity for the worst case is given by $2n^2-3n+1 \in \mathcal{O}(n^2)$.

In this case, the turnaround time is reduced to $5t_{msg}$, and now it is independent of the number of nodes. However, the Vasudevan algorithm has still a higher response time than the Bully and Kordafshari algorithms.

Consensus Election Algorithm

The proposed Consensus Election Algorithm, introduced in Section 4.6.1, is not influenced by the node that starts the election, neither by the topology of the network. However, the use of score messages has a great impact in the message complexity of the algorithm.

Best and Worst Cases

The complexity of the algorithm, both in the **worst** and **best** case scenarios, is given by:

- $n \cdot (n-1)$ *Score* messages, since each node broadcast its score to all nodes except itself.
- $(n-1)$ *Request* messages, each node, except the best ranked, sends a *Request* message to the best ranked.
- $(n-1)$ *Acceptation* messages, as reply from the best ranked node to the *Request* messages received.

Therefore, the complexity of the Consensus algorithm is given by $n \cdot (n-1) + 2 \cdot (n-1)$, which simplified is $n^2 + n - 2 \in \mathcal{O}(n^2)$. The turnaround time, is of $3t_{msg}$.

Voting Election Algorithm

The Voting election algorithm, introduced in Section 4.6.1, is devised to provide a more robust algorithm against *Score* messages loss than the Consensus algorithm.

Best Case

This case happens when only the minimum required *Vote* messages ($\frac{n}{2}+1$) are send, in addition to $n \cdot (n-1)$ *Score* messages and $(n-1)$ *Coordinator* messages. Therefore, the complexity in the best case of the Voting algorithm is given by $\frac{n}{2} + 1 + n \cdot (n-1) + (n-1) = n^2 + \frac{n}{2} \in \Omega(n^2)$. It worth noting, that when the best ranked node casts a vote to itself, this does not imply an actual *Vote* message, as it increments its votes counter internally, without the need of an actual message. Therefore, in some cases the election can be made with $\frac{n}{2}$ *Vote* messages. The turnaround time is of $3t_{msg}$.

Worst Case

This case happens when all $(n-1)$ *Vote* messages are sent, in addition to $n \cdot (n-1)$ *Score* messages and $(n-1)$ *Coordinator* messages. Thus, in the worst case, the message complexity is given by $n^2 + n - 2 \in \mathcal{O}(n^2)$. The turnaround time will be the same that in the best case, $3t_{msg}$.

Table 4.1 *Overview of the analysis performed on the election algorithms. In it, the theoretical message complexity and turnaround time in the worst case are shown.*

	Theoretical $\mathcal{O}$ message complexity	Turnaround time
Adapted Bully	$\frac{3n^2-n-2}{2} \in \mathcal{O}(n^2)$	$2t_{msg} + t_{wait}$
Adapted Kordafshari	$n^2+n-1 \in \mathcal{O}(n^2)$	$4t_{msg}$
Vasudevan	$2n^2-3n+1 \in \mathcal{O}(n^2)$	$5t_{msg}$
Consensus	$n^2+n-2 \in \mathcal{O}(n^2)$	$3t_{msg}$
Voting	$n^2+n-2 \in \mathcal{O}(n^2)$	$3t_{msg}$

^{n} represents the number of nodes involved in the election.
$^{t_{msg}}$ represents the necessary time to transmit and process a message.
$^{t_{wait}}$ represents the time that in the Bully algorithm a node waits to receive *Acknowledge* messages before to declare itself the winner of the election. Note that $t_{wait} > t_{msg}$.

Analysis and Discussion

According to Ahmed et al. [3], the problem of leader election algorithm is closely related to the *Uncapacitated Facility Location Problem* (UFLP) [41], which is known to be NP-hard (see Section 2.4). As it is known, optimal solutions for NP-hard problems require some exponential component in the algorithm [105]. Consequently, in the worst case scenario, for every algorithm the message complexity is in $\mathcal{O}(n^2)$ (Table 4.1). For the best case, the results obtained are shown in Table 4.2 .

The Bully and Kordafshari algorithms stand out for providing good performance in the best case scenario, both for message complexity and turnaround time. In the worst case, which is caused by the number of concurrent elections, both present a considerable increase in message complexity. However, this is smaller in the case of the Kordafshari algorithm, thanks to the improvement introduced over the Bully algorithm to reduce the number of messages. Nevertheless, because of these improvements, the turnaround time of the Kordafshari algorithm is considerably incremented from $2t_{msg} + t_{wait}$ in the Bully algorithm to $4t_{msg}$.

In the case of the Vasudevan algorithm, the message complexity and turnaround time is highly dependent on the kind of network topology. The worst case for the message complexity is given in a mesh topology network, whereas the best case is given in a line topology network. On the contrary, the best turnaround time is given in a mesh topology network, whereas the worst turnaround time is given in a line topology. This fact highlights the existing trade-off between message complexity and turnaround time, which can also be seen in the Bully and Kordafshari algorithms.

Table 4.2 *Overview of the analysis performed on the election algorithms. In it, the theoretical message complexity and turnaround time in the best case are shown.*

	Theoretical $\mathcal{O}$ message complexity	Turnaround time
Adapted Bully	$2 \cdot (n-1) \in \Omega(1)$	$2t_{msg} + t_{wait}$
Adapted Kordafshari	$2 \cdot (n-1) \in \Omega(1)$	$2t_{msg} + t_{wait}$
Vasudevan	$3 \cdot (n-1) \in \Omega(1)$	$3 \cdot (n-1) \cdot t_{msg}$
Consensus	$n^2 + n - 2 \in \Omega(n^2)$	$3t_{msg}$
Voting	$n^2 + \frac{n}{2} \in \Omega(n^2)$	$3t_{msg}$

^{n} represents the number of nodes involved in the election.
$^{t_{msg}}$ represents the necessary time to transmit and process a message.
$^{t_{wait}}$ represents the time that in the Bully algorithm a node waits to receive *Acknowledge* messages before to declare itself the winner of the election. Note that $t_{wait} > t_{msg}$.

Finally, the new proposed algorithms, Consensus and Voting, present a consistent performance, which will not be affected by the network topology or concurrent elections. In comparison with the Bully and Kordafshari algorithms, this will be an advantage in highly dynamic networks, where the worst case scenario for the Bully and Kordafshari will be more frequent. However, in static and reliable networks, this represents a drawback, owing to the Bully and Kordafshari will tend to their best case scenarios. Regarding turnaround time, Consensus and Voting algorithms provide the best time of the five elections algorithms.

It should be noted that the theoretical analysis and empirical validation performed in this chapter have been done under ideal conditions: static and reliable networks have been assumed, where there are no dynamicity or message failure. Nevertheless, it helps to understand the behaviour of the algorithms presented and how different cases can influence in their performance. This will be useful to analyse and to understand the results that will be obtained in the evaluation of the architecture in non-reliable and mobile networks, presented in Chapter 7.

4.7 Summary

This chapter has introduced the Proteo architecture and its main components. The objective of Proteo is to provide a self-adaptive and resource-efficient solution to support the availability of services in dynamic topology environments. The proposal is based on service replication techniques together with a self-configuration approach for the activation/hibernation of the replicas of the service depending on relevant context information from the system.

Proteo is composed of three main elements: (1) *Monitoring Subsystem*, (2) *Context Manager Service*, and (3) *Replica Manager Service*. The responsibilities of these elements are closely related to the phases of MAPE-K autonomic loop of Autonomic Computing: the *Monitoring Subsystem* senses the context information in relation to the node in which is deployed; the *Context Manager Service*, in addition to process this information, will also be responsible for analysing it to decide when a new system adaptation is necessary; and the *Replica Manager Service* will be responsible for coming to an agreement with the rest of the *Replica Manager Services* deployed in the system to establish what replica will be activated.

This coordination is performed on the basis of a distributed host election algorithm, and the election is based on the basis of a dynamic score obtained through a utility function. To this regard, two new election algorithms are proposed: Consensus and Voting. These election algorithms are devised to operate in mobile environments with highly dynamic network topologies and under unreliable communication channels. Additionally, to compare the performance of the new two proposed election algorithms with existing approaches, the election algorithms of the Bully, Kordafshari, and Vasudevan have been selected. To this end, adaptations on the Bully and Kordafshari algorithms have been made. These have been necessary to base the elections, instead of on known IDs, on dynamic scores.

Finally, the performance of the five algorithms have been theoretically analysed, according to the indicated by Coulouris et al. [42], who establish that the performance of an election algorithm is measured by its total use of network bandwidth (the total number of messages sent during the election, i.e., message complexity), and the turnaround time (the number of serialized messages transmission). As a result, Consensus and Voting algorithms provide the better turnaround time, whereas the Kordafshary and Bully algorithms provide the best results regarding message complexity in their best case scenario.

As features that can be noted of Proteo are:

- It is **reactive** to context changes, owing to the *Context Manager Service* is responsible for analysing when a new system adaptation is necessary.

- It bases its decisions on **local knowledge**. To achieve this, a cross-layer approach to monitoring the network is used. Additionally, a heuristic method has been proposed in Section 4.5 to determine the more centric node in the network topology, on the basis of the information of its routing table. However, this requires a **proactive** routing protocol.

- The election of the best node to act as server is based on a **Utility Function**, which facilitates to adapt the election criteria and to take into consideration the resources of the nodes, as energy, providing a **resource-aware** solution.

Proteo architecture and its components will be completely detailed and modelled in the next chapter.

Chapter 5

Proteo Model

Chapter Abstract

In this chapter Proteo architecture, introduced in Chapter 4, is modelled using SysML. Proteo model is divided into four main parts: (1) *Operational Domain Model*, which contains the elements that can be found in the system's operating environment; (2) *Information Model*, which groups the definition of data items and value types used in the system; (3) *Structural Model*, which contains the elements that compose the system; and (4) *Behavioural Model*, which includes the elements that describe the control structure of the system. These parts together define in detail the components of Proteo architecture, their relations and their behaviour during system operation.

Chapter Contents

5.1 Introduction

The package diagram in Fig. 5.1 presents the organization for this model. All the models are contained in the top-level model package Proteo Model. The model is divided into four main packages, which primarily organize the model elements on the basis of their nature.

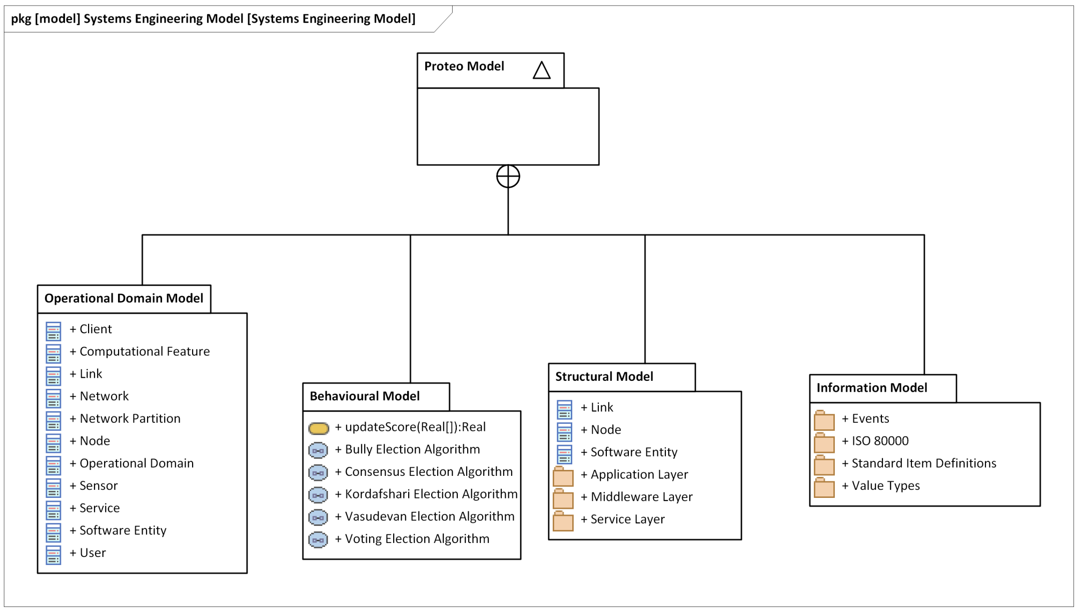


Fig. 5.1 *Package diagram showing how the model is organized into packages.*

These packages are:

- **Operational Domain Model** (Section 5.2): It contains the elements that can be found in the system's operating environment and their main relations. The objective is to describe the operating conditions in which the system will operate.
- **Information Model** (Section 5.3): It groups the definition of data items and value types used in the system.
- **Structural Model** (Section 5.4): It contains the elements that compose the system. This model intends to provide a view of the system independent of time, emphasizing the structure of the elements, relationships, and attributes.
- **Behavioural Model** (Section 5.5): Also known as *Dynamic Model*, this package contains the elements that describe the control structure of the system. The objective is to describe how the system behaves when it is changing.

These packages are examined in detail in the following sections, to completely describe the model of the system designed.

5.2 Operational Domain Model

The block definition diagram of Fig. 5.2 shows the main elements that compose the system's operating environment. These elements may either directly or indirectly interact with the system. This diagram expects only to show the hierarchy of those elements and their whole-part relationships. For example, a user may be related to a mobile node, yet it is not part of the mobile node. This kind of relationships, i.e., reference properties, will be defined in the Structural Model in Section 5.4.

As can be seen in the diagram of Fig. 5.2, the operational domain of this model is made up of three main elements: *User*, *Software Entity* and *Network*.

Although the interaction of a user with a software system is, generally, a relevant activity to take into consideration in the software design process, in the context of this work, the *User* only interacts indirectly with the proposed system. Specifically, the *User* is responsible, for the most part, of the network topology changes owing to its movements when he or she is carrying a mobile node. In this work, it is considered that user's movement cannot be influenced or modified, and network topology changes produced by this reason are assumed as unavoidable.

Software Entity represents a generalisation of any kind of software that could be found in the system. In this work, the *Service* and *Client* software entities are mainly considered. This abstraction is made because this work aims to enhance the quality attributes of the services, and thus the quality of client-server relations. Moreover, other software elements, such as intelligent agents, applications or components, can be seen as service providers (*Services*) or service requesters (*Clients*) at some point in their life-cycle. For these reasons, the generalization is tagged as *incomplete* and *overlapping*. This is a flexible representation that allows to a software entity, besides of being considered an agent or micro-service, to act at the same time as a service provider to other software entity, and as a service requester to a different or the same software entity.

Finally, the *Network*, which is made up of *Nodes* and *Links*. It can be divided in different *Network Partitions*, or in only one *Network Partition* when all the nodes of the *Network* are connected. A *Network Partition* can only belong to one *Network*, and it is composed of a least one *Node*. In such case, there is no any *Link* in the *Network Partition*. If there are more nodes in the partition, these are connected between them, by means of one-hop links or multi-hop links. Every *Node* belongs to only one *Network partition*.

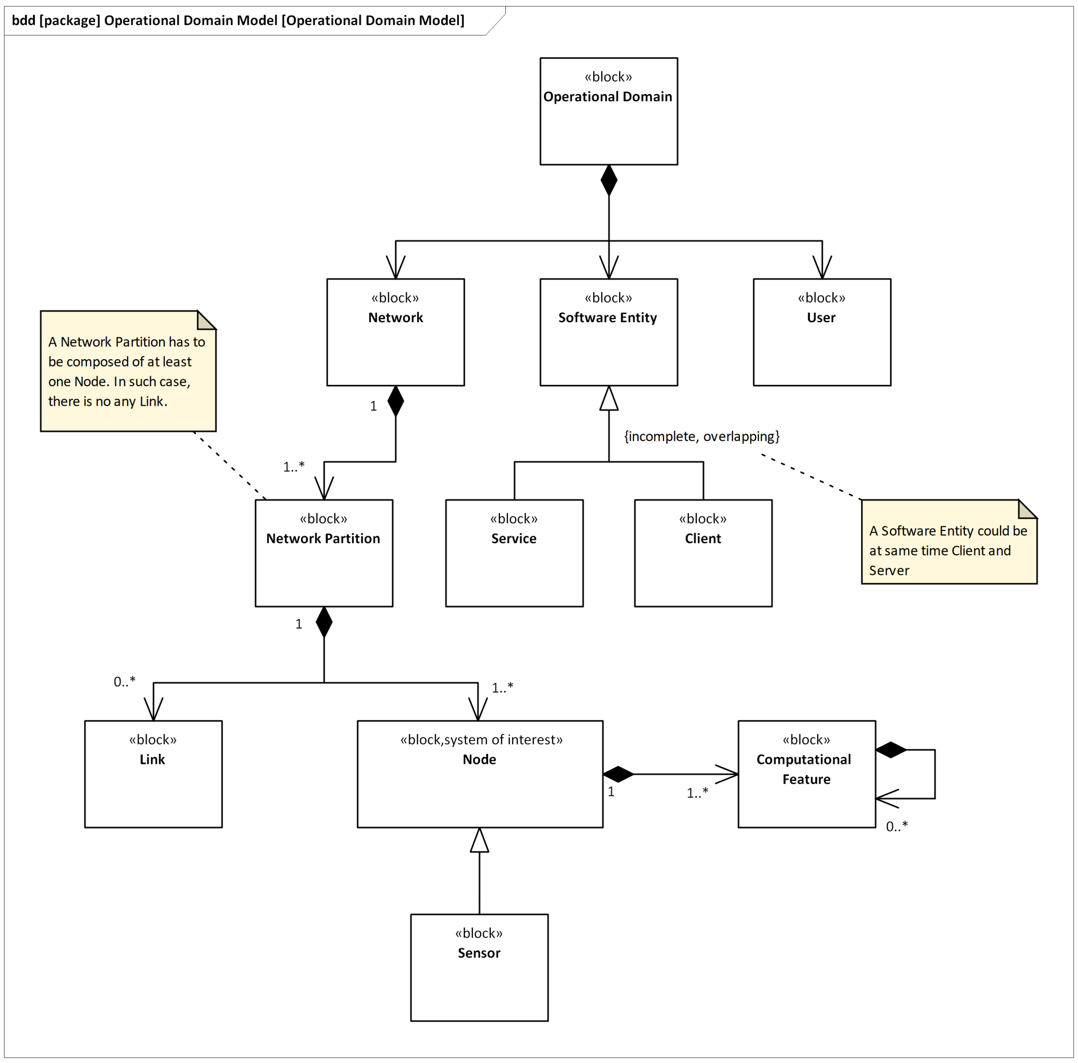

bdd [package] Operational Domain Model [Operational Domain Model]
«block» Operational Domain
«block» Network
«block» Software Entity
«block» User
A Network Partition has to be composed of at least one Node. In such case, there is no any Link.
1
1..*
{incomplete, overlapping}
«block» Network Partition
«block» Service
«block» Client
A Software Entity could be at same time Client and Server
1
0..*
1..*
«block» Link
«block,system of interest» Node
«block» Computational Feature
1
1..*
0..*
«block» Sensor

Fig. 5.2 *Block definition diagram showing the operational domain model, where Node is the system of interest.*

A *Node* is made up of *Computational Features* (battery, storage or memory, among others). These *Computational Features* can be in turn composed of other *Computational Features*. For example, the estimated duration of the battery could be calculated from screen brightness, CPU usage, phone signal strength, and other features. Noted that the *Node* entity is stereotyped as *system of interest*. ISO/IEC/IEEE 15288:2015 (Systems and software engineering – System life cycle processes) defines system of interest as *"the system whose life cycle is under consideration"* [76].

5.3 Information Model

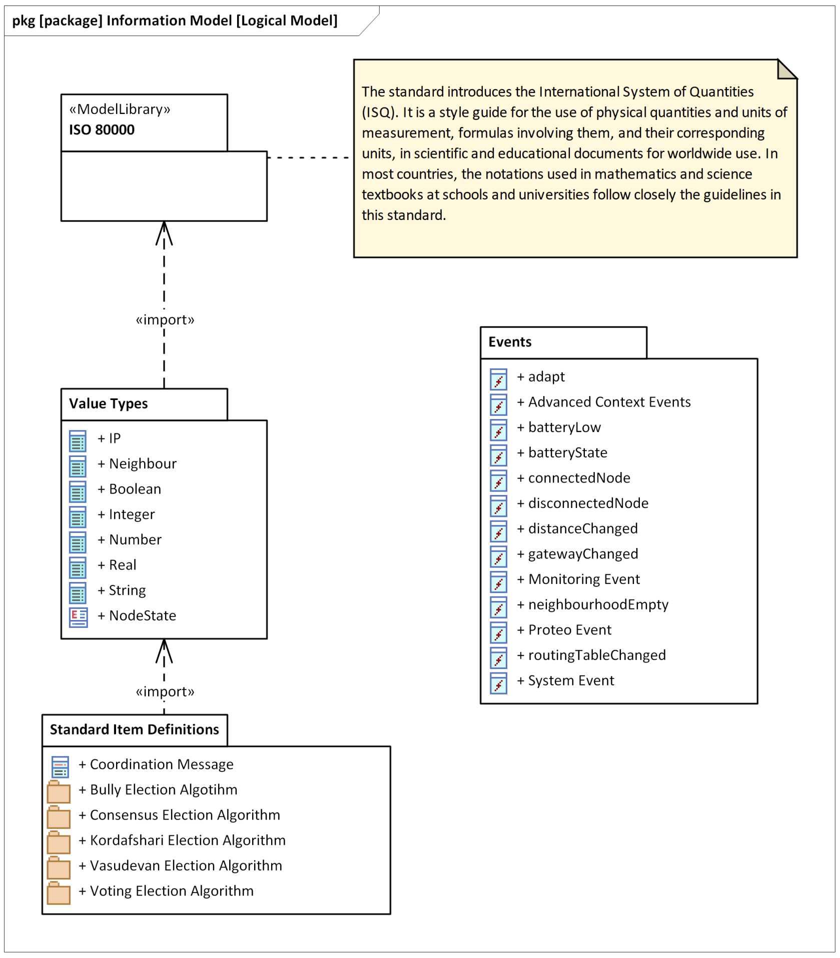

pkg [package] Information Model [Logical Model]
«ModelLibrary»
ISO 80000
The standard introduces the International System of Quantities (ISQ). It is a style guide for the use of physical quantities and units of measurement, formulas involving them, and their corresponding units, in scientific and educational documents for worldwide use. In most countries, the notations used in mathematics and science textbooks at schools and universities follow closely the guidelines in this standard.
«import»
Value Types
+ IP
+ Neighbour
+ Boolean
+ Integer
+ Number
+ Real
+ String
+ NodeState
«import»
Standard Item Definitions
+ Coordination Message
+ Bully Election Algotihm
+ Consensus Election Algorithm
+ Kordafshari Election Algorithm
+ Vasudevan Election Algorithm
+ Voting Election Algorithm
Events
+ adapt
+ Advanced Context Events
+ batteryLow
+ batteryState
+ connectedNode
+ disconnectedNode
+ distanceChanged
+ gatewayChanged
+ Monitoring Event
+ neighbourhoodEmpty
+ Proteo Event
+ routingTableChanged
+ System Event

Fig. 5.3 *Package diagram showing how the Information Model is organized into packages.*

Information Model is divided into three main packages (Fig. 5.3):

- **Value Types** (Section 5.3.1): It contains tue main value type definitions used in Proteo. This package imports the standard library ISO 80000. This library defines, inter alia, the units of measurement of the International system of Quantities.
- **Events** (Section 5.3.3): In this package are defined the context events, i.e., signals in SysML, that are triggered and managed in the system by the *Monitoring Subsystem* and *Context Manager Service*.
- **Standard Item Definitions** (Section 5.3.2): In this package the items that flow between the parts of the system are defined. These are mainly related with the election algorithms and the *Replica Manager Service*.

5.3.1 Value Types

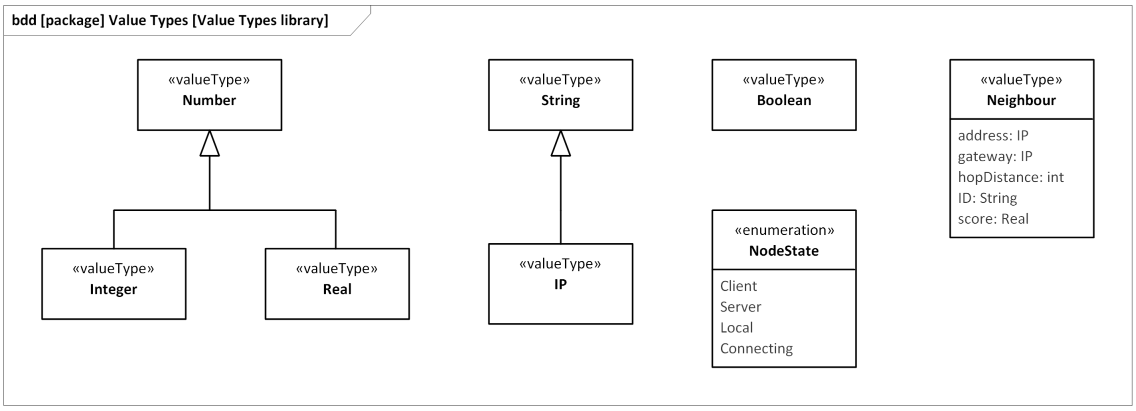


Fig. 5.4 *Block definition diagram showing the value types defined within Proteo system.*

Within the Value Types package (see Fig. 5.4), three elements can be highlighted:

- *IP value type*: It represents, through the specialization of the String value type, the IP of a node.
- *NodeState*: It represents the four states in which a node can be during the operation of Proteo:
  - *Client*: A node is in *Client* state when it has established a successful client server connection with a node in *Server* state in its network partition.
  - *Server*: A node is in *Server* state when it is hosting an active service replica and it is providing service to the rest of the nodes of its network partition.

– *Local*: A node is in *Local* state when it has no connection with other nodes in the network. This is, the node is isolated and thus is the only node in its *Network Partition*.

– *Connecting*: A node is in *Connecting* state when the self-configuration of the network and the client-server scheme is being carried out.

• *Neighbour*: It represents the local information that a node *N* has about a reachable node *P* in its *Network Partition*. It is defined by the information about the node *P* that the node *N* has in its routing table (*address*, *gateway* and *hopDistance*). Additionally, a neighbour (*P*) is defined by its unique *ID* and a *score*. This *score* represents how suitable the neighbour *P* is to host an active service replica.

5.3.2 Signals (Context Events)

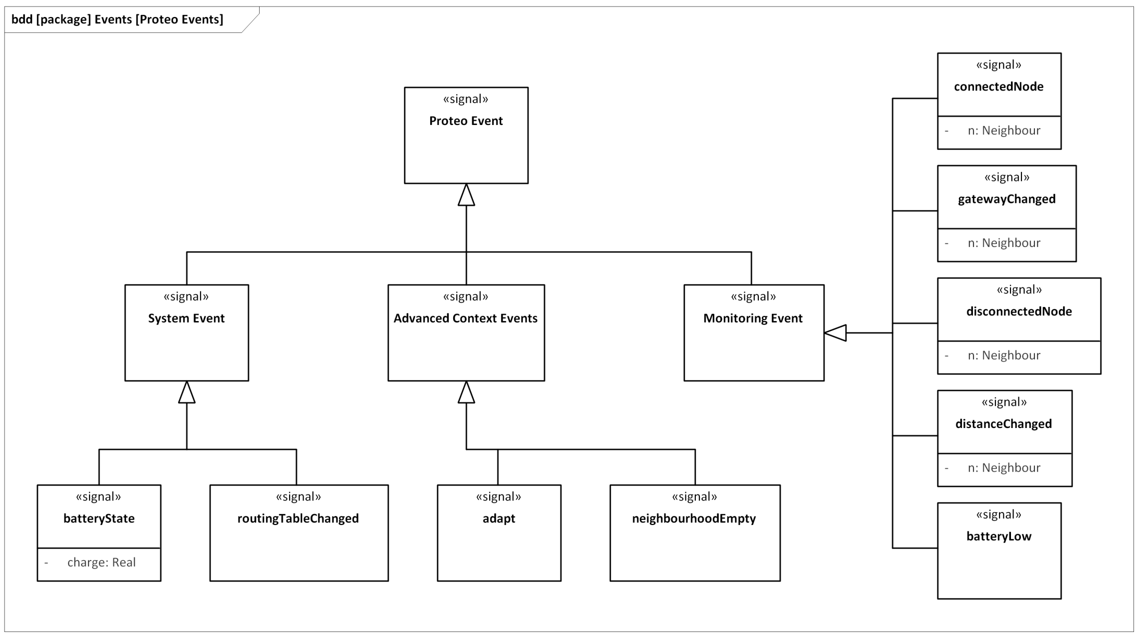


Fig. 5.5 *Block definition diagram showing the hierarchy of signals handled within Proteo system.*

The events in Proteo system can be classified into three categories (see Fig. 5.5) according to their source:

• *System Events*: Between the different signals that can arise in the node operating system or the lower layers of the middleware, Proteo platform takes into consideration those that announce changes on the routing table (*routingTableChanged* signal) and in the battery of the node (*batteryState* signal), if this is a battery supplied node.

- *Monitoring Events*: These are originated in the monitoring services of Proteo. This spreads basic information in the system, such as a node becomes disconnected (*disconnectedNode* signal) or the battery of the node has reached some specified threshold (*batteryLow* signal). Note that neighbour-related signals (*connectedNode*, *disconnectedNode*, *distanceChanged* and *gatewayChanged*) provide the information of the node to which the signal refers by means of a *Neighbour* value type, defined in the *Value Types* package (see Fig. 5.4).

- *Advanced Context Signals*: These signals are originated in the Context Manager Service and they provide elaborated context information directly related to the adaptation needs of the system. These signals are *adapt*, which indicates to the system when it is necessary to perform a revision of the replication scheme; and *neighbourhoodEmpty*, which indicates to the system that the node has no any remain reachable neighbour. The abstraction provided by the *adapt* signal allows to redefine the trigger policies of the signal for each specific scenario, and thus the reactivity or reactivity degree of Proteo.

As noted in Chapter 4, in this work, it has been proposed to monitor the battery of the node and the network topology, since they are considered transversal features. However, the proposed system could be extended according to the requirements of the application scenario, adding new signal definitions to those presented in Fig. 5.5.

5.3.3 Standard Item Definitions

In SysML, **items** are seen as the elements that flow across connectors between blocks or parts. In this package, the items that flow on Proteo communication are defined. The main item is the *Coordination Message*. This is a generic message used to perform the host election algorithm. The specific coordination messages will depend on the specific election algorithm used. Hence *Standard Item Definitions* package is divided into five sub-packages (Fig. 5.6): *Bully Election Algorithm*, *Kordafshari Election Algorithm*, *Vasudevan Election Algorithm*, *Consensus Election Algorithm* and *Voting Election Algorithm*. This is, one for each election algorithm, where their related coordination messages are defined. These messages are defined as a specialization of the *Coordination Message* item, in which the *IP* of the sender node is defined.

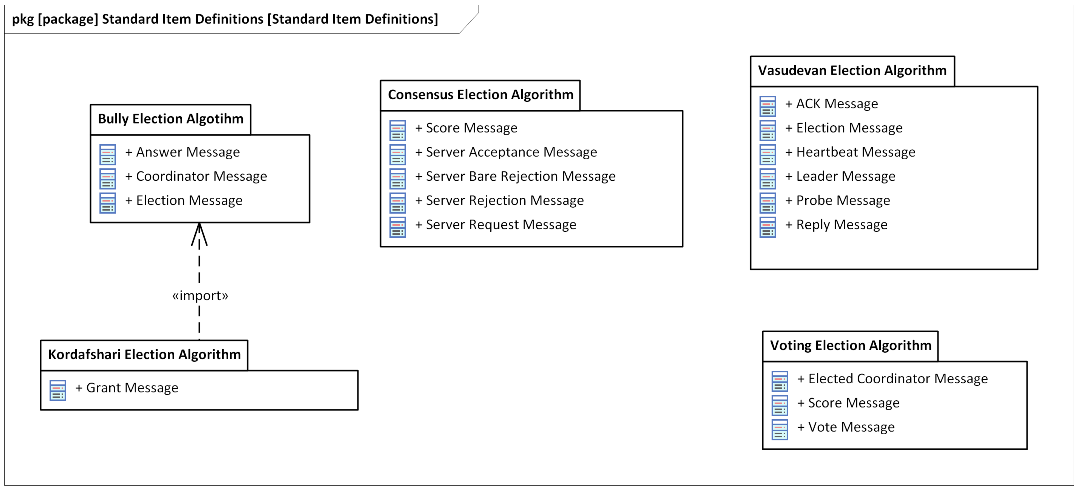

pkg [package] Standard Item Definitions [Standard Item Definitions]
Bully Election Algotihm
+ Answer Message
+ Coordinator Message
+ Election Message
«import»
Kordafshari Election Algorithm
+ Grant Message
Consensus Election Algorithm
+ Score Message
+ Server Acceptance Message
+ Server Bare Rejection Message
+ Server Rejection Message
+ Server Request Message
Vasudevan Election Algorithm
+ ACK Message
+ Election Message
+ Heartbeat Message
+ Leader Message
+ Probe Message
+ Reply Message
Voting Election Algorithm
+ Elected Coordinator Message
+ Score Message
+ Vote Message

Fig. 5.6 *Package diagram showing how the Item Definitions are grouped into different packages according to the algorithm.*

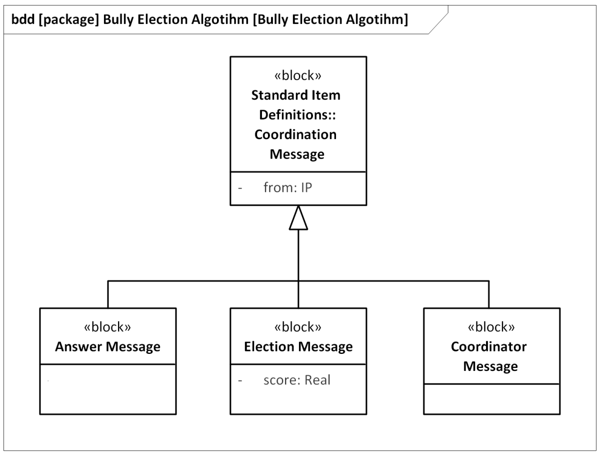

bdd [package] Bully Election Algotihm [Bully Election Algotihm]
«block»
Standard Item Definitions:: Coordination Message
- from: IP
«block»
Answer Message
«block»
Election Message
- score: Real
«block»
Coordinator Message

Fig. 5.7 *Block definition diagram showing the Item Definitions for the Bully Election algorithm.*

Bully Election Algorithm Items

As described in Section 2.4.1, Bully algorithm is based on three kind of messages, modelled as items in Fig. 5.7:

- *Election Message*: It is used to announce an election. As part of the adaptation proposed in Section 4.6.2, this message contains now the current *score* of the sender node.
- *Answer Message*: It is used to respond to the Election message.
- *Coordinator Message*: It is used to announce victory by the winner of the election.

Kordafshari Election Algorithm Items

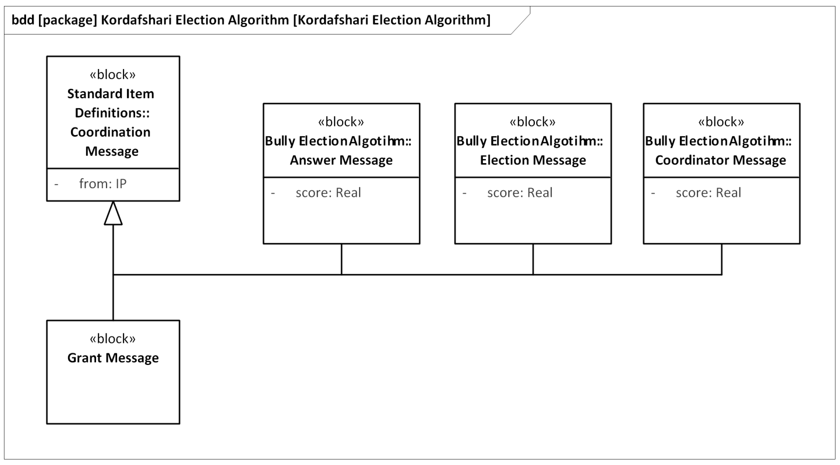

bdd [package] Kordafshari Election Algorithm [Kordafshari Election Algorithm]
«block»
Standard Item Definitions:: Coordination Message
- from: IP
«block»
Bully Election Algotihm:: Answer Message
- score: Real
«block»
Bully Election Algotihm:: Election Message
- score: Real
«block»
Bully Election Algotihm:: Coordinator Message
- score: Real
«block»
Grant Message

Fig. 5.8 *Block definition diagram showing the Item Definitions for the Kordafshari Election algorithm.*

Since Kordafshari (see Section 2.4.2) proposes an extension of Bully, it imports the messages defined in *Bully Election Algorithm* package (Fig. 5.7) and one new message item is defined, the *Grant Message* (Fig. 5.8).

- *Grant Message*: It is used by the node that initiates the election process to communicate to the node that has sent the *Answer* Message with the higher score that it has won the election.

Vasudevan Election Algorithm Items

As described in Section 3.3.2, Vasudevan algoritm implements three basic types of messages (Fig. 5.9):

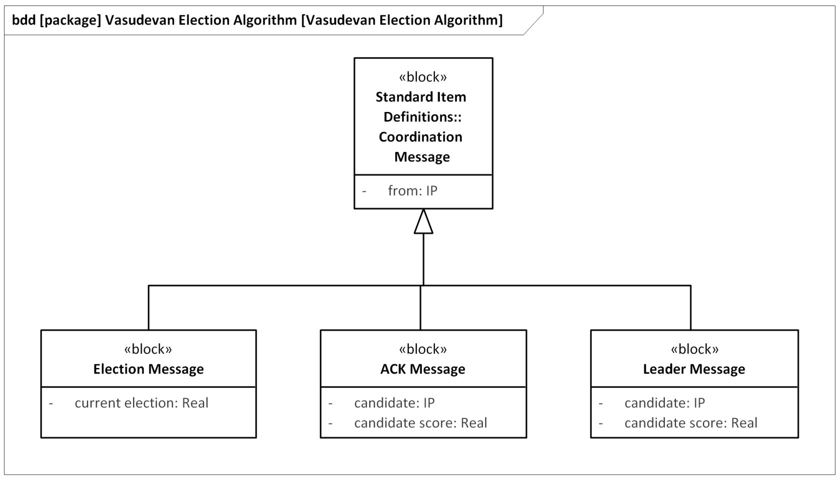


Fig. 5.9 *Block definition diagram showing the Item Definitions for the Vasudevan Election algorithm.*

- *Election Message*: It is used to grow the spanning tree and start the election. This message includes a *timestamp*, determining the moment in which the election started, to avoid concurrent elections.
- *ACK Message*: It is used to answer to an *Election* Message and shrink the spanning tree. This message contains information about the best node of the subtree.
- *Leader Message*: It is used by the root node of the spanning tree to announce the winner node. It contains the information about the elected node.

Additionally, Vasudevan uses specific periodic messages to known if a node is still alive. However, as exposed in Section 4.6.2, these are now unnecessary owing to the integration of the algorithm in Proteo, which will manage the discovery and reachability of the nodes (see Section 5.4.1).

Consensus Election Algorithm Items

Fig. 5.10 defines the messages in which Consensus is based to perform the election (see Section 4.6.1):

- *Score Message*: Through this message, a node communicates its score. The score is expressed through a float number, in the range 0 and 1, where 1 represents the best score attainable.
- *Server Request Message*: It is used when the requesting node wants to establish a client-server connection with the receiver node.

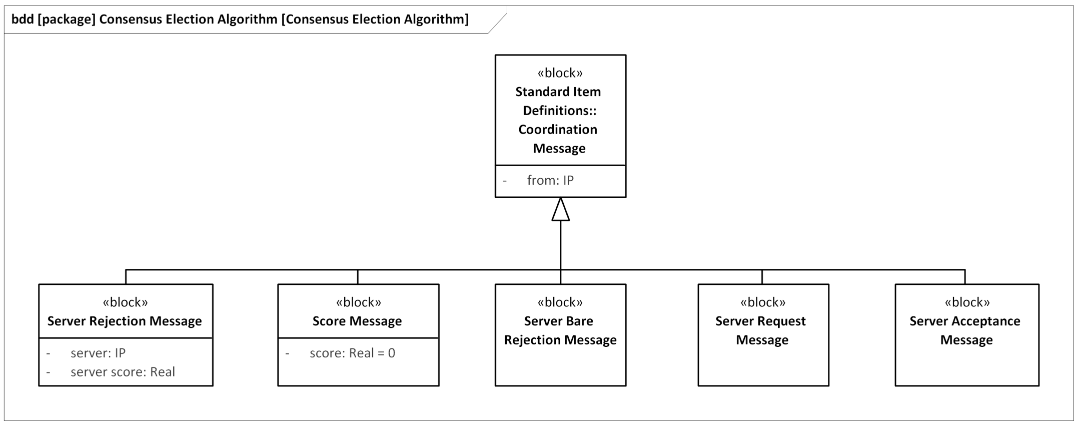

bdd [package] Consensus Election Algorithm [Consensus Election Algorithm]
«block» Standard Item Definitions:: Coordination Message
- from: IP
«block» Server Rejection Message
- server: IP
- server score: Real
«block» Score Message
- score: Real = 0
«block» Server Bare Rejection Message
«block» Server Request Message
«block» Server Acceptance Message

Fig. 5.10 *Block definition diagram showing the Item Definitions for the Consensus Election algorithm.*

- *Server Rejection Message*: It is used to reject a *Server Request* message. In the message, information about the current server (its *IP* and its *score*) is attached.
- *Server Bare Rejection Message*: This message has the same function that a *Server Rejection* message but without additional information.
- *Server Acceptance Message*: It is used to accept a Server Request message.

Voting Election Algorithm Items

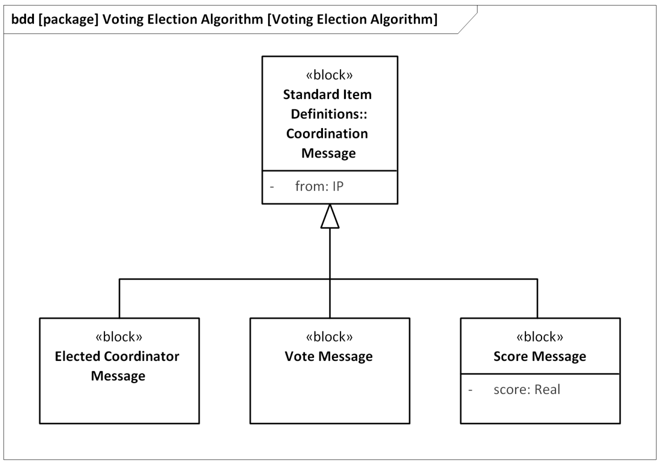

bdd [package] Voting Election Algorithm [Voting Election Algorithm]
«block» Standard Item Definitions:: Coordination Message
- from: IP
«block» Elected Coordinator Message
«block» Vote Message
«block» Score Message
- score: Real

Fig. 5.11 *Block definition diagram showing the Item Definitions for the Voting Election algorithm.*

As exposed in Section 4.6.1, Voting election algorithm is based on three kinds of messages (Fig. 5.11):

- *Score Message*: Identical, in function and structure, to the *Score* message used in Consensus election algorithm.
- *Vote Message*: It is used by a node to emit a vote to the best-ranked node.
- *Elected Coordinator Message*: It is used by a node to announce its victory in the elections.

5.4 Structural Model

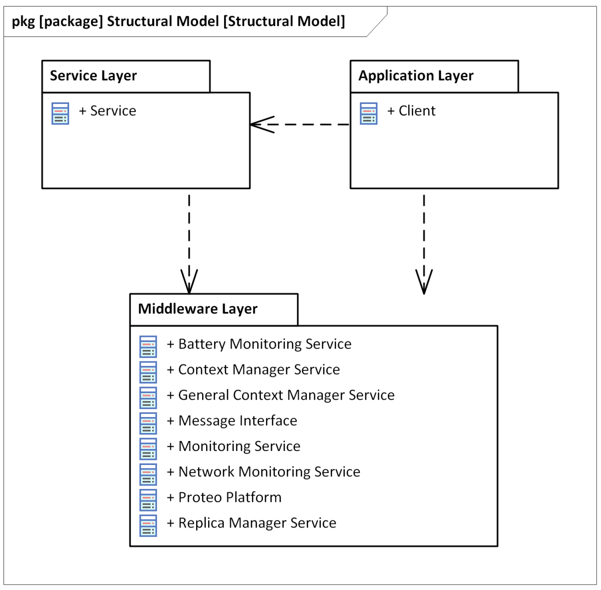

pkg [package] Structural Model [Structural Model]
Service Layer
+ Service
Application Layer
+ Client
Middleware Layer
+ Battery Monitoring Service
+ Context Manager Service
+ General Context Manager Service
+ Message Interface
+ Monitoring Service
+ Network Monitoring Service
+ Proteo Platform
+ Replica Manager Service

Fig. 5.12 *Package diagram showing how the distribution into packages of the Structural Model of Proteo platform.*

Structural Model is used to represent the structure, relationships and attributes from the elements of the system and it is divided into three packages (Fig. 5.12). This division corresponds to a traditional client-server architecture, where three layers can be found:

- **Service Layer**: It contains the different **task services** of the system, i.e., the back-end entities. In SOA approach, these are defined as services within a specific functional context and with a specific purpose, less oriented to reuse. The objective of Proteo system is to guarantee and support the quality attributes of the services allocated to this layer.
- **Application Layer**: It contains the front-end entities of the system, which are based both on the *Service* and *Middleware* Layer.

- **Middleware Layer**: The middleware layer provides support to *Service* and *Application* Layer. A middleware facilitates the development of high-level software and homogenizes the subjacent technologies. There are many functionalities provided by the *Middleware* Layer. However, this work is focused on those elements that make possible to guarantee and support the quality attributes of the task services, without waiving to the possibility of extending this layer with other elements related with other objectives or middleware functionalities.

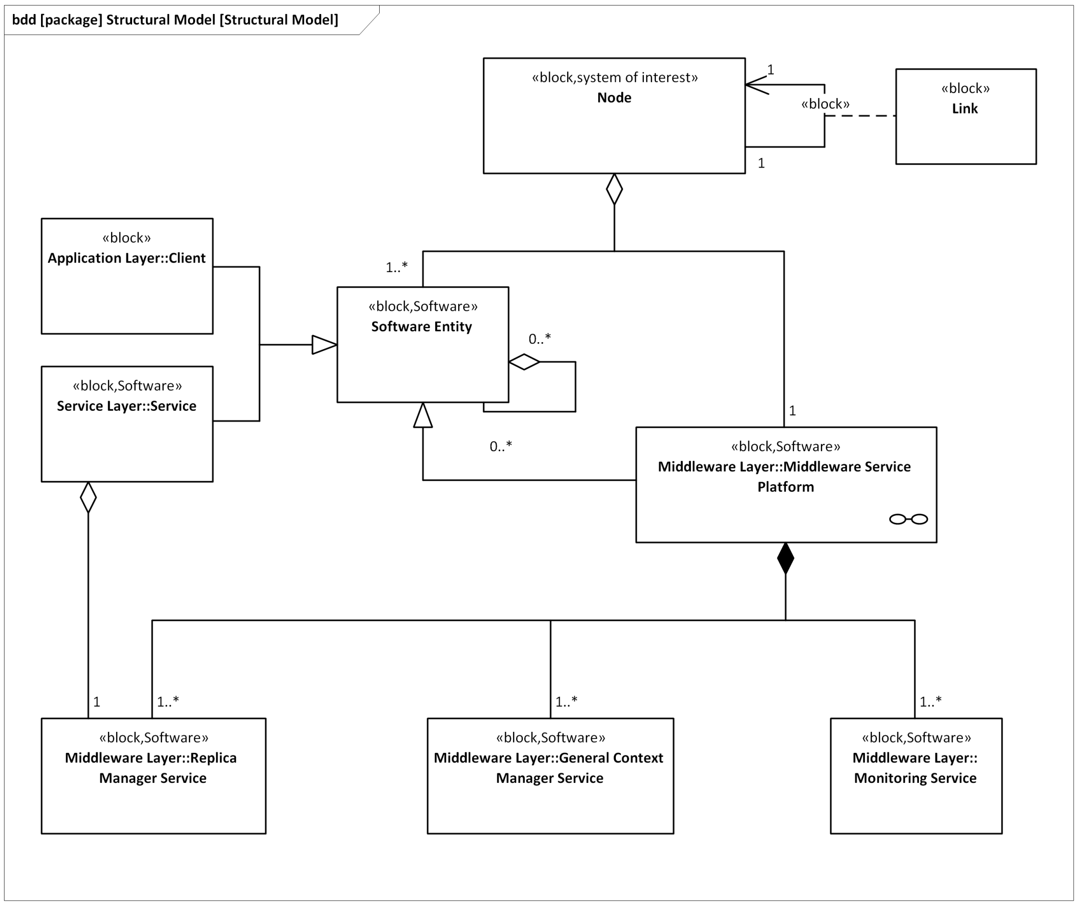

bdd [package] Structural Model [Structural Model]
«block,system of interest» Node
«block» Link
«block»
1
1
«block» Application Layer::Client
«block,Software» Software Entity
«block,Software» Service Layer::Service
1..*
0..*
0..*
1
«block,Software» Middleware Layer::Middleware Service Platform
1
1..*
1..*
1..*
«block,Software» Middleware Layer::Replica Manager Service
«block,Software» Middleware Layer::General Context Manager Service
«block,Software» Middleware Layer:: Monitoring Service

Fig. 5.13 *Block definition diagram showing the relation between the structural elements of Proteo model.*

In Fig. 5.2, within Section 5.2, the main elements that compose the system's operating environment and their hierarchy and whole-part relationships were presented. Fig. 5.13 shows a block definition diagram where the structural relations of the main elements related to the *Node* block, *system of interest* in Fig. 5.2 are shown.

A *Node* can be connected to other nodes of the same partition through a *Link*. In this way, when a *Link* connects two nodes, they are directly connected (one-hop link). If the nodes are not directly connected, they still can be connected by a set of links shared with other in common nodes (multi-hop link).

A *Node* contains one instance of the services of Proteo and at least one *Software Entity*. This *Software Entity* can be a *Service* and/or a *Client*, as described in Section 5.2, and it could be composed on the basis of other *Software Entities*. In this way, service composition (orchestration or choreography) is allowed. *Middleware Service Platform* is made up of three main kinds of blocks: (1) *Monitoring Service*, (2) *Context Manager Service*, and (3) *Replica Manager Service*. The latter manages the replication scheme of a *Service*.

The services that made up *Middleware Service Platform* will be described in more detail below. At this point, it worth noting the relations between them. A *Replica Manager Service* manages the replicas of only one *Service*. This is, if there are different services in the system, *Middleware Service Platform* will have one *Replica Manager Service* for each of them. On the contrary, the *Context Manager Service* could support more than one *Replica Manager Service*. Finally, *Middleware Service Platform* could be composed of different *Monitoring Services*, according to the monitoring requirements for a particular scenario.

Proteo services will interact by three main ways: (1) a *publish-subscribe* paradigm, according to the events, or signals, defined in Section 5.3.2; (2) a *message passing* paradigm, according to the items defined in Section 5.3.3; and (3) a *request-response* paradigm, where the operations defined by the services are invoked. These operations are represented in the block definition diagram of Fig. 5.14 and will be described together with the *Middleware Service Platform* services.

Regarding signal interaction, in SysML, this is shown using the *«signal»* stereotype in the block. If a block requires a signal, the keyword **reqd** is placed before the name of the signal. On the contrary, if the block provides a signal, the keyword **prov** is placed before the name of the signal. The keyword **provreqd** is used when a signal is both provided and required.

5.4.1 Monitoring Service

The *Monitoring Service* block represents a service that senses the context information in relation to the node in order to detect potential events that could affect the availability of a service. Although the implementation of each monitoring service will depend on the kind of resource or element that it is being monitored, they have in common that they are based on event communication.

Fig. 5.14 shows the specialization of the *Monitoring Service* block into two monitoring services: *Battery Monitoring Service* and *Network Monitoring Service*.

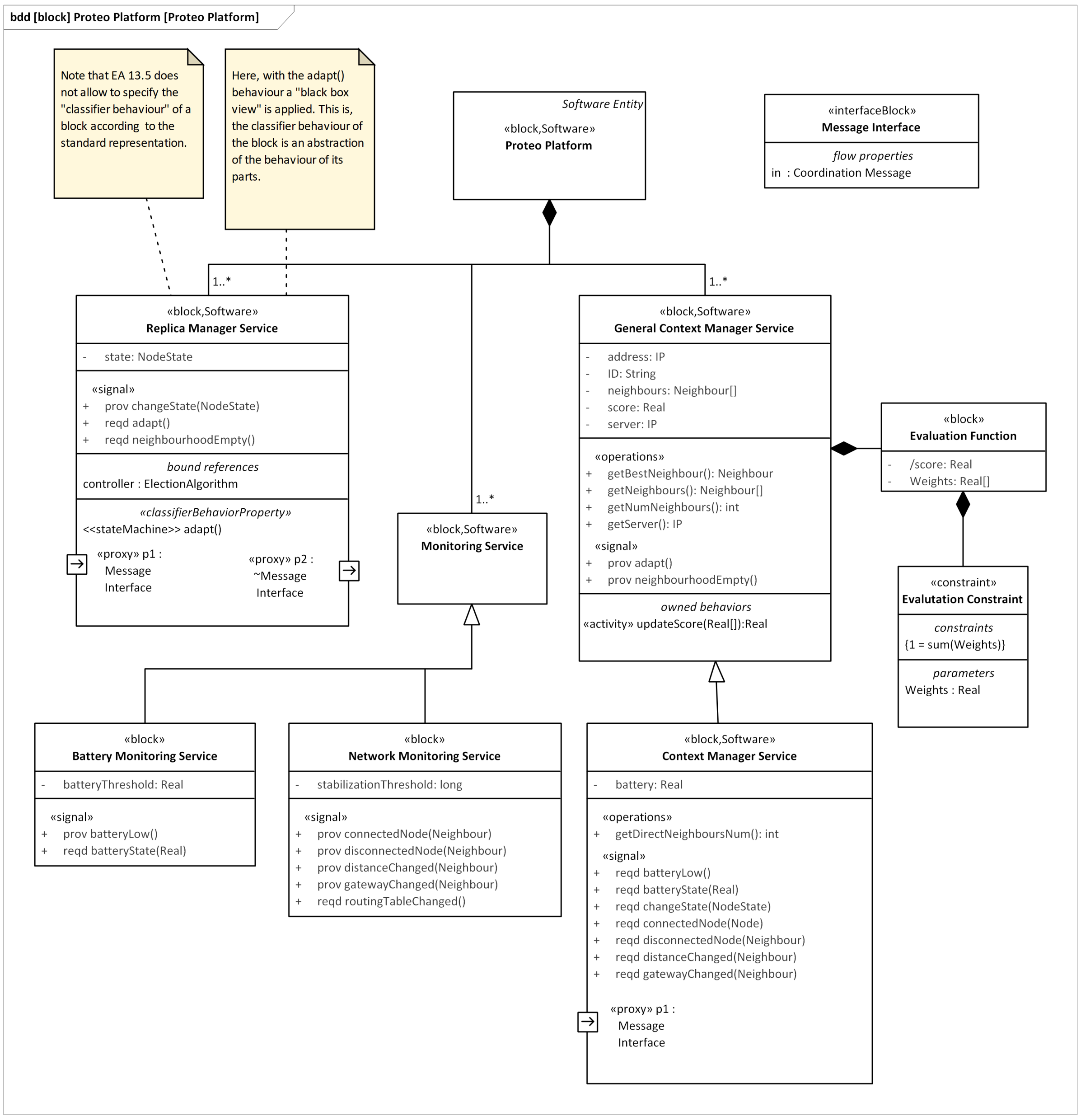

bdd [block] Proteo Platform [Proteo Platform]
Note that EA 13.5 does not allow to specify the "classifier behaviour" of a block according to the standard representation.
Here, with the adapt() behaviour a "black box view" is applied. This is, the classifier behaviour of the block is an abstraction of the behaviour of its parts.
Software Entity
«block,Software» Proteo Platform
«interfaceBlock» Message Interface
flow properties
in : Coordination Message
1..*
1..*
1..*
«block,Software» Replica Manager Service
- state: NodeState
«signal»
+ prov changeState(NodeState)
+ reqd adapt()
+ reqd neighbourhoodEmpty()
bound references
controller : ElectionAlgorithm
«classifierBehaviorProperty»
<<stateMachine>> adapt()
«proxy» p1 : Message Interface
«proxy» p2 : ~Message Interface
«block,Software» Monitoring Service
«block,Software» General Context Manager Service
- address: IP
- ID: String
- neighbours: Neighbour[]
- score: Real
- server: IP
«operations»
+ getBestNeighbour(): Neighbour
+ getNeighbours(): Neighbour[]
+ getNumNeighbours(): int
+ getServer(): IP
«signal»
+ prov adapt()
+ prov neighbourhoodEmpty()
owned behaviors
«activity» updateScore(Real[]):Real
«block» Evaluation Function
- /score: Real
- Weights: Real[]
«constraint» Evalutation Constraint
constraints
{1 = sum(Weights)}
parameters
Weights : Real
«block» Battery Monitoring Service
- batteryThreshold: Real
«signal»
+ prov batteryLow()
+ reqd batteryState(Real)
«block» Network Monitoring Service
- stabilizationThreshold: long
«signal»
+ prov connectedNode(Neighbour)
+ prov disconnectedNode(Neighbour)
+ prov distanceChanged(Neighbour)
+ prov gatewayChanged(Neighbour)
+ reqd routingTableChanged()
«block,Software» Context Manager Service
- battery: Real
«operations»
+ getDirectNeighboursNum(): int
«signal»
+ reqd batteryLow()
+ reqd batteryState(Real)
+ reqd changeState(NodeState)
+ reqd connectedNode(Node)
+ reqd disconnectedNode(Neighbour)
+ reqd distanceChanged(Neighbour)
+ reqd gatewayChanged(Neighbour)
«proxy» p1 : Message Interface

Fig. 5.14 *Block definition diagram showing in detail the structural properties of the entities that compose Proteo.*

5.4.2 Context Manager Service

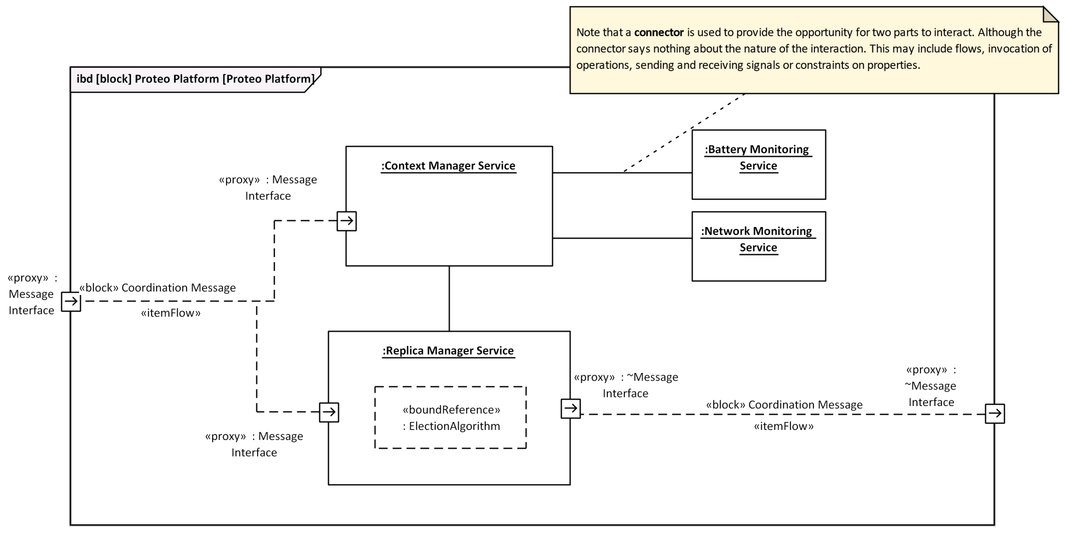

Note that a connector is used to provide the opportunity for two parts to interact. Although the connector says nothing about the nature of the interaction. This may include flows, invocation of operations, sending and receiving signals or constraints on properties.
ibd [block] Proteo Platform [Proteo Platform]
:Context Manager Service
:Battery Monitoring Service
:Network Monitoring Service
:Replica Manager Service
«boundReference» : ElectionAlgorithm
«proxy» : Message Interface
«proxy» : Message Interface
«block» Coordination Message
«itemFlow»
«proxy» : Message Interface
«proxy» : ~Message Interface
«block» Coordination Message
«itemFlow»
«proxy» : ~Message Interface

Fig. 5.15 *Internal block diagram showing the interaction flow of Coordination Message items between the parts of Middleware Service Platform.*

In Fig. 5.14 the detailed description of the *Context Manager Service* is provided in the *Context Manager Service* block, which specializes the *General Context Manager Service* block. The implementation of this service is dependent on the application area. In the design provided, the *Context Manager Service* manages the information received by both the *Battery Monitoring Service* and the *Network Monitoring Service*. Although these features (battery and network) are transversal to the application scenarios for which the proposal has been devised, the *Context Manager Service* specification is not limited to them and it can be extended through specialization.

Context Manager Service feeds from two sources of information: that provided by the monitoring services, obtained through the events interface, and the provided by other nodes in the *Network Partition*, obtained by the *Message Interface* (Fig. 5.14). This relationship is shown in the internal block diagram of Fig. 5.15. The *Context Manager Service* block shows two kinds of relations in Fig. 5.15: an incoming flow connector through the message interface *p1* in Fig. 5.14, by which it will obtain information about other nodes in the network; and a connector with each *Monitoring Service* and the *Replica Manager Service*. Note that in SysML a **connector** represents an opportunity for two parts to interact. Nevertheless, the connector says nothing about the nature of the interaction. It may include flows, invocation of operations, sending and receiving signals or constraints on properties. In this specific case, the relationship of the *Context Manager Service* with the monitoring services is exclusively through sending

Table 5.1 *Context Manager Service attributes, descriptions and modifier actions.*

Attribute	Value Type	Description	Modifiable	Modified by
address	*IP*	IP address of the node	No	-
battery	*Real*	Current charge of the battery of the node	Yes	the system signal batteryState(Real)
ID	*Real*	Unique identifier of the node	No	-
neighbours	*Neighbour[]*	Array of the current neighbours of the node	Yes	the Network Monitoring Service signals connectedNode(Neighbour), disconnectedNode(Neighbour), distanceChanged(Neighbour) and gatewayChanged(Neighbour))
				the reception of a Coordination Message containing the score of a neighbour through the proxy port p1
score	*Real*	Last calculated score of the node	Yes	the invocation of the updateScore(Real[]):Real operation
server	*IP*	IP address of the current server	Yes	the reception of a Coordination Message announcing a new server through the proxy port p1

and receiving signals. In the case of the relationship with the *Replica Manager Service*, this relation will include both sending and receiving signals, and also invocation of operations.

The context information stored and maintained by *Context Manager Service* is described in Table 5.1 and specified in the block definition diagram of Fig. 5.14. As descriptions on Table 5.1 show, this information can be modified by the reception of monitoring and system events, the reception of *Coordination Messages* through the message interface *p1* and the invocation of *Context Manager Service* operations.

5.4.3 Replica Manager Service

As introduced in Section 4.4.3, the *Replica Manager Service* (Fig. 5.14) performs the election algorithm to achieve an efficient configuration for the activation/hibernation of the replicas of the service and manages the state of the node (*state:NodeState* in Fig. 5.14), which is modified by the **classifier behaviour**[1] of the block, defined by the *adapt()* state machine. The *adapt()* behaviour is a **black box view**, this is, the behaviour is an abstraction of the behaviour of its parts. In this case, the *adapt()* state machine is implemented by the *controller* part in Fig. 5.14.

It should be noted that *controller* part is modelled as a **bound reference**. This allow to define variants of the *Replica Manager Service* according to the specific controller part implemented (Fig. 5.16 and Fig. 5.17).

5.5 Behavioural Model

This package of the Proteo Model contains the elements that describe the control structure of the system. In this section, there will be described: (1) the behaviour of the services that

[1] In SysML, the classifier behaviour represents the main behaviour of the block, which starts running at the beginning of the lifetime of the block and usually finishes at the end of its lifetime.

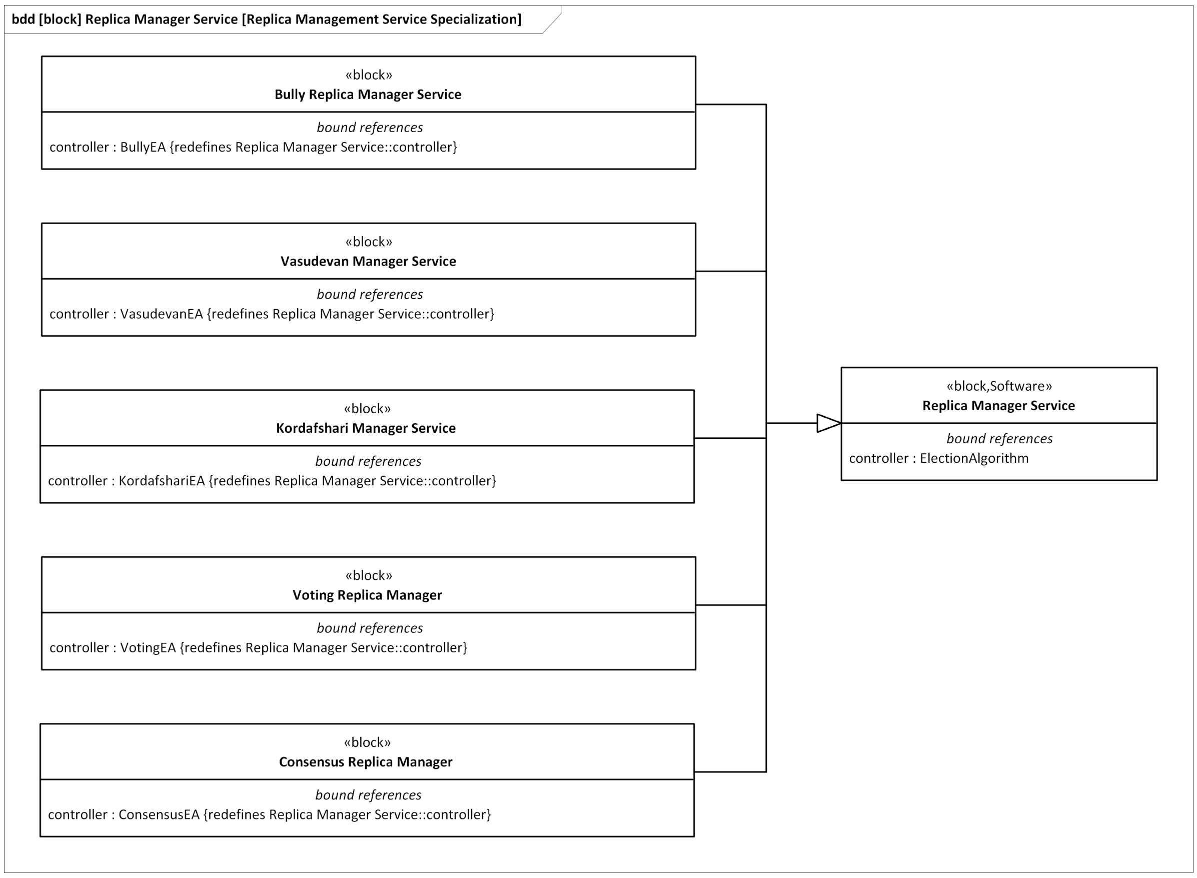

bdd [block] Replica Manager Service [Replica Management Service Specialization]
«block»
Bully Replica Manager Service
bound references
controller : BullyEA {redefines Replica Manager Service::controller}
«block»
Vasudevan Manager Service
bound references
controller : VasudevanEA {redefines Replica Manager Service::controller}
«block»
Kordafshari Manager Service
bound references
controller : KordafshariEA {redefines Replica Manager Service::controller}
«block»
Voting Replica Manager
bound references
controller : VotingEA {redefines Replica Manager Service::controller}
«block»
Consensus Replica Manager
bound references
controller : ConsensusEA {redefines Replica Manager Service::controller}
«block,Software»
Replica Manager Service
bound references
controller : ElectionAlgorithm

Fig. 5.16 *Block definition diagram showing the specialization of the Replica Manager Service block according to the election algorithm used, which redefines the controller part.*

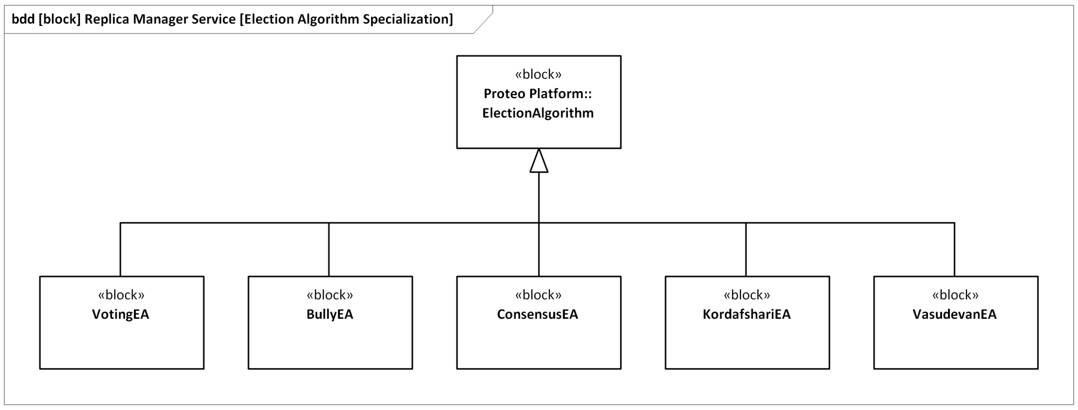


Fig. 5.17 *Block definition diagram showing the specialization of the ElectionAlgorithm block.*

support Proteo architecture, when providing or receiving a signal or when an operation is invoked (Section 5.5.1); (2) the state machine for each of the five variations modelled of the *Replica Manager Service*. These implement the behaviour of the *controller* part of the *Replica Manager Service* block (Section 5.5.2); and (3) the *updateScore(Real[]):Real* activity of the *Context Manager Service* block (Section 5.5.3).

5.5.1 Signals and Operations

In this section, the behaviour of the *Monitoring*, *Context Manager* and *Replica Manager* services (when receiving or providing a signal, or when one of their operations is invoked) is described. The declarations of these signals and operations are defined in the block diagram of Fig. 5.14 for each of the services.

Monitoring Services

- *Battery Monitoring Service*: this service requires the signal *batteryState(Real)*, which is provided periodically by the system on which *Middleware Service Platform* is deployed. The service has an established battery level threshold (*batteryThreshold*), which once reached will make to the *Battery Monitoring Service* to trigger the *batteryLow()* signal.

- *Network Monitoring service*: following the cross-layer approach presented in Section 4.4.1 the *Network Monitoring Service* requires one signal, *routingTableChanged()*, which is triggered by the system when the routing protocol modifies the routing table of the node. The *Network Monitoring Service* then processes the current routing table to find changes. These changes can be diverse and each of them triggers a different signal: (1) *connectedNode(Neighbour)* is triggered when a new node is added to the routing table; (2) *disconnectedNode(Neighbour)* is triggered when a node is deleted from the routing table; (3) *distanceChanged(Neighbour)* is triggered when the hop distance of an existing node in the routing table changes; and (4) *gatewayChanged(Neighbour)* is triggered when the gateway, i.e., the next node to reach a neighbour, changes. Each of this signals provides a *Neighbour* value type (Section 5.3.1) with the new information of the corresponding node. Finally, it should be noted that changes on the routing table can happen successively over short periods of time, depending on the routing protocol. For this reason, *the Network Monitoring Service* has a stabilization threshold (*stabilizationThreshold:long*) and the changes are only processed when the routing table has remained stable for that period. In this way, unnecessary adaptations are avoided.

Table 5.2 *Context Manager Service operations and descriptions.*

Operation	Parameters	Return Value	Related Attribute	Description
getBestNeighbour (GCM)	-	*Neighbour*	*neighbours:Neighbour[]*	Returns the neighbour node with higher score in the neighbour array
getDirectNeighboursNum (CM)	-	*int*	*neighbours:Neighbour[]*	Returns the number of nodes in the neighbour array with distance one hop, i.e., the number of direct connections of the node
getNeighbours (GCM)	-	*Neighbour[]*	*neighbours:Neighbour[]*	Returns the neighbours array
getNumNeighbours (GCM)	-	*int*	*neighbours:Neighbour[]*	Returns the number of total neighbour of the node, regardless the hop distance
getServer (GCM)	-	*IP*	*server:IP*	Return the IP address of the current server
updateScore (GCM)	*Real[]*	*Real*	*battery:Real, neighbours:Neighbour[] score:Real*	Calculates by means of a utility function how suitable is the node to host an active service replica. It takes as parameter an array of weights according to the computational features to evaluate (battery and network) and returns the resulting score, which is stored in the score attribute

GCM General Context Manager Service.
CM Context Manager Service

Context Manager Service

Context Manager Service also defines a set of operations to access to the stored context information through a request-response approach. These operations are specified in the block *Context Manager Service* defined in Fig. 5.14, under the *"operations"* stereotype, and described in Table 5.2.

The operation *updateScore(Real[]):Real* relies on the *Evaluation Function* block, which is a part of the *Context Manager Service* block. The *Evaluation Function* block takes the same parameters as *updateScore* operation, an array of weights. On this block, there is defined a constraint block (*Evaluation Constraint* in Fig. 5.14), which establishes that the sum of the array of weights has to be exactly one.

The *Evaluation Function* block defines the general structure of the utility function and its specialization defines the specific utility function. This allows the *Context Manager Service* to define different evaluation functions according to the priorities and objectives of the system. An example of this specialization can be found in the block definition diagram of Fig. 5.18. In this example, the utility function presented in Section 4.5 is implemented. Its mathematical processing is also shown in the parametric diagram of Fig. 5.19.

Finally, the *General Context Manager Service* provides two signals, which are inherited by the *Context Manager Service*:

- *neighbourhoodEmpty()*: it is triggered when the last reachable node of the neighbourhood becomes disconnected.
- *adapt()*: this signal is triggered when the *Context Manager Service* detects a change in the context that requires revising the current replication scheme in the network (e.g.,

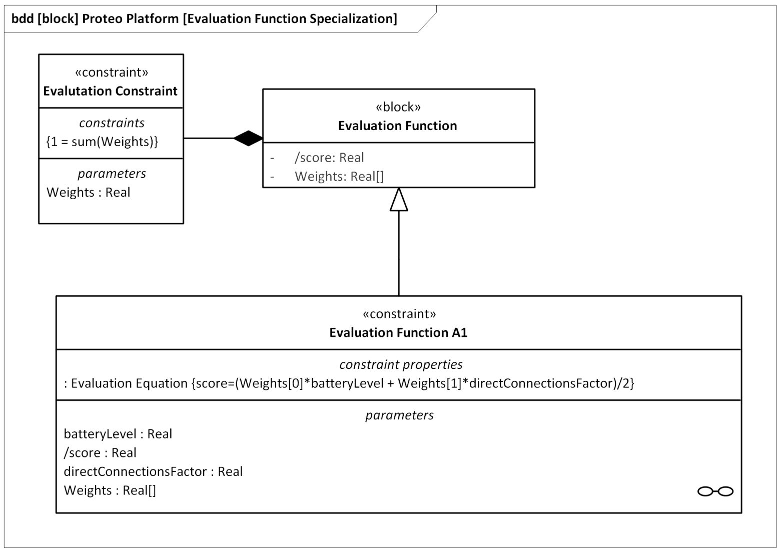


Fig. 5.18 *Block definition diagram showing an example of the specialization of the Evaluation Function.*

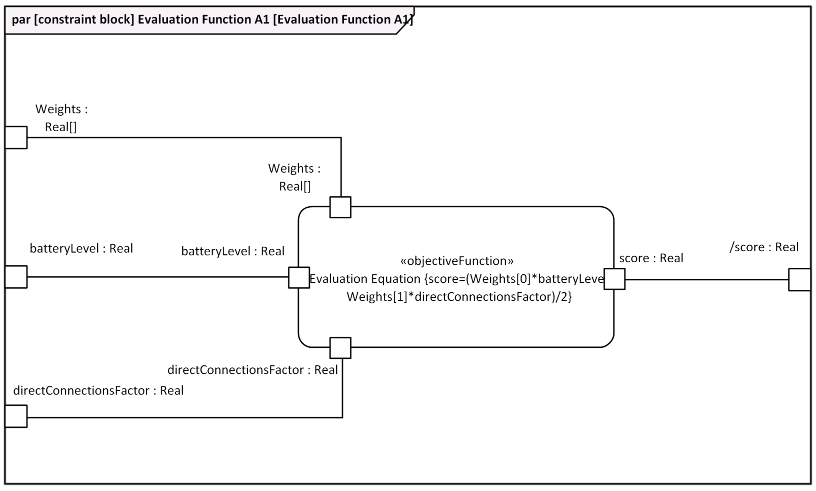

par [constraint block] Evaluation Function A1 [Evaluation Function A1]
Weights : Real[]
Weights : Real[]
batteryLevel : Real
batteryLevel : Real
«objectiveFunction»
Evaluation Equation {score=(Weights[0]*batteryLeve
Weights[1]*directConnectionsFactor)/2}
score : Real
/score : Real
directConnectionsFactor : Real
directConnectionsFactor : Real

Fig. 5.19 *Parametric diagram showing the mathematical processing to obtain the score of a node in the evaluation function specialization example.*

the node is running low of battery, or the current server becomes disconnected). This is defined by a set of rules that describes the trigger activation policies. The trigger policies implemented take as input the context events provided by the monitoring services or the coordination messages received from other nodes. The activation of any of these rules triggers the *adapt()* signal, which will derive in an election process executed by the *Replica Manager Service*. This event-driven approach allows the system to adapt to the particular requirements of a specific application domain, without modifying the *Replica Manager Service*.

Replica Manager Service

When a change in the context is detected that could affect the quality attributes of the service (i.e., the *adapt()* signal is received from *Context Manager Service*), the *controller* part will be responsible for coming to an agreement with the rest of the *Replica Manager Services* deployed in the system. This coordination is performed by means of the interchange of *Coordination Messages* through proxy ports *p1* and *p2* of *Replica Manager Service* block in Fig. 5.14.

5.5.2 Election Algorithm State Machines

In this section, state machine diagrams are used to model each of the election algorithms implemented in Proteo. These diagrams describe in detail the behaviour of the system when a host election is in process.

Bully Election Algorithm State Machine

Fig. 5.20 shows the state diagram that implements the *adapt()* behaviour in the *Bully Replica Manager Service*. This state machine describes the transitions between the four possible states of the node (*Local*, *Server*, *Client* and *Connecting*) according this election algorithm.

A node starts (*PowerOn* event) in the *Local* state. From this state the node goes into the *Connecting* state when the *adapt()* signal from the *Context Manager Service* is received. In the *Connecting* state, an interchange of *Coordination Messages* between the nodes of the *Network Partition* will take place to decide what node goes into the *Server* state and what to the *Client* state. From the *Connecting*, *Server* or *Client* state, the node returns to the *Local* state if the *neighbourhoodEmpty()* signal from *Context Manager Service* is received. Finally, a node goes into the *Connecting* state from *Server* or the *Client* state if it receives an *Election* message. This will mean that other node has started an election. Additionally, a node in the *Connecting* state can also receive an *Election* message, when multiple concurrent elections occurs.

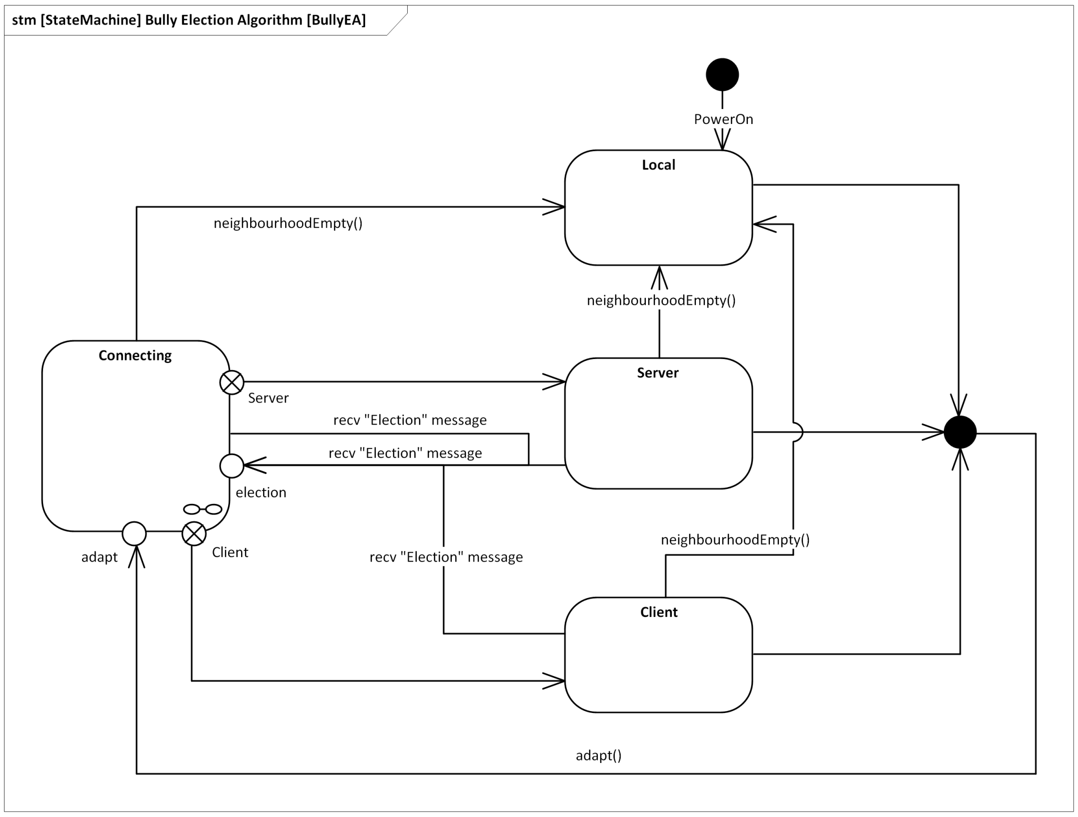


Fig. 5.20 *State machine diagram that shows the states of Bully election algorithm and the transitions between them.*

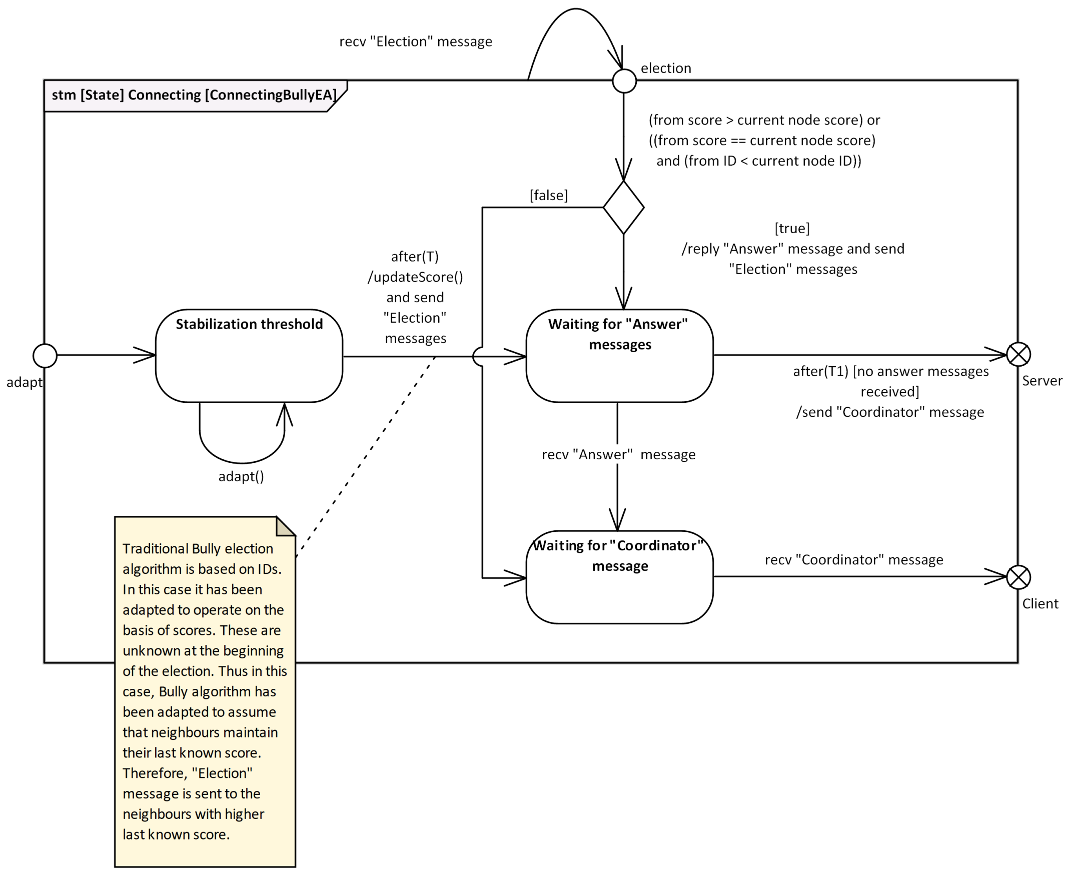

recv "Election" message
election
stm [State] Connecting [ConnectingBullyEA]
(from score > current node score) or
((from score == current node score)
and (from ID < current node ID))
[false]
[true]
/reply "Answer" message and send
"Election" messages
after(T)
/updateScore()
and send
"Election"
messages
Stabilization threshold
Waiting for "Answer"
messages
adapt
adapt()
after(T1) [no answer messages
received]
/send "Coordinator" message
Server
recv "Answer" message
Waiting for "Coordinator"
message
recv "Coordinator" message
Client
Traditional Bully election algorithm is based on IDs. In this case it has been adapted to operate on the basis of scores. These are unknown at the beginning of the election. Thus in this case, Bully algorithm has been adapted to assume that neighbours maintain their last known score. Therefore, "Election" message is sent to the neighbours with higher last known score.

Fig. 5.21 *State machine diagram that shows the behaviour of Bully election algorithm during the Connecting state.*

The node behaviour in the *Connecting* state is represented by the state machine of Fig. 5.21. The state machine has two entry points: (1) *adapt*, activated when an adapt message has been received, and (2) *election*, activated when an *Election* message has been received. It has two exit points: (1) *Server*, which will lead to the *Server* state in Fig. 5.20; and (2) *Client*, which leads to the *Client* state in Fig. 5.20. Additionally, according to the state machine Fig. 5.20, the *Connecting* state can be interrupted in any moment by *neighbourhoodEmpty()* signal from *Context Manager Service* or the reception of an *Election* message.

The first state in the *Connecting* state machine is *Stabilization threshold*. This state is used to wait for the stabilization of the network and avoid adaptation processes in short periods of time. After a timer *T*, the node will update its score, invoking the *updateScore()* operation from *Context Manager Service*, and will send an *Election* message to the nodes with last known score higher than its own current score. As noted in Section 4.6.2, this adaptation of the traditional Bully algorithm has been made to empower it to base the election on dynamic scores instead of static IDs.

After this, following the Bully algorithm, the node will wait for *Answer* messages. If after a timer *T1* no *Answer* messages are received, the node will declare itself as the winner of the election, sending the *Coordinator* message to the other nodes of the network and going into the *Server* state. Else, if an *Answer* message is received, the node will give up the election and will wait for the reception of the *Coordinator* message. When this message is received, a winner, i.e., a server, has been declared and thus the node will go into the *Client* state, establishing the winner as server (*server:IP* in *Context Manager Service* on Fig. 5.14).

A node can also arrive to the *Connecting* state receiving an *Election* message (Fig. 5.20). In this case, the entry point will be *election* on Fig. 5.21. The Bully algorithm does not adopt any measure to avoid concurrent elections. Therefore, the reception of an *Election* message will be treated in the same manner independently of the previous state of the node (*Server*, *Client* or *Connecting*). In the traditional version of the algorithm, the reception of an *Election* message was always replied, since IDs were assumed to be known and static. However, in the case of dynamic scores, the last known score stored by the sender node could vary from the actual score of the receiver node. For this reason, when a node receives an *Election* message, it has to update its score and check if its score is effectively higher than the score of the *Election* message sender.

In case of a tied score, the *ID* of the nodes will be used as tie-breaking, prevailing the node with higher *ID*. Hence, if the receiver node has higher *ID* than the sender node, in case of a tied score, the node will reply to the *Election* message with an *Answer* message to inform to the current candidate that there is a better candidate alive. Then the node will postulate itself as candidate sending an *Election* message to the nodes with last known score higher than its own current score and will wait for *Answer* messages, following the course of a normal adaptation. Else, if the *Election* message sender is a better node, the current node will give up the election and will wait for the reception of the *Coordinator* message, following the course of a normal adaptation.

Kordafshari Election Algorithm State Machine

Fig. 5.22 shows the state diagram that implements the *adapt()* behaviour in the *Kordafshari Replica Manager Service*. As mentioned in Section 2.4.2, Kordafshari algorithm is a modification of Bully algorithm devised to avoid concurrent elections and reduce the number of messages used to perform an election.

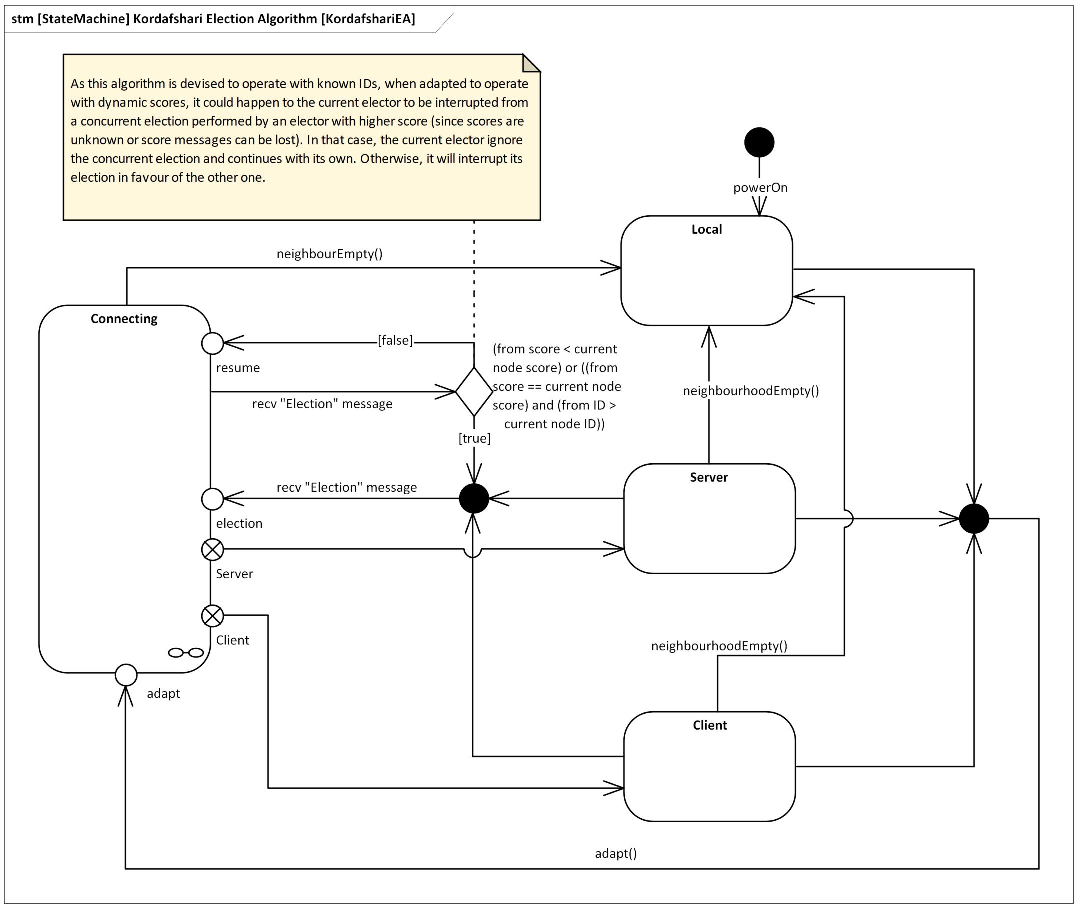

stm [StateMachine] Kordafshari Election Algorithm [KordafshariEA]
As this algorithm is devised to operate with known IDs, when adapted to operate with dynamic scores, it could happen to the current elector to be interrupted from a concurrent election performed by an elector with higher score (since scores are unknown or score messages can be lost). In that case, the current elector ignore the concurrent election and continues with its own. Otherwise, it will interrupt its election in favour of the other one.
powerOn
Local
neighbourEmpty()
Connecting
[false]
resume
recv "Election" message
(from score < current node score) or ((from score == current node score) and (from ID > current node ID))
[true]
neighbourhoodEmpty()
recv "Election" message
election
Server
Server
Client
neighbourhoodEmpty()
adapt
Client
adapt()

Fig. 5.22 *State machine diagram that shows the states of Kordafhari election algorithm and the transitions between them.*

As with Bully election algorithm state machine, a node starts (*PowerOn* event) in the *Local* state. From this state the node goes into the *Connecting* state when the *adapt()* signal from the

Context Manager Service is received. In the *Connecting* state, an interchange of *Coordination Messages* between the nodes of the *Network Partition* will take place to decide what the most suitable node to act as *Server* is. From the *Connecting*, *Server* or *Client* state the node returns to the *Local* state if the *neighbourhoodEmpty()* signal from *Context Manager Service* is received.

A node goes into the *Connecting* state from *Server* or *Client* state if it receives an *Election* message. This will mean that another node has started an election. However, on the contrary to Bully, if the node receives an *Election* message when it is on the *Connecting* state (i.e., there are concurrent elections), the node will interrupt its own election in favour of the other one if this is a better candidate, or resume its own election process and ignore the new one if itself is a better candidate than the other one.

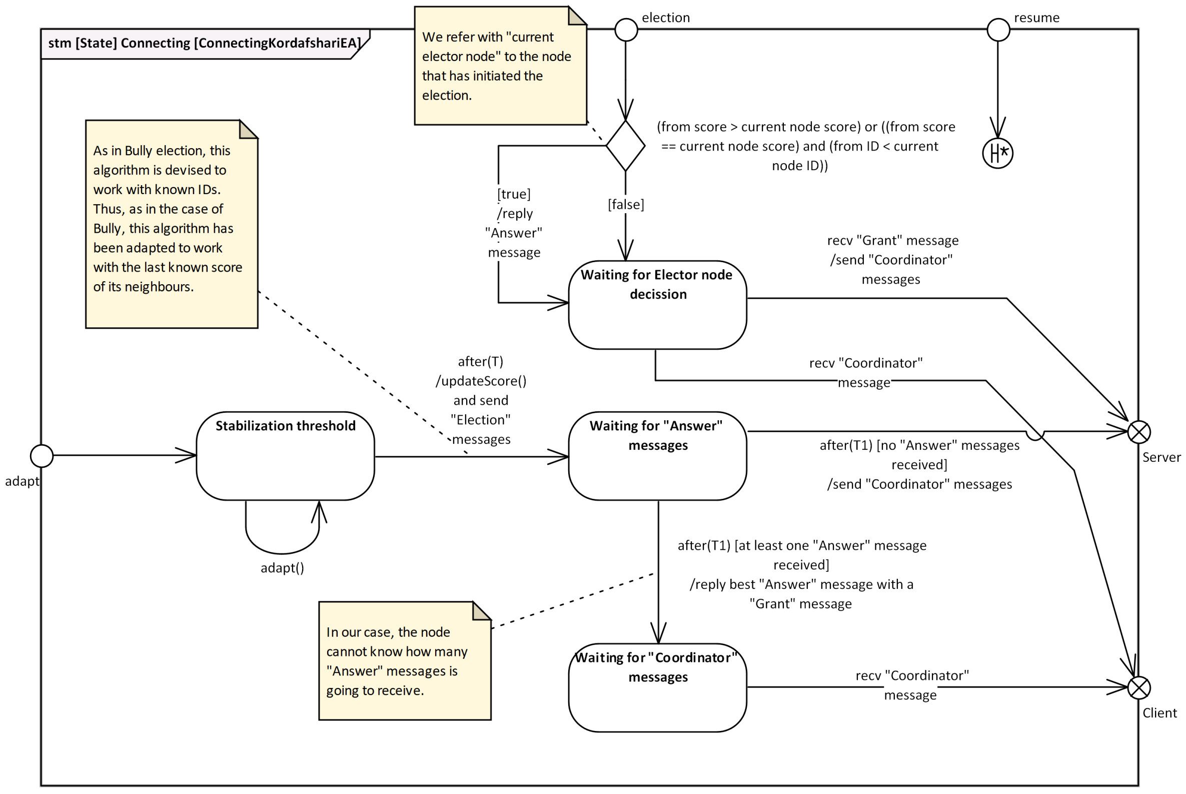

stm [State] Connecting [ConnectingKordafshariEA]
We refer with "current elector node" to the node that has initiated the election.
election
resume
As in Bully election, this algorithm is devised to work with known IDs. Thus, as in the case of Bully, this algorithm has been adapted to work with the last known score of its neighbours.
(from score > current node score) or ((from score == current node score) and (from ID < current node ID))
H*
[true] /reply "Answer" message
[false]
Waiting for Elector node decission
recv "Grant" message /send "Coordinator" messages
recv "Coordinator" message
after(T) /updateScore() and send "Election" messages
Stabilization threshold
Waiting for "Answer" messages
after(T1) [no "Answer" messages received] /send "Coordinator" messages
Server
adapt
adapt()
after(T1) [at least one "Answer" message received] /reply best "Answer" message with a "Grant" message
In our case, the node cannot know how many "Answer" messages is going to receive.
Waiting for "Coordinator" messages
recv "Coordinator" message
Client

Fig. 5.23 *State machine diagram that shows the behaviour of Kordafhari election algorithm during the Connecting state.*

The state machine of Fig. 5.23 represents the node behaviour in *Connecting*. The state machine has three entry points: (1) *adapt*, activated when an adapt message has been received; (2) *election*, activated when an Election message has been received; and (3) *resume*, when an *Election* message has been received in the *Connecting* state and the sender is not a better candidate than the current node. In this case, the state machine resumes its connecting process employing a **deep history pseudostate**[2].

It has two exit points: (1) *Server*, which will lead to the *Server* state in Fig. 5.22; and (2) *Client*, which leads to the *Client* state in Fig. 5.22. Additionally, according to the state machine in Fig. 5.22, the *Connecting* state can be interrupted in any moment by *neighbourhoodEmpty()* signal from *Context Manager Service* or the reception of an *Election* message.

The first state in the *Connecting* state machine is *Stabilization threshold*. As on the other algorithms, this state is used to wait for the stabilization of the network and avoid adaptation processes in short periods of time. After a timer *T*, the node will update its score, invoking the *updateScore()* operation from *Context Manager Service*, and will send an *Election* message to the nodes with the last known score higher than its own current score. As in the case of the Bully algorithm, this adaptation has been made to allow the Kordafshari election algorithm to base the election on dynamic scores instead of static IDs.

After this, the node will wait for *Answer* messages. If after a timer *T1* no *Answer* messages are received, the node will declare itself as the winner of the election, sending the *Coordinator* message to the other nodes of the network and going into the *Server* state.

Until this point, the Kordafshari algorithm is similar to Bully algorithm. However, when the *Answer* messages are received, instead of wait for the *Coordinator* message, after a timer *T1*, the current node will send a *Grant* message to the best-ranked node from those that have replied an *Answer* message. Then the node will wait for a *Coordinator* message. When it is received the node will go into the *Client* state, establishing the winner as server (*server:IP* in *Context Manager Service* in Fig. 5.14).

A node can arrive to the *Connecting* state receiving an *Election* message as well (Fig. 5.22). In this case, the entry point will be *election* in Fig. 5.23. The node will reach this point from the *Server* or *Client* state, or from the *Connecting* state if the node that has initiated the concurrent election is a node with a lower score. This is, in the Kordafhari algorithm, if there are concurrent elections, the election initiated by the node with the lowest score will prevail. As in the case of the adaptation performed on the Bully algorithm, the last known score stored by the sender node could vary from the actual score of the receiver node. For this reason, when a node receives an *Election* message, it has to update its score and check if its score is certainly higher than the score of the *Election* message sender. If the current node is the best node, it will reply the election message with an *Answer* message, and then, on the contrary that in Bully algorithm, it will wait for a *Grant* message or a *Coordinator* message. If the current node is not

[2]Deep history pseudostate refers to the last active state configuration when the composite state was exited. Therefore, this pseudostate, when active, restarts the composite state that contains it to its last configuration.

a better node than the *Election* message sender, it will only wait for a *Coordinator* message without replying any message.

In this point, if the node receives a *Grant* message, it means that it has won the election. Thus it will broadcast a *Coordinator* message and will go into the *Server* state. Else, if the node receives a *Coordinator* message, it means that another node has won the election. Hence, the node will go into the *Client* state, stablishing the winner as server (*server:IP* in *Context Manager Service* in Fig. 5.14).

Vasudevan Election Algorithm State Machine

Fig. 5.24 shows the state diagram that implements the *adapt()* behaviour in the *Vasudevan Replica Manager Service*.

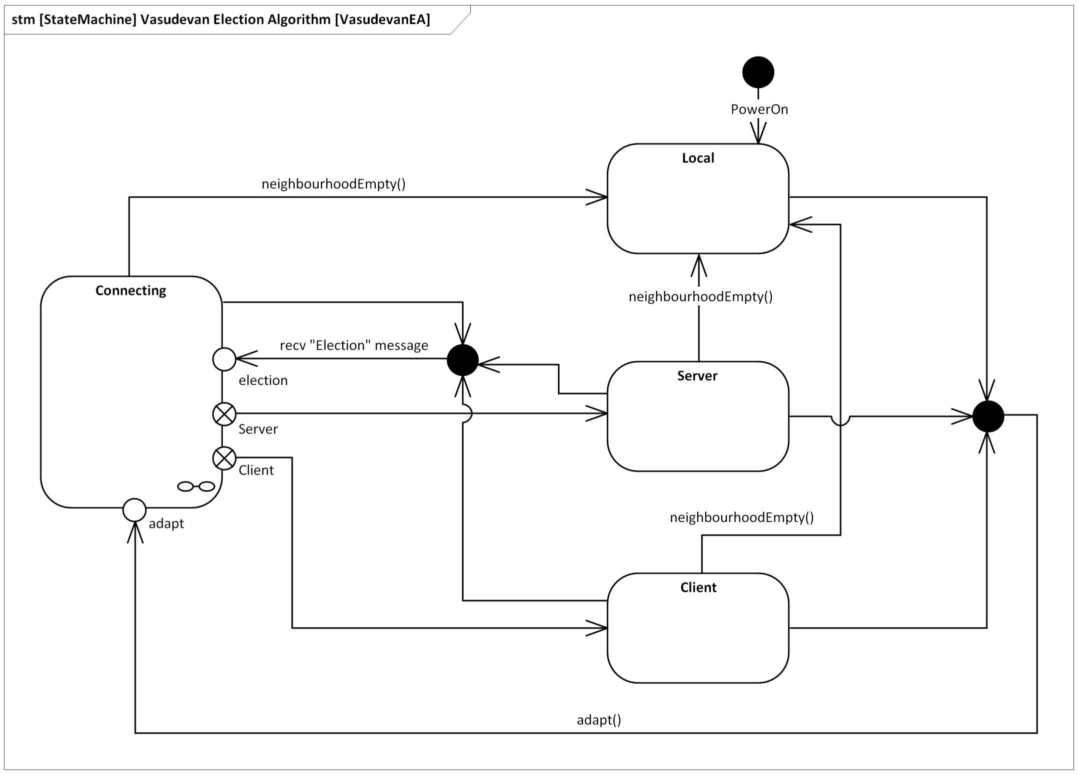


Fig. 5.24 *State machine diagram that shows the states of Vasudevan election algorithm and the transitions between them.*

A node starts (*PowerOn* event) in the *Local* state. From this state the node goes into the *Connecting* state when the *adapt()* signal from the *Context Manager Service* is received. In the *Connecting* state, an interchange of *Coordination Messages* between the nodes of the *Network Partition* will take place to decide what the most suitable node to act as *Server* is. From the *Connecting*, *Server* or *Client* states, the node returns to the *Local* state if the *neighbourhoodEmpty()* signal from *Context Manager Service* is received.

A node goes into the *Connecting* state from the *Server* or *Client* states if it receives an *Election* message. This message is used to grown the spanning tree. It will mean that another node has started an election. When there are concurrent elections, this is, when a node in the *Connecting* state receives an *Election* message, the node will support the newest election, interrupting the current one. This is managed within the *Connecting* state.

The node behaviour in the *Connecting* state is represented by the state machine of Fig. 5.25. The state machine has two entry points: (1) *adapt*, activated when an *adapt()* signal has been received; and (2) *election*, activated when an *Election* message has been received. As in the other algorithms, it has two exit points: (1) *Server*, which will lead to the *Server* state in Fig. 5.24; and (2) *Client*, which leads to the *Client* state in Fig. 5.24. Additionally, according to the state machine of Fig. 5.24, the *Connecting* state can be interrupted in any moment by *neighbourhoodEmpty()* signal from *Context Manager Service* or the reception of an *Election* message.

As on the other algorithms, the first state in the *Connecting* state machine is *Stabilization threshold*. After a timer *T* the node will send an *Election* message to its direct neighbours (i.e., nodes at one hop distance), to start growing the spanning tree. Unlike in the previous algorithms (Bully and Kordafhari) this message does not contain the score of the sender node, thus, to invoke the *updateScore()* operation is not still necessary. The node that has started the election marks itself as the root of the spanning tree. After this, the node will wait for the *ACK* messages of their children.

A node that is not a root node, i.e., that has not started the election, will entry to the *Connecting* state by the election entry point in Fig. 5.25, after receiving an *Election* message from its parent node. This node will check if the *Election* message received corresponds to the current election, if the node is already in an election process. If so, the node will check if it is the first *Election* message received in this election, in other words, if a new election is starting. If it is the case, the node will mark the *Election* message sender node as its parent. Then, it will send (i.e., propagate) an *Election* message to its direct neighbours, except its parent node. After this, the node will wait for the *ACK* messages of its child.

When all *ACK* expected are received (i.e, a leaf node does not send any *Election* message, thus it does not expect to receive any *ACK*) the node checks if it is the root node of the tree. If not, the node updates its score, invoking the *updateScore()* operation from *Context Manager Service*, checks the scores received in the *ACK* messages received, and sent an *ACK* to its parent with the *score* and *ID* of the best node of its subtree, including itself. If the node is the

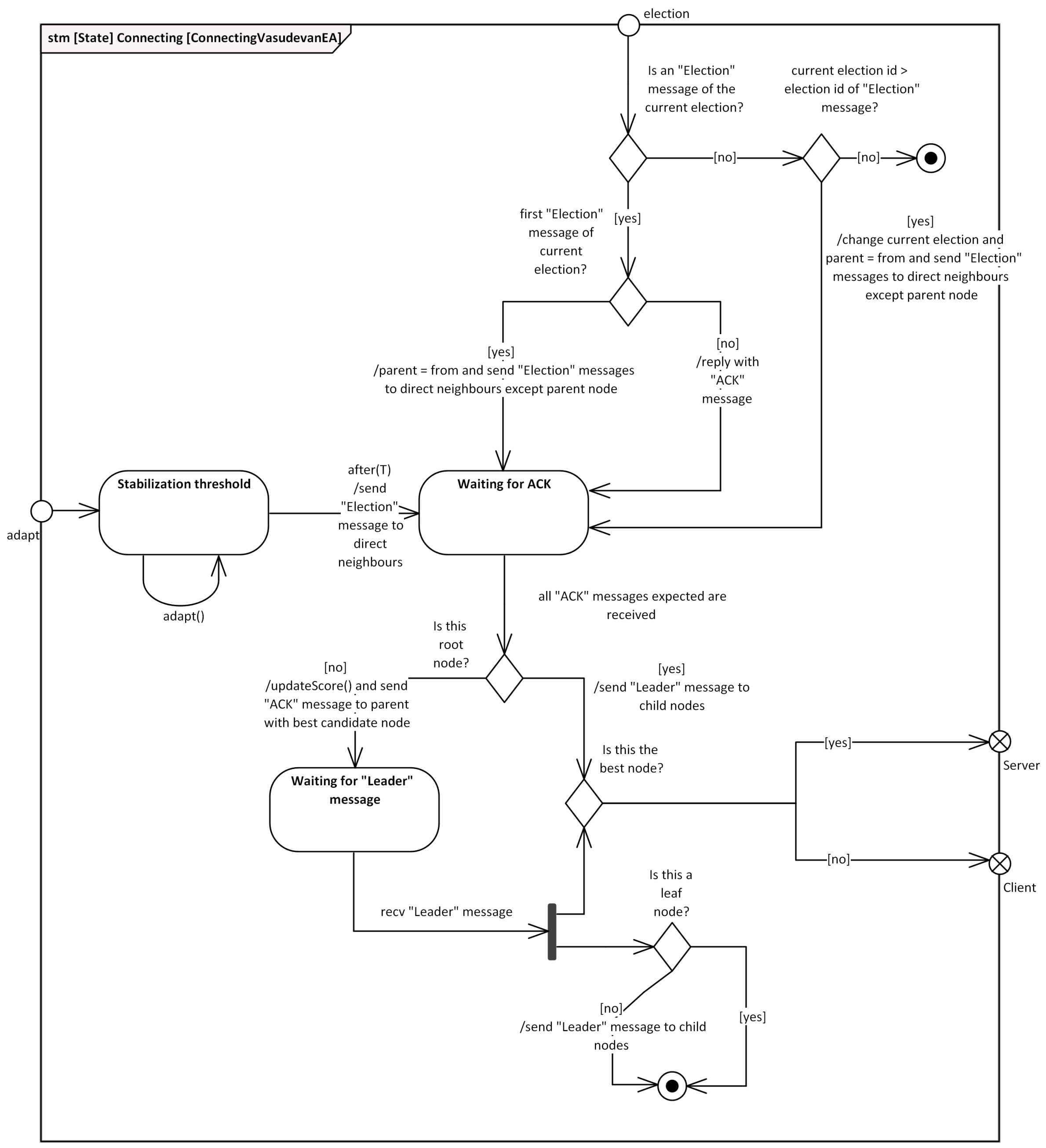

stm [State] Connecting [ConnectingVasudevanEA]
election
Is an "Election" message of the current election?
current election id > election id of "Election" message?
[no]
[no]
[yes]
[yes] /change current election and parent = from and send "Election" messages to direct neighbours except parent node
first "Election" message of current election?
[yes] /parent = from and send "Election" messages to direct neighbours except parent node
[no] /reply with "ACK" message
adapt
Stabilization threshold
adapt()
after(T) /send "Election" message to direct neighbours
Waiting for ACK
all "ACK" messages expected are received
Is this root node?
[no] /updateScore() and send "ACK" message to parent with best candidate node
[yes] /send "Leader" message to child nodes
Waiting for "Leader" message
Is this the best node?
[yes]
Server
[no]
Client
recv "Leader" message
Is this a leaf node?
[no] /send "Leader" message to child nodes
[yes]

Fig. 5.25 *State machine diagram that shows the behaviour of Vasudevan election algorithm during the Connecting state.*

root node, it sends a *Leader* message, announcing the winner of the election, to its child nodes. If the winner of the election is itself, the node goes into the *Server* state, if not, into the *Client* state.

In the case of the other nodes (i.e., non-root nodes), after sending the *ACK* message to its parent, they wait for the *Leader* message. After receiving this message, announcing the winner, the behaviour of the node continues into two parallel flows. On the one hand, if the winner of the election is itself, the node goes into the *Server* state, if not, into the *Client* state, setting the winner as the server. On the other hand, if the node is not a leaf node, it will propagate the *Leader* message to its child, to get the message across all nodes of the tree.

Consensus Election Algorithm State Machine

Fig.5.26 shows the state diagram that implements the *adapt()* behaviour in the *Consensus Replica Manager Service*. The node starts (*PowerOn* event) in the *Local* state. From this state the node goes into the *Connecting* state when the *adapt()* signal from the *Context Manager Service* is received. In the *Connecting* state, the coordination with other nodes of the network partition takes place, and the node will go into the *Server* or *Client* state according to the agreement reached. From the *Connecting*, *Server* or *Client* state the node returns to the *Local* state if the *neighbourhoodEmpty()* signal from *Context Manager Service* is received.

On the contrary to previous algorithms, the *Connecting* state only has an entry point, *adapt*. Thus, each node on the *Network Partition* receives the *adapt()* signal, regardless of what node starts the election. Note that, in Consensus algorithm, the *adapt()* signal could be also triggered by the reception of a *Score* message in the *Context Manager Service*.

Consensus algorithm, as informally introduced in Section 4.6.1 provides a mechanism to reduce the number of concurrent servers in network partitions in case of message loss. Thus, if a node receives a *Server Request* message when it is in the *Server* state, it will answer with a *Server Acceptance* message. However, if the node that receives this message is in the *Client* state, it will mean that the requester node has not the correct information about the best-ranked node in the network partition, but the receiver node has this information since it is in the *Client* state. Therefore, the receiver node will reply with a *Server Rejection* message, which contains the information about the current server of the network partition, and thereby completing the information of the requester node.

A node can also receive the *Server Request* message when it is in the *Connecting* state. This is due to the mismatch between the processing speed of the different nodes in the distributed system. In this way, the requester node has processed the coordination messages and has resolved who is the winner of the election, whereas the requested node has not yet done so. In

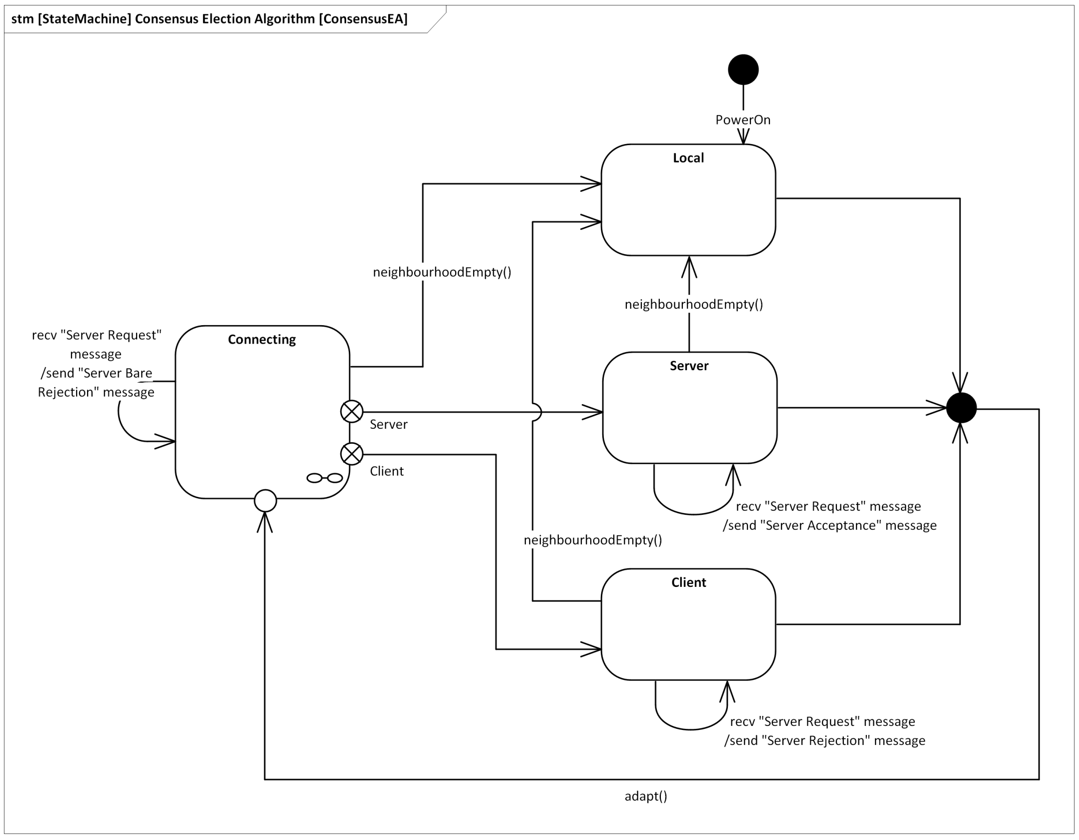

stm [StateMachine] Consensus Election Algorithm [ConsensusEA]
PowerOn
Local
neighbourhoodEmpty()
neighbourhoodEmpty()
recv "Server Request" message /send "Server Bare Rejection" message
Connecting
Server
Server
Client
recv "Server Request" message /send "Server Acceptance" message
neighbourhoodEmpty()
Client
recv "Server Request" message /send "Server Rejection" message
adapt()

Fig. 5.26 *State machine diagram that shows the states of Consensus election algorithm and the transitions between them.*

this case, the requested node will reply with a *Server Bare Rejection* message, which simply rejects the request.

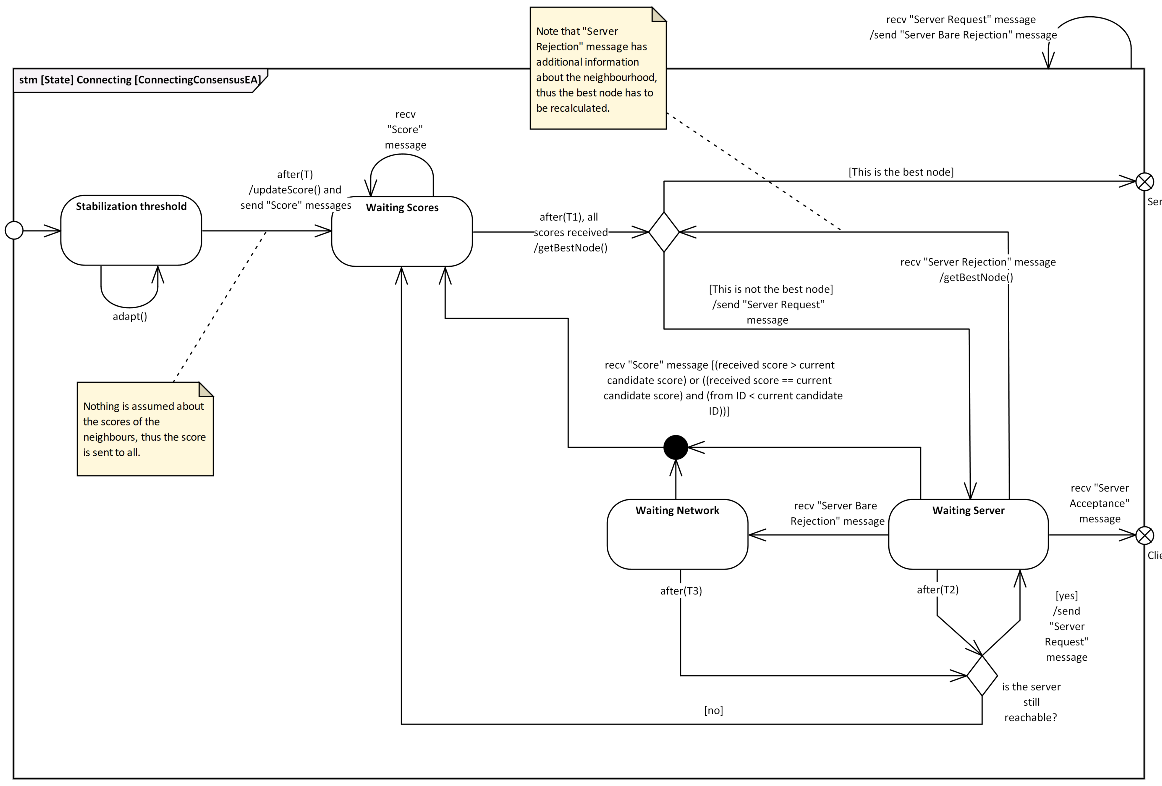

stm [State] Connecting [ConnectingConsensusEA]
Note that "Server Rejection" message has additional information about the neighbourhood, thus the best node has to be recalculated.
recv "Server Request" message /send "Server Bare Rejection" message
recv "Score" message
after(T) /updateScore() and send "Score" messages
Stabilization threshold
Waiting Scores
adapt()
after(T1), all scores received /getBestNode()
[This is the best node]
Server
recv "Server Rejection" message /getBestNode()
[This is not the best node] /send "Server Request" message
Nothing is assumed about the scores of the neighbours, thus the score is sent to all.
recv "Score" message [(received score > current candidate score) or ((received score == current candidate score) and (from ID < current candidate ID))]
Waiting Network
recv "Server Bare Rejection" message
Waiting Server
recv "Server Acceptance" message
Client
after(T3)
after(T2)
[yes] /send "Server Request" message
is the server still reachable?
[no]

Fig. 5.27 *State machine diagram that shows the behaviour of Consensus election algorithm during the Connecting state.*

The node behaviour in the *Connecting* state is represented by the state machine of Fig. 5.27. The state machine has only one entry point: *adapt*, activated when an adapt message has been received. It has two exit points: (1) *Server*, which will lead to the *Server* state in Fig.5.26; and (2) *Client*, which leads to the *Client* state in Fig.5.26. Additionally, according to the state machine Fig.5.26, the *Connecting* state can be interrupted in any moment by *neighbourhoodEmpty()* signal from the *Context Manager Service*. Note that the reception of a *Server Request* message in the *Connecting* state does not interrupt the state machine.

The first state in the *Connecting* state machine is *Stabilization threshold*, to avoid adaptation processes in short periods of time. After a timer *T* the node will update its score, invoking the *updateScore()* operation from *Context Manager Service*, and will send a *Score* message to the nodes of its *Network Partition*. After this, the node will wait for the *Score* messages of the other nodes of the partition. After that a timer *T1* expires or all *Score* messages are received, the node will invoke the *getBestNode()* operation from *Context Manager Service*, in order to know who is the best ranked node available in its *Network Partition*. Note that, if a node does not send its *Score* message it cannot be eligible, thus, *Score* messages are used to know what nodes are alive and can be elected as servers.

If the best node is itself, it goes directly into the *Server* state and waits for *Server Request* messages, which will be replied affirmatively with *Server Acceptance* messages (Fig.5.26). Else, if the best ranked node is other node of the partition, the node will send to this node (candidate server) a *Server Request* message and will wait for its reply in the (*Waiting Server* state in Fig. 5.27).

At this point one of next five situations may occur:

- The node receives a *Server Acceptance* message. Thus it establishes the requested node as the server and goes into *Client* status.

- The node receives a *Server Rejection* message. This happens when the request was made to a wrong node, which is in *Client* status. In this case, the received message has additional information about the current server of the *Network Partition*. Therefore, the node invokes the *getBestNode()* operation again and sends the request message, this time, to the correct node.

- The node receives a *Server Bare Rejection* message. The request was made to a node that is in the *Connecting* state. In this case, the node waits for the other nodes to process the election information since it has no new information about the network. If after a timer *T3* the candidate node is still reachable, the node will send the *Server Request* message again. If not, the node waits for new scores messages and, if there are, it recalculates the best node (i.e., it returns to the *Waiting Scores* state in Fig. 5.27).

- After a timer *T2* the node has not received any reply. In this case, if the candidate node is still reachable, the node resends the request message. If not, the node returns to the *Waiting Scores* state to elect another candidate server.

- A new *Score* message is received. In this case, if the sender of the message is a better node than the current candidate server, the node returns to the *Waiting Scores* state to elect this new candidate or another that could arise. If not, the node simply ignores the message and waits for a reply to its *Server Request* message.

Voting Election Algorithm State Machine

Fig. 5.28 shows the state diagram that implements the *adapt()* behaviour in the *Voting Replica Manager Service*. As introduced in Section 4.6.1, the Voting algorithm arises as a more reliable and simpler alternative to Consensus algorithm.

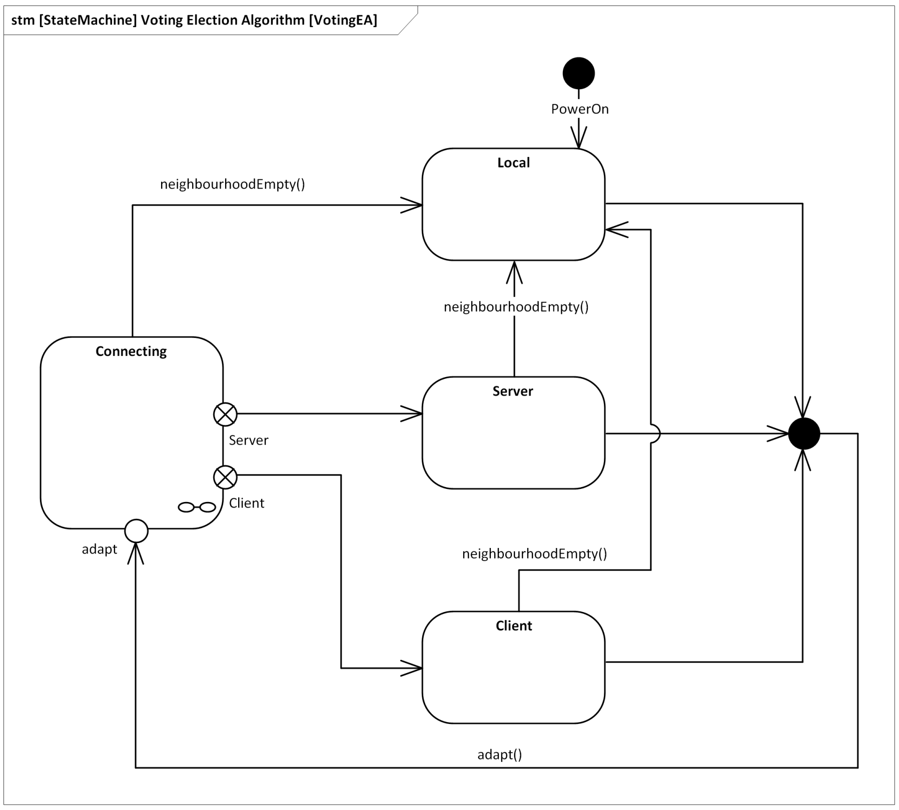


Fig. 5.28 *State machine diagram that shows the states of Voting election algorithm and the transitions between them.*

The node starts (*PowerOn* event) in the *Local* state. From this state the node goes into the *Connecting* state when the *adapt()* signal from the *Context Manager Service* is received. From the *Connecting*, *Server* or *Client* state the node returns to the *Local* state if the *neighbourhood-Empty()* signal from *Context Manager Service* is received. As in Consensus algorithm, the *Connecting* state only has an entry point, *adapt*. Thus, each node on the *Network Partition* receives the *adapt()* signal, regardless of what node starts the election.

The node behaviour in the *Connecting* state is represented by the state machine of Fig. 5.29. The state machine has only one entry point: *adapt*, activated when an *adapt()* signal has been received. It has two exit points: (1) *Server*, which will lead to the *Server* state in Fig. 5.28; and (2) *Client*, which leads to the *Client* state in Fig. 5.28. Additionally, according

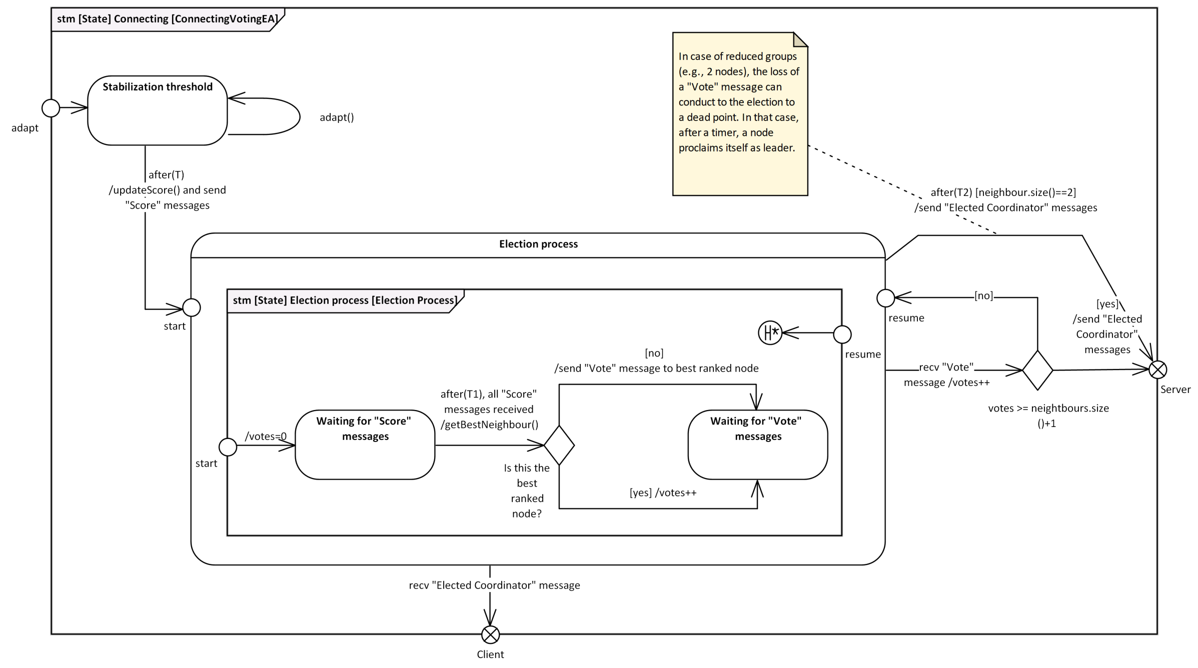

stm [State] Connecting [ConnectingVotingEA]
adapt
Stabilization threshold
adapt()
after(T)
/updateScore() and send
"Score" messages
In case of reduced groups (e.g., 2 nodes), the loss of a "Vote" message can conduct to the election to a dead point. In that case, after a timer, a node proclaims itself as leader.
after(T2) [neighbour.size()==2]
/send "Elected Coordinator" messages
Election process
stm [State] Election process [Election Process]
start
start
resume
resume
[no]
[yes]
/send "Elected Coordinator" messages
Server
recv "Vote" message /votes++
votes >= neightbours.size ()+1
[no]
/send "Vote" message to best ranked node
after(T1), all "Score" messages received
/getBestNeighbour()
/votes=0
Waiting for "Score" messages
Waiting for "Vote" messages
Is this the best ranked node?
[yes] /votes++
recv "Elected Coordinator" message
Client

Fig. 5.29 *State machine diagram that shows the behaviour of Voting election algorithm during the Connecting state.*

to the state machine Fig. 5.28, the *Connecting* state can be interrupted in any moment by *neighbourhoodEmpty()* signal from *Context Manager Service*.

As in previous algorithms, to avoid adaptation processes in short periods of time, the first state in the *Connecting* state machine is *Stabilization threshold*. After a timer *T* the node will update its score, invoking the *updateScore()* operation from *Context Manager Service*, and will send a *Score* message to the nodes of its *Network Partition*.

After this, the node enters in the *Election* process state (Fig. 5.29). The first step is to wait for the *Score* messages of the other nodes of the partition and set the vote counter to 0. After a timer *T1* expires or all *Score* messages are received, the node will invoke the *getBestNode()* operation from *Context Manager Service*, to obtain the best ranked node as candidate to win the election. If this node is itself, it will increment its vote counter (this action does not require a *Vote* message). Else, it will send a *Vote* message to the candidate node. Then it will wait for *Vote* messages or for the winner to announce its election using an *Elected Coordinator* message in the *Waiting for Vote messages* state.

The election process is interrupted when:

- An *Elected Coordinator* message is received. In this case, another node of the partition has won the election. Thus the node set the winner as the server and goes into the *Client* state.

- A *Voting* message is received. The node increments its vote counter and checks if it has the half plus one of the votes. If so, it has won the election. Thus, it will announce this through an *Elected Coordinator* message to its neighbours and it will go into the *Server* state. Else, if it has not still the half plus one of the votes, it will continue waiting for more *Voting* messages or for another node to win the election.

- The timer *T2* expires and the network size is equal to two nodes. In case of reduced groups, the loss of a *Vote* message can conduct the election to a dead point. In this case, after the timer expires, the node proclaims itself as the winner. Although this could not conduct to the optimal solution, it leads to a local optimum instead to a dead point in the election.

5.5.3 updateScore Activity

The last element of the Behavioural Package is the activity that describes the behaviour of the *updateScore(Real[]):Real* operation. This activity diagram (Fig. 5.30) describes the collaboration of the *Context Manager Service* block with its part, *Evaluation Function*, to obtain the score of a node (Fig. 5.19).

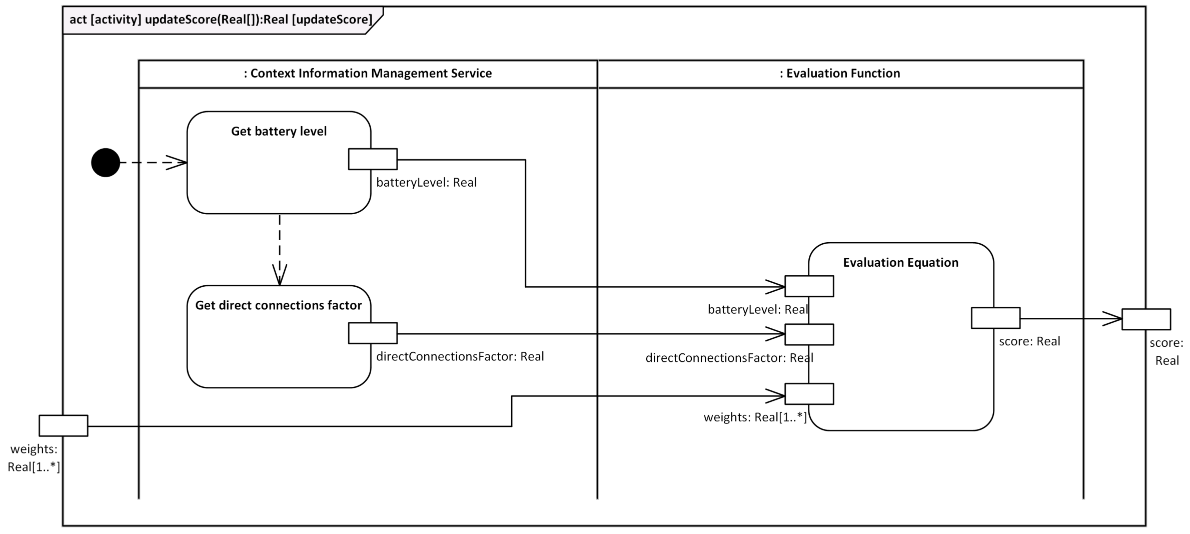

act [activity] updateScore(Real[]):Real [updateScore]
: Context Information Management Service
: Evaluation Function
Get battery level
batteryLevel: Real
Get direct connections factor
directConnectionsFactor: Real
Evaluation Equation
batteryLevel: Real
directConnectionsFactor: Real
weights: Real[1..*]
score: Real
score: Real
weights: Real[1..*]

Fig. 5.30 *Activity diagram showing the collaboration between the Context Manager Service and Evaluation Function components when the operation updateScore is invoked.*

The *updateScore(Real[]):Real* operation takes as parameter an array of reals, which represents the weights assigned to each of the computational features of the node to be evaluated (*weights:Real[1..*]* in Fig. 5.30). The *Context Manager Service* first obtains the current level of battery of the node (*batteryLevel:Real*) and then the direct connections factors. This is calculated as the number of direct connections of the node divided by the total number of nodes in the network partition. These parameters are provided to the *Evaluation Equation* objective function (Fig. 5.30), which returns the score of the node (*score:Real* in Fig. 5.19) as a result of the *updateScore(Real[]):Real* operation.

5.6 Summary

Throughout this chapter, the *Operational Domain Model*, *Information Model*, *Structural Model*, and *Behavioural Model* of Proteo Model have been detailed using SysML. These parts together define in detail the components of Proteo architecture, their relations and their behaviour during system operation.

Chapter 6

Simulation: Development and Settings

Chapter Abstract

In order to validate the proposed architecture, and specifically the behaviour of the election algorithms, the ns-3 network simulator has been used. In the first part of this chapter, an overview of the design and development of Proteo is introduced. Proteo has been designed and developed as a new module in ns-3. This module provides a new ns-3 application that can be installed in the nodes of the simulator. In ns3, applications represent the software entities located at the Application layer in the TCP/IP model. Additionally, the specific configuration and features of the nodes in the simulation are presented. These will be mobile nodes, based on a lithium-ion battery and with an IEEE 802.11b wireless connection.

In the second part, the configuration of the simulations under which the architecture is going to be evaluated is exposed. This information is important for the correct evaluation and validation of our proposal since it allows us to explicitly known the circumstances under which the evaluation results will be obtained. These results will be shown in the next chapter.

Chapter Contents

6.1 Introduction

Within the pragmatic evaluation of software architectures, the *Solution Adequacy Check* (SAC) is defined as *"to check whether the architecture solutions at hand are adequate for the architecture drivers identified and whether there is enough confidence in the adequacy"* [88]. This analysis of system adequacy is traditionally addressed from two approaches in predicted evaluation[1]: *analytical approach* and *simulation approach*. Whereas *analytical approach* describes a system mathematically, often making simplifying assumptions, *simulation approach* requires less abstraction in the model. Usually, this last approach is preferable in large and complex systems. Additionally, *simulation approach* allows us to represent the dynamic nature of computer networks, and therefore to handle and evaluate dynamic models [78]. Hence, simulation is considered the highest step in predicted evaluation, and the last before to the architecture prototyping. Although simulation is useful to evaluate complex systems and complex situations, the experience requirement is high to associate the simulation model with the real behaviour of the system.

In the process of simulation evaluation, three main tasks have to be performed [146]: (1) to define the problem, the model and the set of experiments that should be carried out. This usually represents the 40% of the effort; (2) to implement the model, which represents the 20% of the effort; and (3) to verify and validate the simulation model, the inputs and outputs correspondence, and to analyse the results obtained, which represents the remaining 40%. This chapter attempts to provide an overview of the two first tasks. The third one, analysis of the results obtained, will be addressed in Chapter 7.

The rest of this chapter is organized as follows. Fist, Section 6.2 provides an overview of the architecture of ns-3 and how it has been extended in this thesis to make possible the evaluation of the proposed architecture. Second, in Section 6.3, the setting of the simulations under which the architecture is going to be evaluated is exposed.

6.2 Design and Development

The objective of this section is to provide an overview of how the proposed architecture has been implemented on ns-3. First, a brief overview of the high-level architecture of the basic ns-3 Node is provided in Section 6.2.1. Then, Section 6.2.2 describes how ns-3 have been extended with a new module, called *Proteo Module*, which implements the proposed architecture. Section

[1] *"Prediction means using existing knowledge, current information, and historical data to reason about the future. Confidence is achieved by using the available resources to predict properties of the software system."*[87]

6.2.3 shows how the basic Node of ns-3 is extended and configured to host the *Proteo Module*. Finally, Section 6.2.4 describes the modifications that have been necessary to perform on ns-3, to allow the evaluation of the architecture and overcoming some of the limitations of the simulator.

6.2.1 ns-3 Node High-Level Architecture

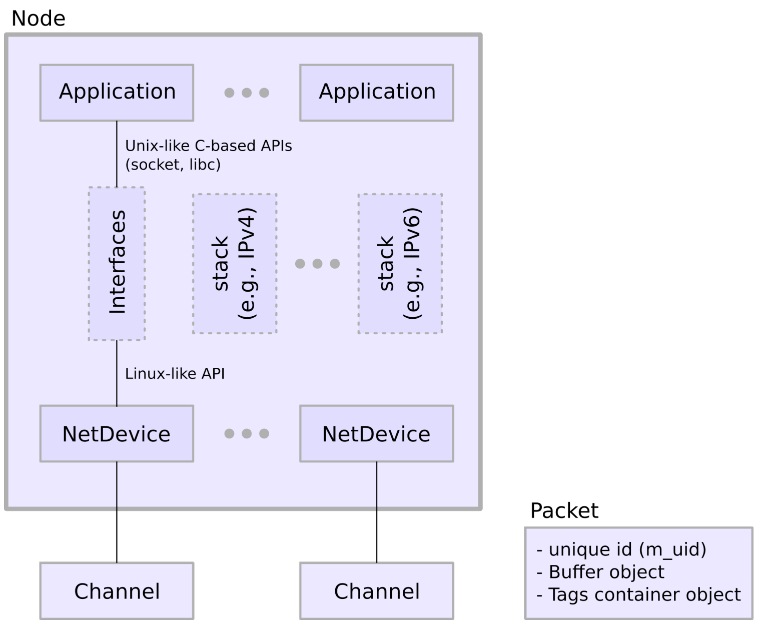


Fig. 6.1 *High-level ns-3 node architecture. Extracted from [112].*

In the ns-3 software architecture, the class node is the main base class [112]. In Fig. 6.1 the high-level ns-3 node architecture is shown. As it can be seen a node can have different *Applications* installed, as well as different *NetDevices* that can operate in different communication *Channels*. The class *NetDevice* represents a physical interface on a node (e.g., an Ethernet interface). The first level of the interface of *NetDevice* is close to the point in Linux kernel where *dev_queue_xmit()* is called, and the data members are similar to the found in Linux *struct net_device*. Detailed information about the implementation of *NetDevice* can be found on [112].

The class *Channel* represents the logical path over the information flows, and the class *Application* represents user-defined processes, which generate network traffic to send across the simulated networks.

Finally, the class *Socket*, which connects application layer with the transport layer, in ns-3 tries to imitate the standard Berkeley Software Distribution (BSD) sockets API [38]. The main difference is that BSD socket calls are synchronous, but in ns-3 these are asynchronous.

6.2.2 Extending ns-3: Proteo Module

Proteo Module, which implements the proposed service architecture, is designed and implemented as a module of ns-3. This module provides a ns-3 *Application* that can be installed in a ns3 *Node*. In ns3, applications represent the software entities located at the Application layer in the TCP/IP model and can make use of the lower layers (i.e., Transport, Internet and Network Interface). *Proteo Module* is implemented in C++ and is made up of a total of **13615** code lines. This module, as it will be seen in Section 6.2.3, will interact with other already existing modules in ns-3, which implement the Internet stack, the NetDevice, and the Communication Channel. The directory and the file hierarchy of the implemented module are shown in Fig. 6.2.

The elements of the architecture are implemented in:

- *Replica Manager Service* (Section 5.4.3) is in *REPLICAMANAGER* folder, Fig. 6.2. It is implemented as an abstract class that is specialized according to the election algorithm: Bully, Kordafshari, Vasudevan, Consensus or Voting (Section 5.5.2). A non-adaptive replica manager is implemented, as well.

- *Network Monitor* and *Battery Monitor* services (Section 5.4.1) are in *MONITORS* folder in Fig. 6.2. *Network Monitor* supervises the routing table, which is maintained at the Internet stack. *Battery Monitor* interacts with the energy module, which will be installed in the nodes and will manage the energy levels of the node.

- *Context Manager Service* (Section 5.4.2) is in the *CONTEXTMANAGER* folder Fig. 6.2.

Finally, in *APPS* folder in Fig. 6.2, in order to make possible the module to be installed as an application in a node, a specialization of the ns3 *Application* is implemented. This is the bridge between the ns3 *Node* and the *Proteo Module*. Its main responsibility is initializing all *Proteo Module* components.

Additionally to the software architecture, in the module there are implemented a set of necessary components for performing the evaluation with ns-3:

- **Communication management** (implemented in the *COMMUNICATOR* folder, Fig. 6.2). Since the highest level provided by ns-3 is socket level, it is necessary to provide a layer to manage sockets (UDP or TCP) and *Coordination Messages*. This can be

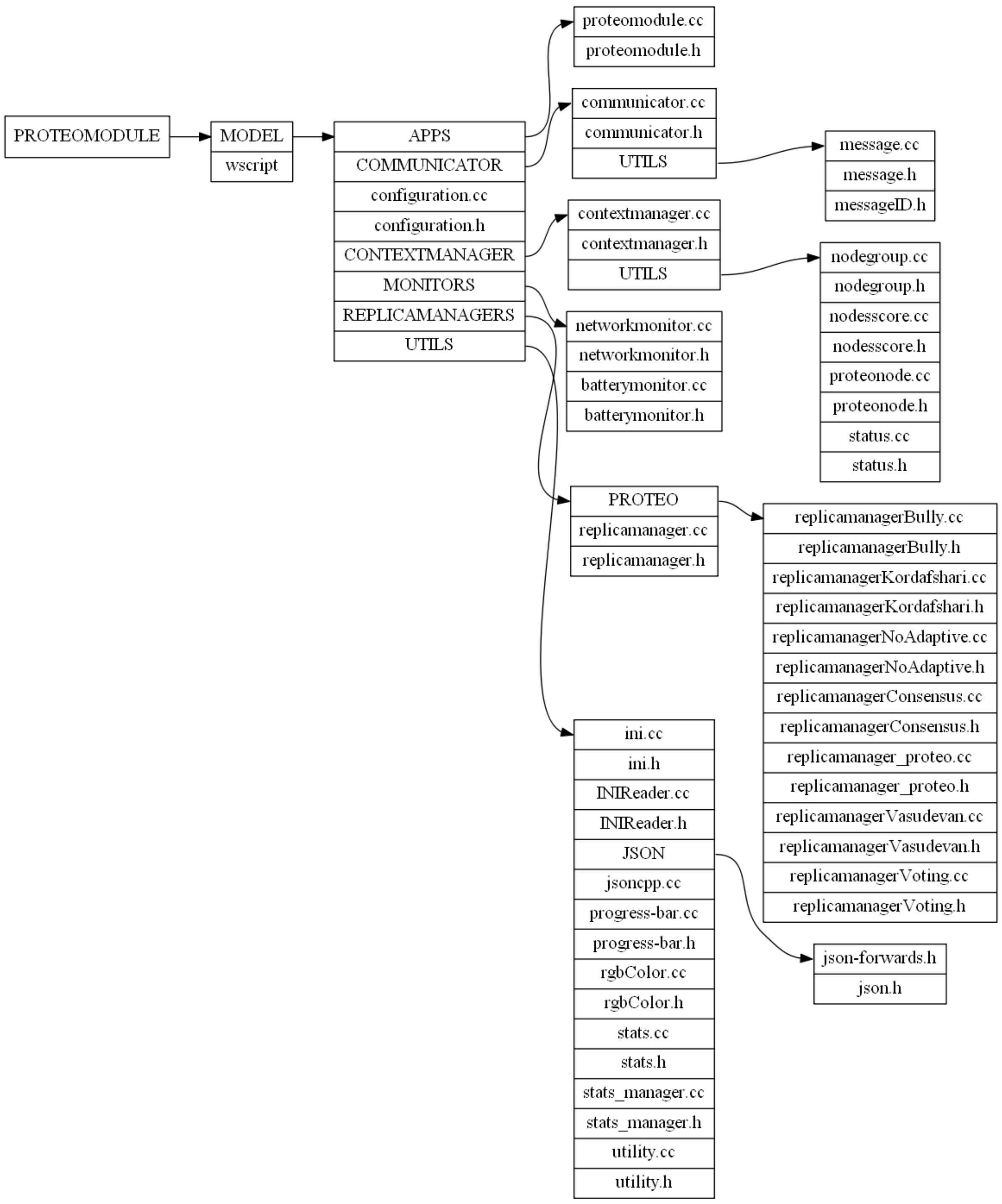


Fig. 6.2 *Proteo Module structure directory on ns-3 implementation. Folders are shown in capital letters.*

considered as a simple communication middleware, which abstracts the communication details to other components of *Proteo Module*.

- **Measures management** (implemented in the *stats_manager* class under *UTILS* folder, Fig. 6.2). To implement a module that monitors and stores the behaviour of the node during the simulation is necessary to evaluate the performance of the architecture. The measures that are taken are: time that node spends on each specific state (*Client*, *Server*, *Local* or *Connecting*), *Coordination Messages* sent and received, KB sent and received by the *Proteo Module*, KB sent and received by the routing protocol, and the battery levels of the node.

- **Configuration manager** (implemented in the *configuration* class under *MODEL* folder, Fig. 6.2). A configuration manager is provided to modify the configuration of the node and the simulation, without recompile the module. This manager reads a configuration file where different features of the *Proteo Module* can be modified, such as the election algorithm used by the *Replica Manager*, transport layer protocol (UDP or TCP), battery capacity and initial battery levels, mobility model, number of nodes in the network, utility function weights, simulation duration, seed, and output verbosity level.

6.2.3 Configuring a ns3 Node to host Proteo Module

In our implementation, the high-level architecture of a node is shown in Fig. 6.3.

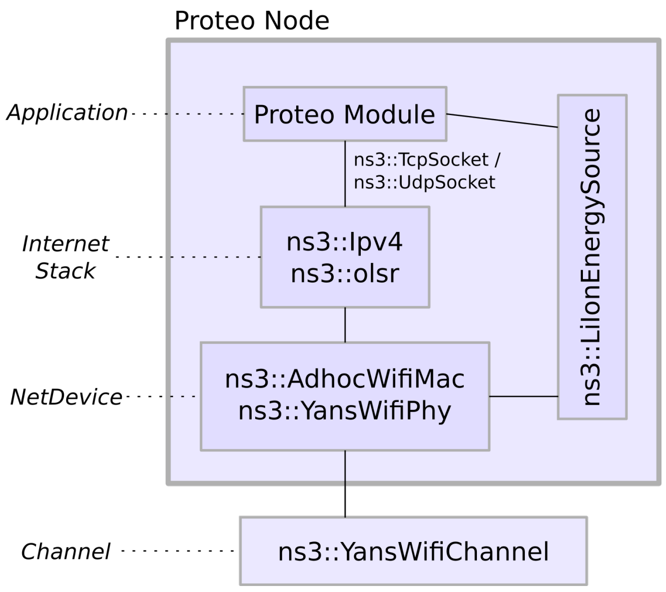


Fig. 6.3 *High-level Proteo node architecture in ns-3.*

The communication channel used is *ns3::YansWifiChannel*, in order to interconnect *ns3::YansWifiPhy NetDevices*. The propagation model implemented by this Wi-Fi channel is described in [92], which follows the 802.11 specification. In the ns-3 script configuration, it has been configured as shown in Code 6.1.

Code 6.1 Communication channel configuration in ns-3 for Proteo simulation.

```
YansWifiChannelHelper wifiChannel;
wifiChannel.SetPropagationDelay("ns3::ConstantSpeedPropagationDelayModel");
wifiChannel.AddPropagationLoss("ns3::TwoRayGroundPropagationLossModel",
"SystemLoss", DoubleValue(1), "HeightAboveZ", DoubleValue(1.5));
```

Where *"ns3::ConstantSpeedPropagationDelayModel"*, sets a constant speed propagation on the channel; and *"ns3::TwoRayGroundPropagationLossModel"*, a Two-Ray Ground propagation loss model, which considers both the direct path (single line-of-sight) of communication between two nodes and a ground reflection path [113].

The *NetDevice* used has been *ns3::YansWifiPhy*, and it has been configured as shown in Code 6.2.

Code 6.2 NetDevice installation and configuration in ns-3 for Proteo simulation.

```
std::string phyMode ("DsssRate1Mbps");

// Set up WiFi
WifiHelper wifi;
YansWifiPhyHelper wifiPhy = YansWifiPhyHelper::Default ();
wifiPhy.SetPcapDataLinkType(YansWifiPhyHelper::DLT_IEEE802_11);

// For range near 250m
wifiPhy.Set("TxPowerStart", DoubleValue(33));
wifiPhy.Set("TxPowerEnd", DoubleValue(33));
wifiPhy.Set("TxPowerLevels", UintegerValue(1));
wifiPhy.Set("TxGain", DoubleValue(0));
wifiPhy.Set("RxGain", DoubleValue(0));
wifiPhy.Set("EnergyDetectionThreshold", DoubleValue(-61.8));
wifiPhy.Set("CcaMode1Threshold", DoubleValue(-64.8));

// Bind NetDevice with communication Channel
wifiPhy.SetChannel(wifiChannel.Create());

// Add a non-QoS upper mac
NqosWifiMacHelper wifiMac = NqosWifiMacHelper::Default ();
wifiMac.SetType ("ns3::AdhocWifiMac");
```

```
// Set 802.11b standard
wifi.SetStandard(WIFI_PHY_STANDARD_80211b);
wifi.SetRemoteStationManager("ns3::ConstantRateWifiManager", "DataMode",
StringValue(phyMode), "ControlMode", StringValue(phyMode));

NetDeviceContainer devices = wifi.Install (wifiPhy, wifiMac, mobileNodes);
```

In this way, power transmission parameters of the NetDevice has been set to approximate a wireless communication range of 250 meters. The enumerator of the pcap data link types has been set to *YansWifiPhyHelper::DLT_IEEE802_11*, this is, the IEEE 802.11 Wireless LAN headers on packets, and the standard used is 802.11b (*WIFI_PHY_STANDARD_80211b*). The MAC high model in the *NetDevice* is set to *"ns3::AdhocWifiMac"*, this is a non-QoS MAC, which does not perform any kind of beacon generation, testing, or association.

Regarding Internet stack, it has been used the IPv4 protocol (the IP directions assigned to the nodes range from 10.1.1.1 to 10.1.1.254) and OLSR routing protocol [79], as a proactive routing protocol. This is configured in ns-3 as shown in Code 6.3.

Code 6.3 Internet stack configuration in ns-3 for Proteo simulation.

```
//  Enable OLSR
OlsrHelper olsr;
// Set up internet stack
InternetStackHelper internet;
internet.SetRoutingHelper(olsr);
internet.Install(mobileNodes);

// Set up Addresses
Ipv4AddressHelper ipv4;
ipv4.SetBase("10.1.1.0", "255.255.255.0");
Ipv4InterfaceContainer ifcont = ipv4.Assign(devices);
```

For the energy model of the devices, the *ns3::LiIonEnergySource* has been used. This model represents the non-linear discharge of a generic lithium-ion battery, based on the studies of [148] and [160]. This model takes into consideration both the natural discharge of a battery and the energy consumption of the *NetDevice*. It has been configured in ns-3 as it is shown in Code 6.4.

Code 6.4 Battery configuration in ns-3 for Proteo simulation.

```
/* energy source */
LiIonEnergySourceHelper liionSourceHelper;
```

```
// configure energy source
liionSourceHelper.Set ("RatedCapacity",
DoubleValue (batt_conf->getReal("battery","capacity")));
// install source
EnergySourceContainer sources = liionSourceHelper.Install (mobileNodes);
/* device energy model */
WifiRadioEnergyModelHelper radioEnergyHelper;
// configure radio energy model
radioEnergyHelper.Set ("TxCurrentA", DoubleValue (0.0174));
radioEnergyHelper.Set ("RxCurrentA", DoubleValue (0.0197));
// install device model
DeviceEnergyModelContainer deviceModels =
radioEnergyHelper.Install (devices, sources);
```

Where *batt_conf->getReal("battery","capacity")* obtains from the Proteo configuration file the battery capacity specified for the simulation.

Finally, *Proteo Module* (Section 6.2.2) is installed in the node as a ns3 *Application*.

6.2.4 Variations on ns-3

To make possible the evaluation and validation of the proposed architecture in ns-3, two main variations on the network simulator have been necessary.

The first of them is related to the cross-layer approach that Proteo architecture follows to detect changes in the network topology (see Section 4.4.1). Proteo uses the information gathered by the routing protocol (i.e., the routing tables) with two objectives: (1) to monitor and detect changes in the network topology; and (2) to use this information as a substitute of a discovery service. That information is not accessible from the *ns3::Application* layer. Therefore, a new ns3::callback[2] has been added to *ns3::Ipv4RoutingProtocol* that is triggered when the routing table changes, providing the new information of the current routing table. In this way, now, any entity from *ns3::Application* layer can bind to this new callback and receive the information of the routing table.

The second variation is related to battery-based nodes in ns-3. Whereas ns-3 provides advances models regarding energy consumption, it does not allow to add or remove nodes at run-time, neither to deactivate or delete a node when its energy is depleted. In this way, when the *ns3::EnergySource* is depleted, the node continues to operate normally. Therefore, to accurately evaluate the operating conditions of battery-based environments and a resource-aware solution, a new callback has been added to *ns3::EnergySource*. This callback is triggered when the

[2]Callbacks in ns-3, https://www.nsnam.org/docs/manual/html/callbacks.html

Table 6.1 *Node setting parameters used in ns-3 to perform the simulations.*

Wi-Fi Standard	802.11b
Transportation Protocol	TPC
Data Rate	250 Kbps
Routing Protocol	OLSR
Connection Range	250 m.
Battery Capacity	2.45 Ah
Initial Charge	[1.0-2.45] Ah

energy source is depleted and *ns3::Application*, *ns3::NetDevice* and *ns3::MobilityModel* are binding to them. Hence, now with this change, when the *ns3::EnergySource* of the node is depleted, the *ns3::Application* stops its operation. The *ns3::NetDevice* is deactivated to avoid that a node with a depleted battery can act as a router in a multi-hop communication; and the *ns3::MobilityModel* is also deactivated so that the node stops moving.

6.3 Simulation Settings

This section describes the settings that have been established in the simulations to evaluate the proposal. This information is essential to know under what circumstances the architecture has been evaluated and thus to correctly assess the results obtained.

6.3.1 General and Node Settings

For the evaluation of the proposal, the 3.21 version of ns-3, released in September 2014, has been used. As shown in Section 6.2.3, the communication channel has been configured with a constant speed propagation delay model and a two-ray ground propagation loss model. A lithium-ion energy model manages the battery of the nodes. The battery is configured with a maximum capacity of 2.45 Ah, and the initial charge of the battery is randomly set in the range [1.0-2.45] Ah (Table 6.1). Moreover, the nodes communicate under an 802.11b wireless connection, under the TCP transportation protocol.

The number of nodes in the network will be incrementally ranged from 4 to 20. Each node will have the Proteo Module installed (Section 6.2.2). The Proteo Module will be evaluated with each one of the five election algorithms implemented, under each one of the three mobility models (Section 6.3.2). The duration of the simulations has been of 21600 seconds (i.e., six hours) each one, enough time for the nodes to deplete their batteries.

To simulate the execution of a service, an energy penalty has been introduced. The node hosting the active replica, i.e., the node acting as the server, will consume battery twice more quickly than other nodes, in addition to the battery consumed by the network card.

Finally, to eliminate the influence of any possible random factor, each configuration has been simulated 100 times with 100 different random seeds. The seed influences on all random factors during the simulation, which are the initial positions of the nodes, the initial battery charges and the movement of the nodes. For the same seed, these factors will be identical. Therefore, the different election algorithms are compared under identical circumstances.

6.3.2 Mobility Settings

As exposed by [177] and [178] (see Chapter 3), the mobility model can have a deep impact in the performance of the proposal. Thus, Proteo architecture will be evaluated under three different mobility models: *Manhattan Grid*, *Random Walk* and *Reference Point Group*. The features of these models and the specific setting parameters used in each of them are detailed below.

Manhattan Grid

The Manhattan Grid mobility model [162] intends to represent an urban area (e.g., a grid of roads or pedestrian pavements) or an indoor scenario (e.g., a grid of corridors). This mobility model could be considered related to scenarios of *Advanced Intelligent Transportation Systems* [32] or, for example, tourism in *Smart Cities* [71].

The Manhattan Grid restricts the movement of the nodes to a set of horizontal and vertical paths, which cross each other. When a node is situated at an intersection of a vertical and a horizontal path, it can turn right, left or go straight according to a defined probability.

Under this model, the probability of two nodes to meet could be considered low, since to get in contact, these nodes have to concur in the same path or near of a cross. Of course, this probability will depend on the number of nodes and the number of paths of which the grid is composed. Thus, under this mobility model, the nodes will spend more time on the Local state, and the network partitions will have a more reduced number of nodes than under other mobility models.

Table 6.2 shows the configuration parameters used to generate the mobility trace files under the Manhattan Grid model using Bonnmotion tool (Section 2.6.1). It has been generated a scenario of 1000 m2 with 14 blocks along the x-axis and another 14 along the y-axis. This means a total of 15 horizontal paths and another 15 vertical paths. The mean speed of the nodes

Table 6.2 *Setting parameters used for the Manhattan Grid model in Bonnmotion (Section 2.6.1) to perform the simulations.*

Speed change probability	0.4
Min. speed	0.5 m/s
Mean speed	1.5 m/s
Speed SD	0.5
Max. pause	350 s
Pause probability	0.2
Update distance	50 m
Turn probability	0.5
Blocks along x-axis	14
Blocks along y-axis	14
Scenario max. x	1000 m
Scenario max. y	1000 m

has been set to 1.5 m/s, which is intended to represent the preferred walking speed of humans (1.4 m/s [104]). The nodes have a probability of 0.2 to make a pause of maximum duration 350 seconds, this is, about 6 minutes. During this pause, it can be assumed that the users are performing some task on their mobile devices. The speed of a node is revised every 50 meters, and it can change with a probability of 0.4.

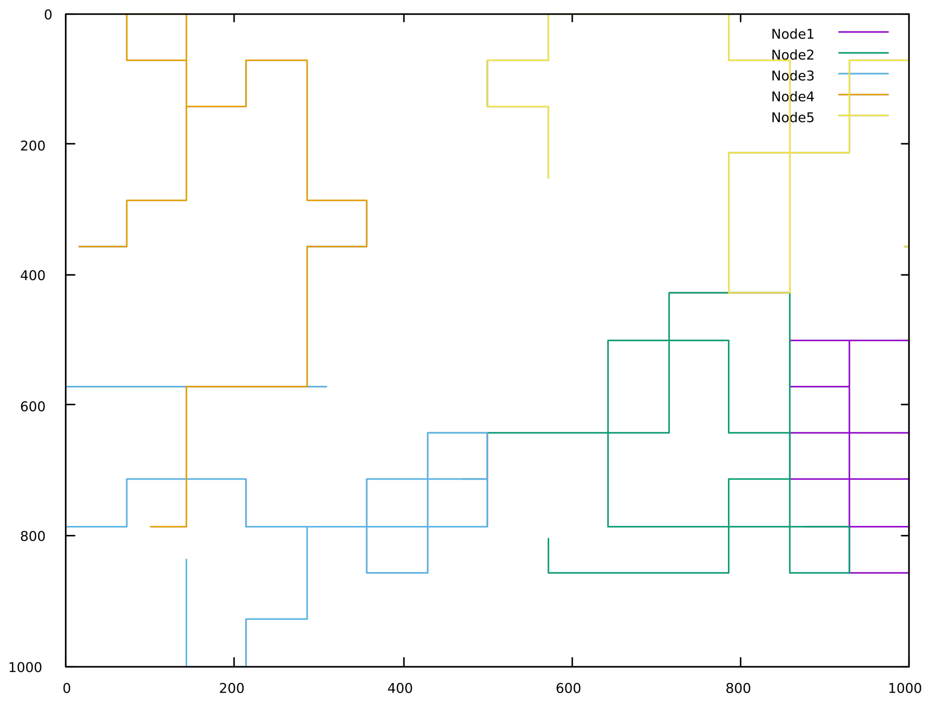


Fig. 6.4 *A plot of the trace of an example Manhattan Grid scenario with 5 nodes and a duration of 1600 seconds, using the configuration parameters described on Table 6.2.*

Table 6.3 *Setting parameters used for the Random Walk model in ns3 to perform the simulations.*

Min. speed	0.5 m/s
Max. speed	2.0 m/s
Min. pause	60 S
Max. pause	350 s
Scenario max. x	1000 m
Scenario max. y	1000 m

Fig. 6.4 shows a plot of the trace that five nodes have followed under a the Manhattan Grid model scenario with a duration of 1600 seconds, using the configuration parameters described in Table 6.2. As it can be seen, some lanes only have been covered for certain nodes and there are lanes that have not been covered by any of them.

It should be noted, that **the scenarios depicted in Fig. 6.4 and the following (Fig. 6.5 and Fig. 6.7) are examples** generated to provide a better understanding of the behaviour of the nodes under the different mobility models and under the configuration provided. In the evaluation performed, these scenarios will vary according to the seed provided.

Random Walk

The Random Walk mobility model [25] is one of the most widespread models, due to its simplicity. It usually consists of a rectangular area, which is entirely walkable. Nodes have an individualist behaviour and move between points with random directions and speeds. Although this cannot be considered a real-world scenario, since in realistic scenarios the movement of the nodes (people or vehicles) is not random, it is considered an excellent test scenario. This is due to it represents the worst-case scenario regarding dynamicity on network topology, which is traduced in an unstable ad-hoc communication network. Under this scenario, the probability of two nodes to meet is higher than under the Manhattan Grid model.

It should be noted that ns-3 provides native support to generate the Random Walk mobility scenarios. Therefore, in this case, the support of Bonnmotion has not been necessary. In Table 6.3 the configuration parameters used to generate mobility scenarios under the Random Walk model are shown. It is intended to generate a scenario where the nodes have similar features regarding speed and pauses than on the other mobility models. Hence, it has been generated scenarios of 1000 m^2, where the nodes move with a speed between 0.5 and 2.0 m/s, with random pauses between 60 and 350 seconds.

Fig. 6.5 shows a plot of the trace that ten nodes have followed under a the Random Walk model scenario with a duration of 1600 seconds, using the configuration parameters described in Table 6.3. As it can be seen, the movement of the nodes is entirely random, and whereas

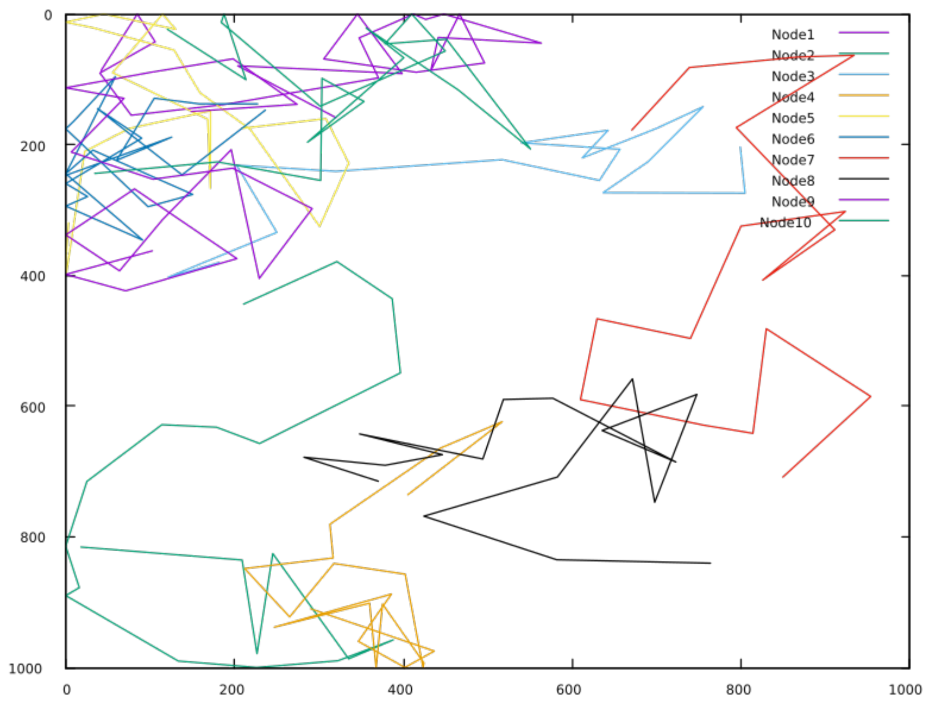


Fig. 6.5 *A plot of the trace of an example the Random Walk scenario with 10 nodes and a duration of 1600 seconds, using the configuration parameters described on Table 6.3.*

near the upper-left corner (point (0, 0) in Fig. 6.5) a group of 6 nodes has remained relatively close, the other four nodes have moved more individually throughout the rest of the area.

Reference Point Group Mobility

The Reference Point Group Mobility model (RPGM) [68] is devised to simulate group behaviour in open areas. This mobility model could be considered related to scenarios of *Emergency Scenarios & Situations* or *Smart Cities*, for example, a system to support students in a university campus, providing services to document sharing or information [151] [23].

In the RPGM, the nodes of the network are divided randomly into groups. Each group has a fictional group leader node that determines the direction and speed of the group. The group members derive their direction and their speed from the group leader. Thus, the group members have an independent behaviour within their group, slightly varying their movements from those decided by the group leader. This mobility model intends to represent a more realistic mobility model, where groups of people move along an open area. This could represent tourist or rescue crews from different groups that work cooperatively.

Table 6.4 *Setting parameters used for the RPGM model in Bonnmotion to perform the simulations.*

Min. speed	0.5 m/s
Max. speed	2.0 m/s
Max. pause	350 s
Average no. of nodes per group	$\left[\frac{total\ no.\ of\ nodes}{3}\right]$
Group change probability	0.1
Max. distance to group centre	$(\frac{Scenario\ max.}{10})$ = 100 m
Group size SD	2
Scenario max. x	1000 m
Scenario max. y	1000 m

The implementation of this model also is provided by the Bonnmotion tool. It is based on the design given by [68] and [25]. Whereas the work in [68] does not provide pausing behaviour, the implementation provided by Bonnmotion takes into consideration the design of the Random Walk mobility model [25] and determines that the group leader could pause for a determined amount of time according to a defined probability. When the group leader pauses, the group members pause for the same amount of time. Moreover, the Bonnmotion implementation provides the possibility of "dynamic groups", where, when two groups overlap on the same area, a group member node could switch groups according to a determined probability.

In Table 6.4 the configuration parameters used to generate mobility scenarios under the RPGM model with Bonnmotion are shown. As stated previously, it is intended to generate scenarios where the nodes have similar features regarding speed and pauses than on the other mobility models. Scenario size is of 1000 m^2, where the nodes move with a speed between 0.5 and 2.0 m/s, with random pauses of maximum 350 seconds. As in the Random Walk mobility model, the whole scenario area is walkable under the RPGM.

Regarding specific parameters of the RPGM model, it has been determined the average number of nodes per group as a third of the total number of nodes in the network with a standard deviation of 2. This means, for a network of 10 nodes, that the average number of nodes per group will be of 3 ± 2 nodes. The evolution of the size of the group according to the total number of nodes in the network and the configuration parameters provided can be shown in the chart of Fig. 6.6. Finally, the maximum distance of the group members to the centre of their group is of a tenth of the maximum size of the scenario, this is 100 m. According to the configuration of the wireless network adapters of the nodes, the range of the wireless

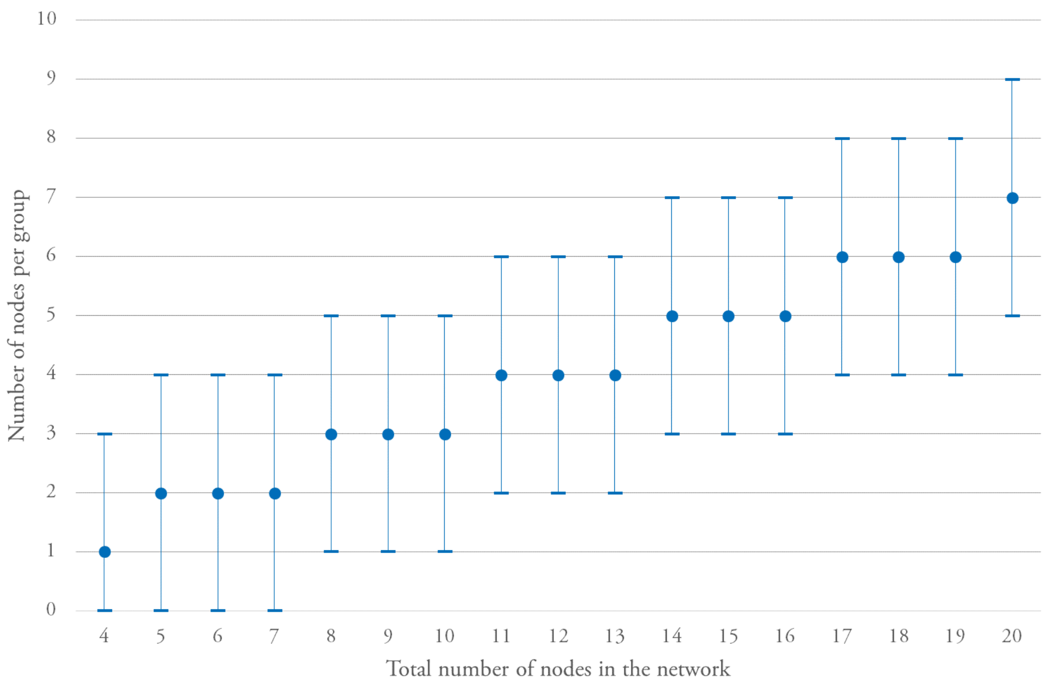


Fig. 6.6 *Number of nodes per group under the RPGM model according to the total number of nodes in the network and the configuration parameter provided in Table 6.4.*

connection is approximately of 250 m. Hence, the maximum distance set to the group centre guarantees a stable communication between the nodes of the same group.

In comparison with the other mobility models, the RPGM provides, on the one hand, a more stable network topology, but on the other hand, the number of nodes involved in the election process is more significant, especially when two or more groups of nodes meet. Using the setting parameters provided in Table 3, Fig. 6.7 shows a plot of the trace that ten nodes have followed under an RPGM model scenario with a duration of 1600 seconds. It can be seen how there are three groups of nodes: (1) that composed by Node8, Node9, and Node10; (2) that composed by Node4, Node5, Node6 and Node7; and (3) that composed by Node1, Node2, and Node3. It can also be seen how Node10 switches groups, from group 2 to group 1. Finally, it can be seen, that although there are three different groups, sometimes these meet forming a group that encompasses every node on the network. Although this is also possible under the Manhattan Grid and Random Walk mobility models, under the RPGM model this is more frequent since the nodes have a groupal behaviour.

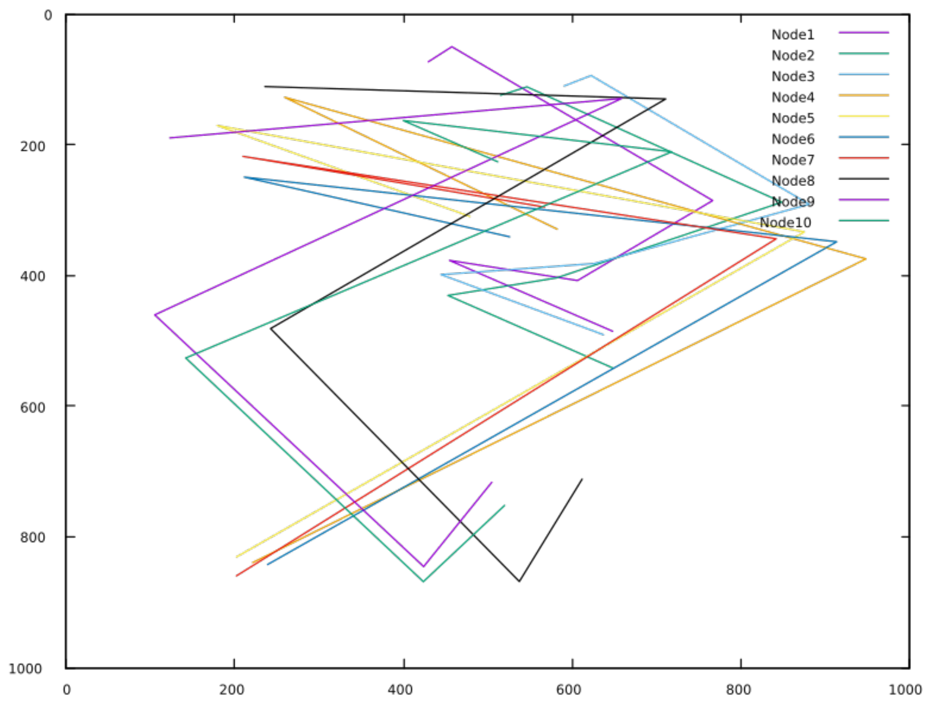


Fig. 6.7 *A plot of the trace of an example RPGM scenario with 10 nodes and a duration of 1600 seconds, using the configuration parameters described on Table 6.4.*

6.4 Summary

This chapter has provided an overview of the design and development of Proteo in the ns-3. Proteo architecture has been developed as a new module in ns-3, denominated *Proteo Module*. This module provides a ns-3 *Application* that can be installed in a ns3 *Node*. *Proteo Module* is implemented in C++ and is made up of a total of **13615** code lines. This module interacts with other already existing modules in ns-3. Some of these are *ns3::YansWifiChannel* which implements the model of the WiFi channel, *ns3::YansWifiPhy NetDevice* which implements the model of the wireless network layer to device interface, and *ns3::LiIonEnergySource* which implements the model for the energy source of the node.

Additionally, to make possible the evaluation and validation of the proposed architecture in ns-3, two main variations on the network simulator have been performed: (1) a new ns3::callback has been added to *ns3::Ipv4RoutingProtocol* that is triggered when the routing table changes, to allow Proteo Module to access the information of the routing table, allowing a cross-layer interaction; and (2) a new callback has been added to *ns3::EnergySource* to deactivate the nodes when the energy source is depleted, feature that is not implemented by default in ns3.

Moreover, in this chapter, the settings established in the simulations to evaluate the proposal have been presented. The battery of the nodes is configured with a maximum capacity of 2.45 Ah, and the initial charge of the battery is randomly set in the range [1.0-2.45] Ah. The number of nodes in the network will be ranged from 4 to 20. The duration of the simulations has been of 21600 seconds (i.e., six hours) each one. To eliminate the influence of any possible random factor, each configuration has been simulated 100 times with 100 different random seeds. Finally, the architecture is evaluated under three mobility models: *Manhattan Grid*, *Random Walk* and *Reference Point Group*.

Chapter 7

Simulation Results in ns-3

Chapter Abstract

This chapter shows and analyses the results obtained during the simulations. The analysis has been carried out in terms of service availability, election algorithm reliability, use of coordination messages and network lifetime with different mobility models and elections algorithms. The results obtained have shown that Bully, Kordafshari, and Voting provide the better results regarding service availability. Nevertheless, the reliability of Bully and Kordafshari regarding concurrent elections is low. To this regard, algorithm reliability, Voting, and Vasudevan provide the best results. However, Vasudevan, especially when there are several nodes involved in an election, make more intensive use of coordination messages than the other algorithms. Low reliability (i.e., a high number of concurrent elections) and high use of coordination messages have a negative impact in the network lifetime. For this reason, Voting provides the better results to this respect, in comparison with Kordafshari or Vasudevan. In conclusion, globally, the Voting election algorithm proves to show the best behaviour, although Kordafshari provides a slight improvement in service availability at the expense of reliability and Vasudevan provides a slight improvement in algorithm reliability at the expense of coordination message usage. Regarding the Consensus algorithm, although it provides a slight improvement on the reliability in comparison to the Bully and Kordafshari algorithms, globally speaking it does not provide any relevant characteristic that differentiates it over the other algorithms.

Chapter Contents

7.1 Introduction

This chapter presents the results obtained during the evaluation of our proposal. The evaluation of Proteo architecture has been carried out in the ns-3 network simulator, under the simulation conditions presented in Section 6.3.

The chapter is organized as follows: first, each quality attribute is studied individually for each of the five election algorithms implemented (Bully, Kordafshari, Consensus, Vasudevan, and Voting) under the three mobility models (Manhattan Grid, Random Walk, and RPGM). These features are service availability (Section 7.2); election algorithm reliability (Section 7.3), in terms of redundant server elections; coordination messages used by the election algorithms (Section 7.4); and network lifetime (Section 7.5), in terms of Time until First Node Dies (TFND), Time until Half of the Nodes Dies (THND) and Time until the Last Node Dies (TLND).

Additionally, as a reference, a non-adaptive solution has been implemented where the half of the mobile nodes of the network operate as servers by default. This version of the architecture is not entirely measurable, for example, it does not use any coordination messages, and thus it only will be taken into consideration in some of the features evaluated.

Then, these features are faced in terms of efficiency (Section 7.6), confronting target qualities (service availability and algorithm reliability) against cost measures (messages used and network survivability (TLND)). Finally, a global performance analysis of each algorithm is presented, concerning service availability, algorithm reliability, message used (i.e., bandwidth usage) and network lifetime (i.e., energy consumption). It should be noted that in this chapter the results obtained are displayed using charts, to facilitate comparisons of the results obtained with the different configurations. The complete data tables can be found in Appendix A.

7.2 Service Availability

From the general definition of service availability provided in Section 2.5.5, in the context of this thesis, **service availability** is defined as the time in which a node has a successful client-server relation established when there is at least one node reachable. Therefore, service availability will be directly related to the time that takes the node to reach the Server or Client state from the Connecting state. Fig. 7.1, Fig. 7.2 and Fig. 7.3 show the results obtained regarding service availability provided by the architecture using the different election algorithms under the Manhattan Grid mobility model, the Random Walk mobility model, and the RPGM mobility model, respectively.

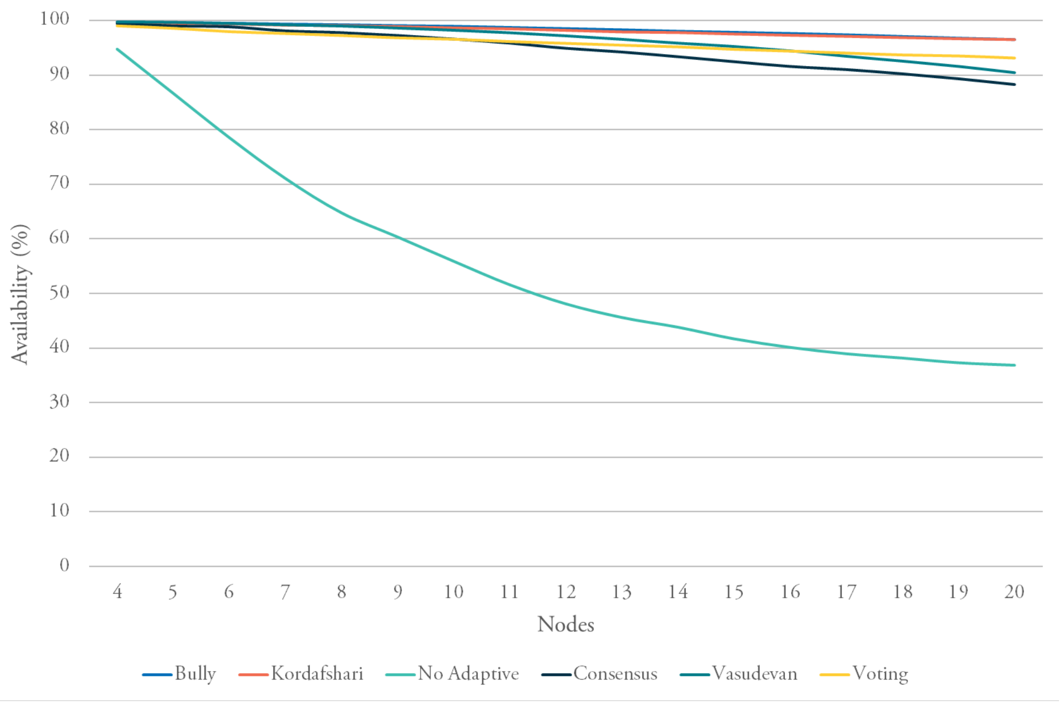


Fig. 7.1 *Service availability provided by the different election algorithms under the Manhattan Grid mobility model. Data on Table A.1.*

In the case of the Manhattan Grid (Fig. 7.1) the availability provided is high, generally over the 90%. This is because in the Manhattan Grid the available area is highly restricted. Therefore the nodes will meet in reduced groups with short connection distances. In this way, the problems caused by connection fading is reduced, and the reduced groups imply short elections. With this mobility model, the best Service availability is provided by the Bully and Kordafshari algorithms, with an availability of 96.50% and 96.49% respectively on a network of 20 nodes, in contrast to the Consensus algorithm which provides an availability of 88.27%.

In the case of the Random Walk (Fig. 7.2) three groups can be differentiated on a network of 20 nodes: (1) the Bully and Kordafshari algorithms, which again provide the best availability, 95.19%, and 95.69% respectively. (2) the Voting algorithm, which provides a slightly fewer availability of 92.08%. (3) the Consensus and Vasudevan algorithms, which provides the lowers results, 83.04% and 84.82% respectively.

Finally, in the case of the RPGM (Fig. 7.3), where the number of nodes involved in the election tends to be higher than in the other mobility models, two groups can be differentiated on a network of 20 nodes: (1) the Bully, Kodafshari and Voting algorithms, which provide an availability of 97.40%, 97.98% and 96.48% respectively; and (2) the Consensus and Vasudevan algorithms, which provide an availability of 88.01% and 82.55% respectively. It worth noting

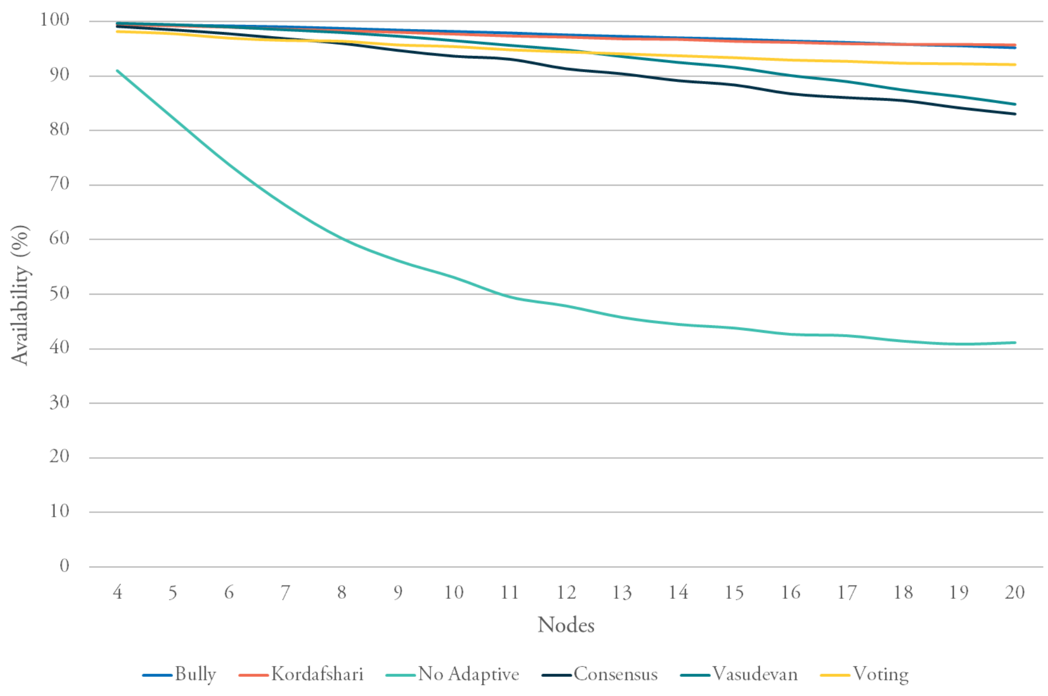


Fig. 7.2 *Service availability provided by the different election algorithms under the Random Walk mobility model. Data on Table A.2.*

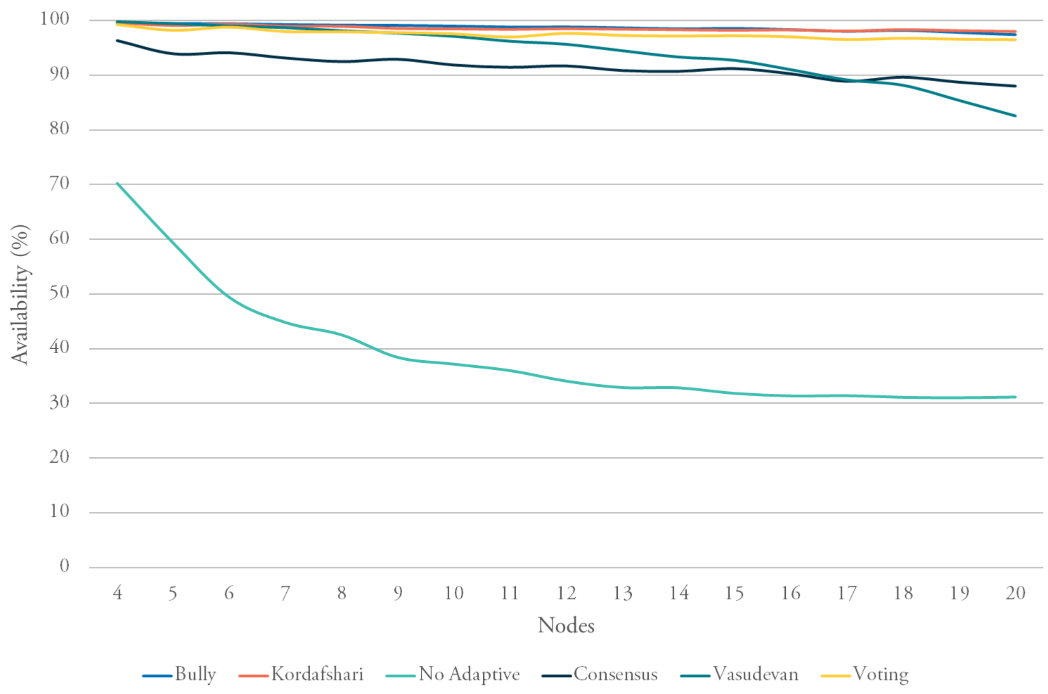


Fig. 7.3 *Service availability provided by the different election algorithms under the RPGM mobility model. Data on Table A.3.*

that the availability provided by the first group of nodes has been little affected by the increasing number of nodes in the network, whereas this fact has had a profound impact in the performance of the Vasudevam algorithm.

In conclusion, the Bully and Kordafshari algorithms have shown the best performance in terms of service availability, independently of the mobility model. The Voting algorithm has shown better performance on the RPGM mobility model than on the others mobility models. In the RPGM model, in comparison with the other mobility model applied, there is a high number of nodes involved in the election, although the groups of nodes are more stable. This has had a deep impact on the performance of the Vasudevan algorithm, which sees its performance profoundly reduced when the number of nodes increases, both on the RPGM and the Random Walk mobility models. Finally, the Consensus algorithm, generally, provides the worst results regarding service availability.

7.3 Algorithm Reliability

From the definition of reliability (Section 2.5.3) to measure the reliability of a system, or usually, of a system component, it is firstly required to define when it fails. According to the host election problem definition provided (Section 2.4), in this thesis, we assume as a failure when more than one node is designed to act as the server in the same network partition, i.e., when a redundant server election happens. This failure happens when some of the coordination messages are lost, usually when the links within the nodes are not stable, which leads to an unstable network partition.

According to the previous definitions, two main measures are considered to evaluate the reliability of the host election algorithm: (1) the percentage of failures (i.e., redundant server elections) according the total executions of the election algorithm during the simulation (Section 7.3.1), (2) and the Mean Time Between Failures (MTBF) (Section 7.3.2).

Contrary to service availability, which is directly and unequivocally measurable, the way in which reliability is measured can vary the results obtained, especially in a distributed system that relies on a non-reliable communication system. For this reason, it is necessary to identify clearly for each moment: the network partitions, the nodes that compose them and if these are well-established network partitions where the election algorithm could operate correctly. This has to be done under the following circumstances (see System Model on Section 4.3):

- Although bidirectional links are assumed; it is also considered that while the connection between two nodes is being established links could be not bidirectional.

- A network partition is considered stable when each link in the network partition is bidirectional.
- The system is fully distributed and there is no any central supervisor entity that can unequivocally determine, at each moment, to which network partition each node belongs. However, for all the nodes of the network, to have synchronized clocks is assumed.

Therefore, according to the definition of failure considered, the following approach has been implemented to measure the reliability provided by the elections algorithms:

- Each node stores a log of its reachable neighbour nodes and its state at that moment (*Local*, *Client*, *Server* or *Connecting*). This log is updated when a change on the reachable neighbours or node state happens.
- Once the operating time has finished, these logs are processed together, providing an adjacency matrix that represents the links between the nodes for each moment, additionally to the state of each node.
- With this matrix, the stable network partitions are determined. A stable network partition is determined if its adjacency matrix is symmetric, that means that links are bidirectional and thus the reliability of the algorithm can be measured. If not, the network partition is unstable, and the reliability of the election algorithm should not be evaluated.

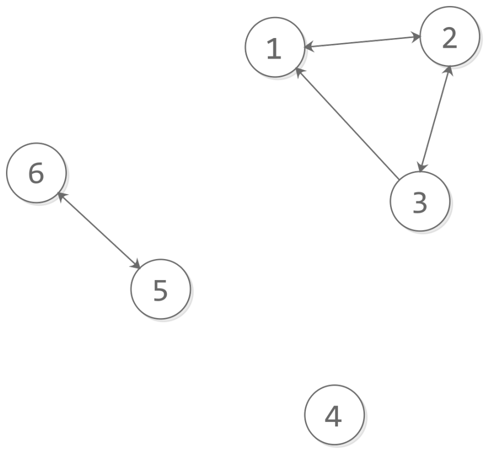


Fig. 7.4 *A directed graph which represents the nodes of a network and the links between them. In the graph three network partition can be seen: (1) composed by the nodes 1, 2 and 3; (2) composed by nodes 5 and 6; and (3) composed only by the node 4.*

Lets consider the example depicted in Fig. 7.4, which shows a directed graph that represents the nodes of a network and the links between them. In the graph, three network partition can be

Table 7.1 *Adjacency matrix representing the network depicted in Fig. 7.4. Where $A_{(i,j)} = 0$ means that there is no link from the node i to j, and $A_{(i,j)} = 1$ means that there is a communication link established from i to j.*

	1	2	3	4	5	6
1	-	1	0	0	0	0
2	1	-	1	0	0	0
3	1	1	-	0	0	0
4	0	0	0	-	0	0
5	0	0	0	0	-	1
6	0	0	0	0	1	-

seen: (1) composed by the nodes 1, 2 and 3; (2) composed by nodes 5 and 6; and (3) composed only by the node 4.

The network of Fig. 7.4 can be represented by the adjacency matrix of Table 7.1. In this matrix to determine the three network partitions is possible. Also, it can be seen that the network partition composed of the nodes 1, 2 and 3 is still unstable since the link between nodes 1 and 3 is not bidirectional. On the contrary, the network partition composed of nodes 5 and 6 is stable, and the reliability of the network partition can be measured. Thus if during the lifespan of this stable group there is more of one node acting as a server, a redundant election has happened.

7.3.1 Redundant Server Elections

Fig. 7.5, Fig. 7.6, and Fig. 7.7 show the results obtained regarding the reliability provided by the election algorithms in terms of redundant server elections for the Manhattan Grid mobility model, the Random Walk mobility model, and the RPGM mobility model, respectively.

As can be seen in the data obtained, the Vasudevan and Voting algorithms provide excellent reliability regardless the mobility model and the number of nodes. The percentage of redundant server elections is always lower than 0.4%.

On the contrary, the Bully and Kordafshari algorithms provide poor reliability, which is mainly affected by the increasing number of nodes in the network. In Table 7.2, it can be seen that redundant server elections increment, from a network of four nodes to a network of twenty nodes, a 10.27% for the Bully algorithm and a 10.20% for the Kordafshari algorithm. In the case of the RPGM mobility model (Fig. 7.7), owing to network partitions are generally more extensive (i.e., more nodes are participating in the election process), the increase in the redundant server elections worsens. This results in an increase of 14.86% for the Bully algorithm, with a total of 17.41% of redundant elections for a network of twenty nodes, and

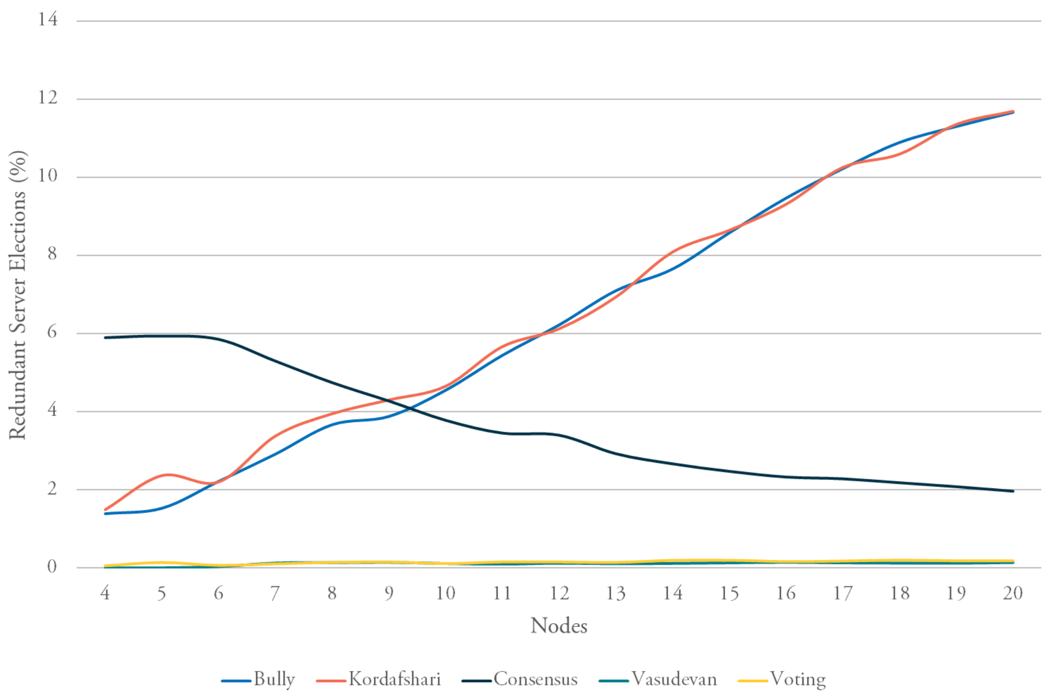


Fig. 7.5 *Percentage of redundant server elections by the election algorithms studied under the Manhattan Grid mobility model. Data on Table A.4.*

of 15.80% for the Kordafshari algorithm, with a total of 18.73% of redundant elections for a network of twenty nodes.

Table 7.2 *Increase of the percentage of redundant elections for the Bully and Kordafshari algorithms for each mobility model, from a network of four nodes to a network of twenty nodes.*

	Manhattan Grid		Random Walk		RPGM	
	Bully	Kordafshari	Bully	Kordafshari	Bully	Kordafshari
4 nodes	1.39%	1.49%	1.89%	2.22%	2.55%	2.93%
20 nodes	11.66%	11.69%	16.05%	16.44%	17.41%	18.73%
δn	10.27%	10.20%	14.16%	14.22%	14.86%	15.80%

As the number of nodes in the network increases, the Consensus algorithm increases its reliability. This is, the Consensus algorithm tends to generate less redundant server elections in a large group of nodes. For networks of 20 nodes, Consensus generates 1.96% redundant server elections with the Manhattan Grid model (Fig. 7.5), 1.84% with the Random Walk model (Fig. 7.6) and 1.68% with the RPGM model (Fig. 7.7). However, these results are still higher than those provided by the Vasudevan and Voting algorithms, which, in conclusion, present the best results concerning to avoid redundant server elections.

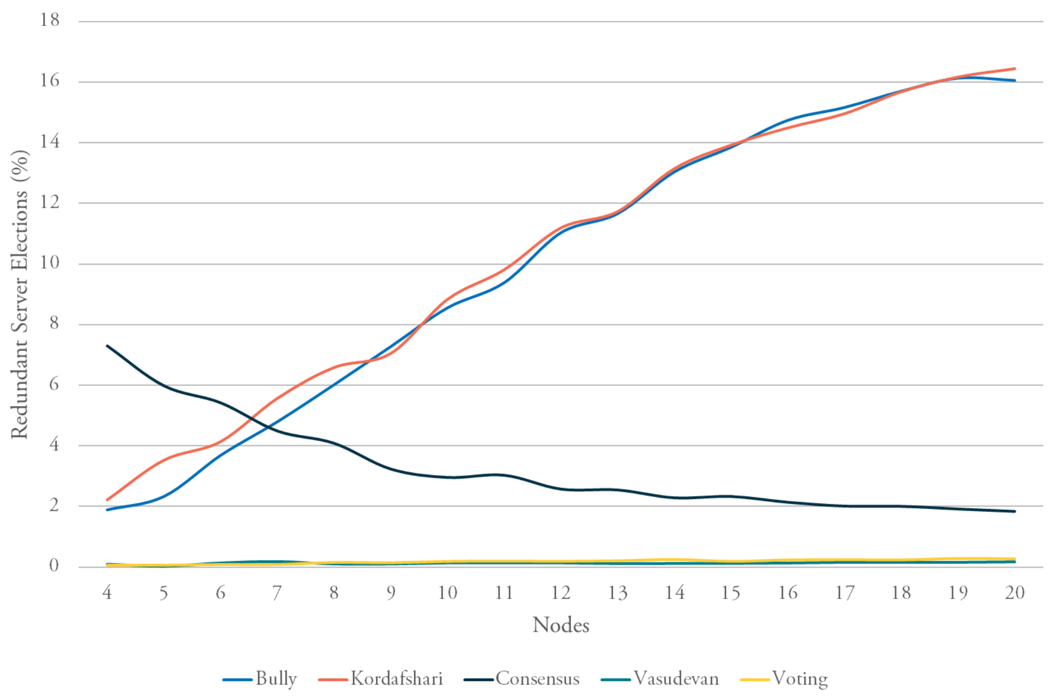


Fig. 7.6 *Percentage of redundant server elections by the election algorithms studied under the Random Walk mobility model. Data on Table A.5.*

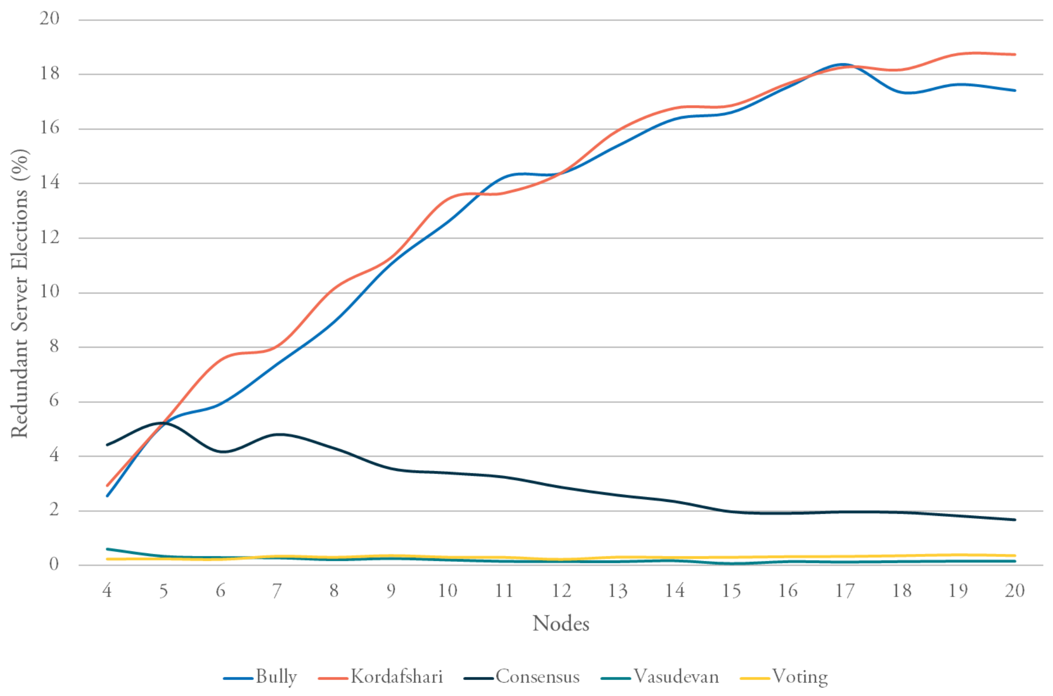


Fig. 7.7 *Percentage of redundant server elections by the election algorithms studied under the RPGM mobility model. Data on Table A.6.*

7.3.2 Mean Time Between Failures (MTBF)

Fig. 7.8, Fig. 7.9 and Fig. 7.10 show the results obtained regarding the reliability provided by the election algorithms concerning MTBF (Section 2.5.3) for the Manhattan Grid mobility model, the Random Walk mobility model, and the RPGM mobility model, respectively. It should be noted that at least is necessary two failures happen to be able to obtain the MTBF. In the charts provided, if there is no data about one algorithm for a particular network size, it does not mean that MTBF is zero, but that the algorithm has no presented more than one failure during simulation.

From the results obtained, it can be highlighted that, although both Vasudevan and Voting provide excellent reliability, failures in Voting happen with more frequency.

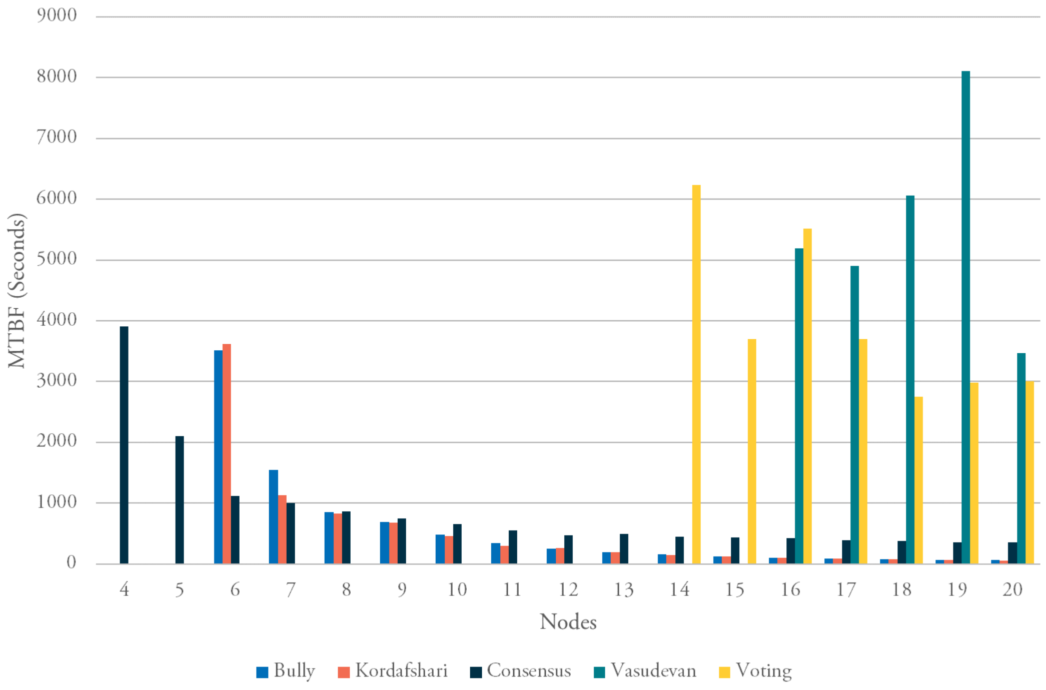


Fig. 7.8 *MTBF of the election algorithms studied under the Manhattan Grid mobility model. Data on Table A.7.*

7.4 Coordination Messages Usage

The coordination messages usage is one of the primary cost measures that has to be considered to measure the performance of the election algorithms. It is directly related to the use of energy

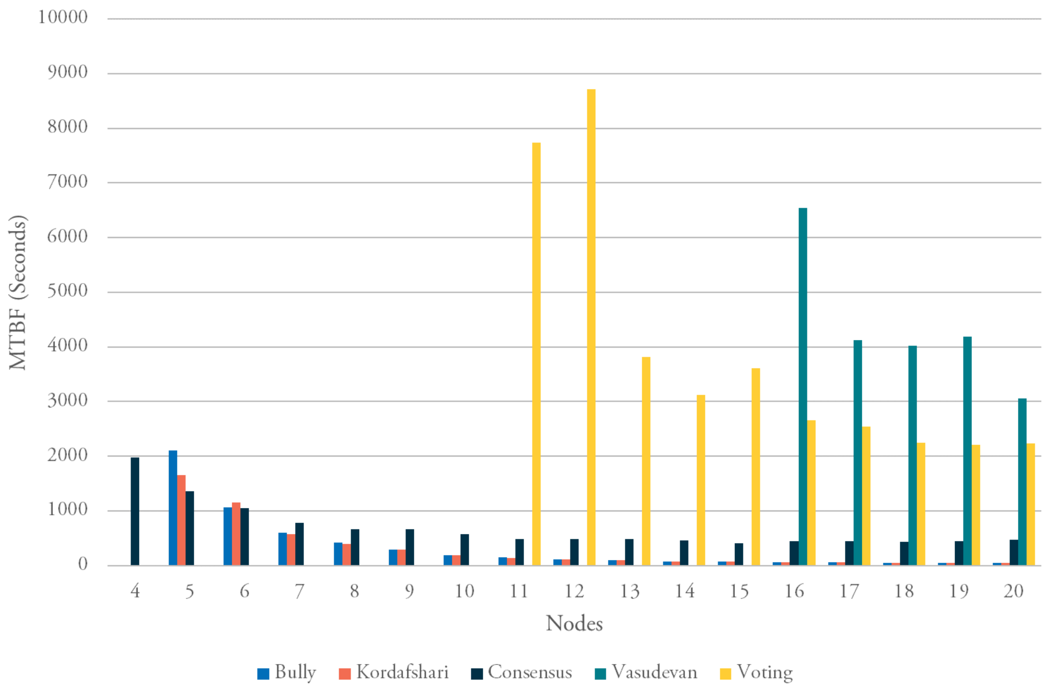


Fig. 7.9 *MTBF of the election algorithms studied under the Random Walk mobility model. Data on Table A.8.*

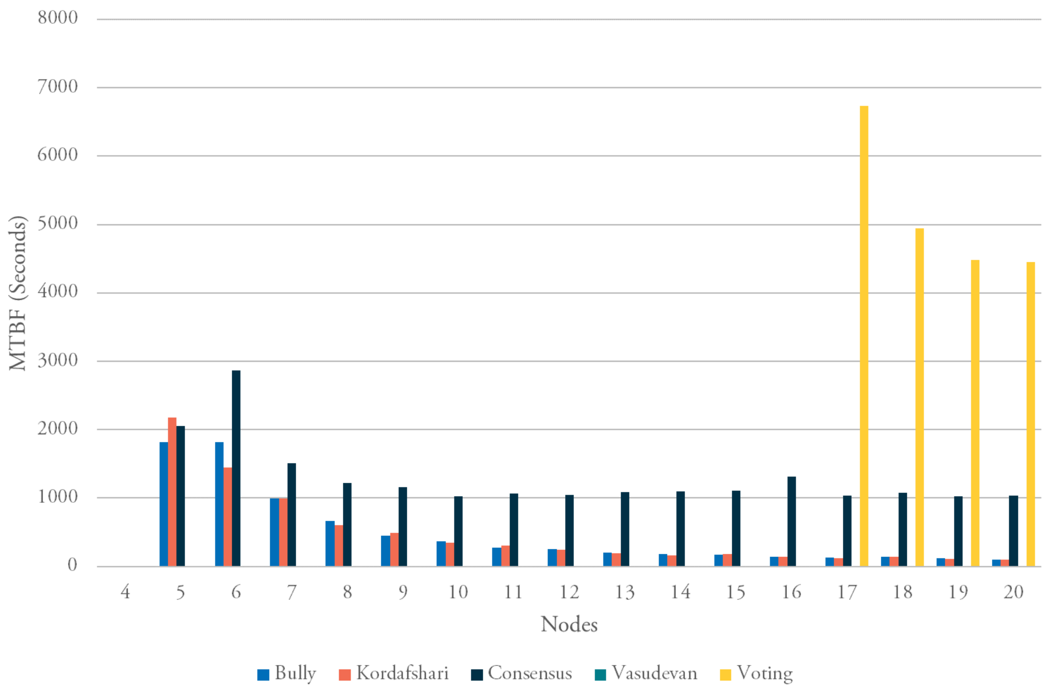


Fig. 7.10 *MTBF of the election algorithms studied under the RPGM mobility model. Data on Table A.9.*

(wireless communications and system processing) and of bandwidth. Moreover, it can be objectively measured, i.e., without underlying technology influencing the result.

This section is divided into two parts: first, the message complexity of the algorithms that was theoretically analysed in Section 4.6.3, is empirically checked for static and reliable networks in Section 7.4.1; and second, Section 7.4.2 presents the results obtained regarding to message complexity under mobile and unreliable networks, i.e., under the simulation conditions presented in Section 6.3.

7.4.1 Static and Reliable Networks

In this section, the algorithms have been applied in a static network on the ns-3 simulator. The objective is to empirically validate the theoretical message complexity obtained in Section 4.6.3. In this simulation, the algorithms only perform one election on a reliable network.

Adapted Bully Algorithm

In the case of the adapted version of the Bully algorithm (Fig. 7.11), the empirical message complexity is given by $0.9249n^2 - 9.6589n + 35.932$, following a quadratic complexity. The results obtained correspond with the theoretical limits defined.

Adapted Kordafshari Algorithm

In the case of the adapted version of the Kordafshari algorithm (Fig. 7.12), the empirical message complexity is given by $0.7713n^2 - 6.9986n + 24.466$, following a quadratic complexity within the theoretical limits defined, as well as in the case of the Bully algorithm. It should be noted that the Kordafshari algorithm only reduces slightly the number of coordination messages used in comparison with the Bully algorithm. This is because the Kordafshari algorithm provides that improvement in the worst case scenario. For the best case scenario, both algorithms present the same message complexity.

Vasudevan Algorithm

The message complexity of the Vasudevan algorithm (Fig. 7.13) in a static network is given by $1.1921n^2 + 22.476n - 84.98$, following a quadratic complexity. As noted, the simulated data is above of the theoretical worst case message complexity. This is because of the scenario simulated matches with the worst case of the Vasudevan algorithm, where all the nodes of the network are siblings. Additionally, messages produced by concurrent elections, that have

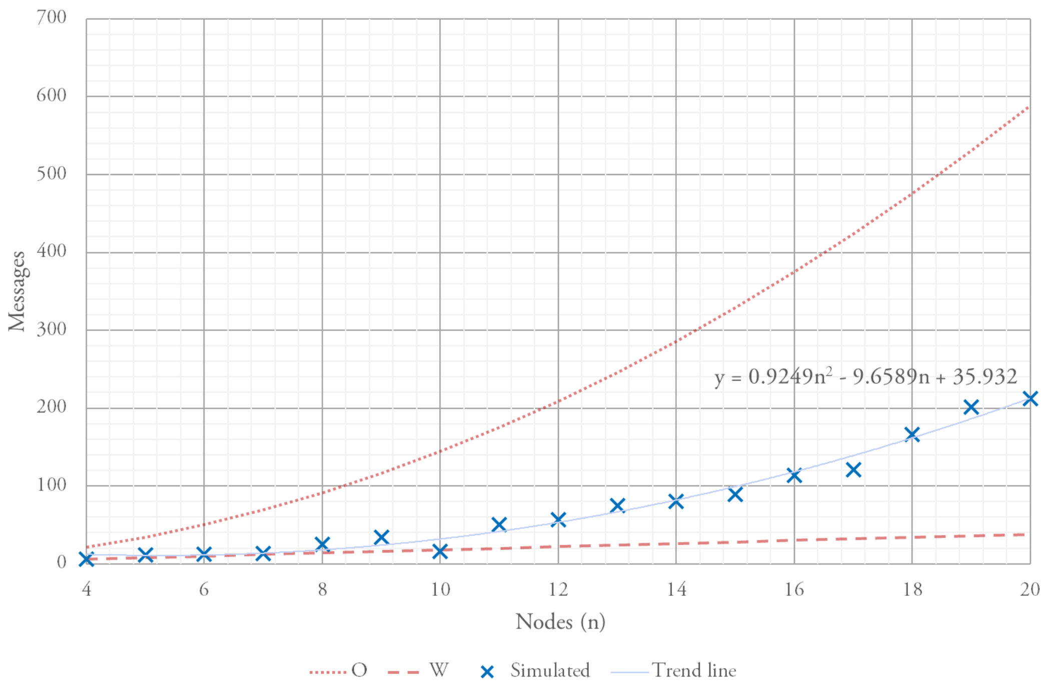


Fig. 7.11 *Comparison between theoretical and empirical message complexity of the Bully election algorithm.*

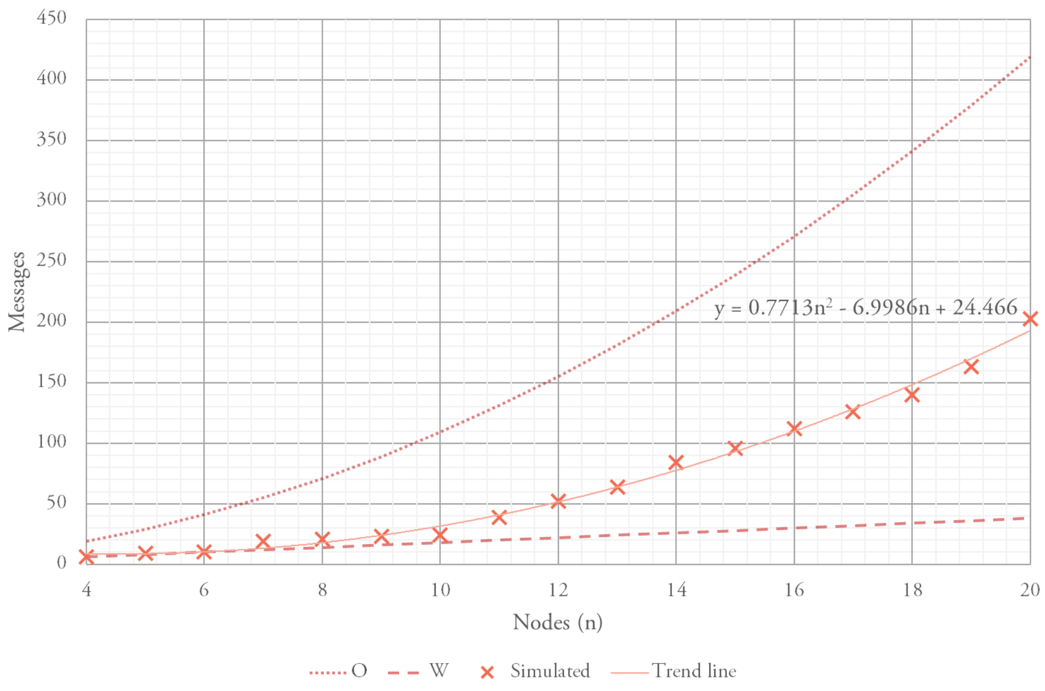


Fig. 7.12 *Comparison between theoretical and empirical message complexity of the Kordafshari election algorithm.*

not been taken into consideration in the theoretical study are the cause of the overcome of the theoretical limit.

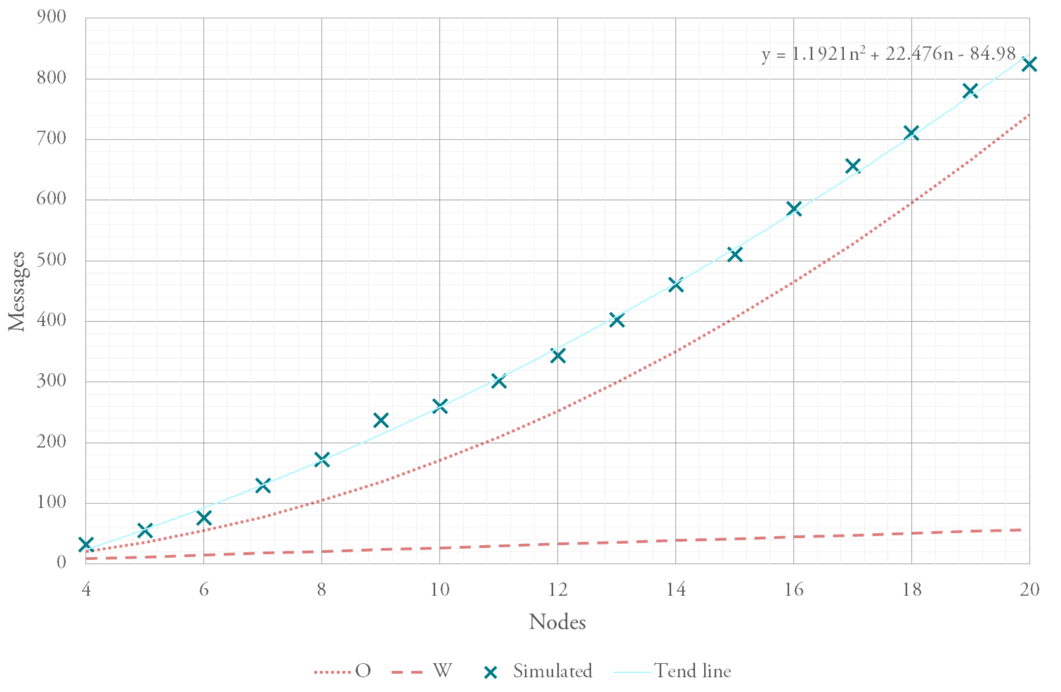


Fig. 7.13 *Comparison between theoretical and empirical message complexity of the Vasudevan election algorithm.*

Consensus Algorithm

The empirical message complexity of the Consensus algorithm is given by $1.0031n^2 + 2.0531n - 3.3932$ 7.14), following a quadratic complexity and slightly above the theoretical complexity calculated. This is because of, in the network simulated, the links between nodes are not reliable, and message loss can happen. Therefore the use of additional messages, as depicted in Fig. 4.8, in Section 4.6.1, is necessary to recover message loss in the election, avoiding concurrent servers or nodes without leader.

Voting Algorithm

The message complexity for the Voting algorithm in a static network is given by $0.9933n^2 + 1.4306n - 5.6285$ (Fig. 7.15), following a quadratic complexity. In this case, the empirical and theoretical message complexity correlate.

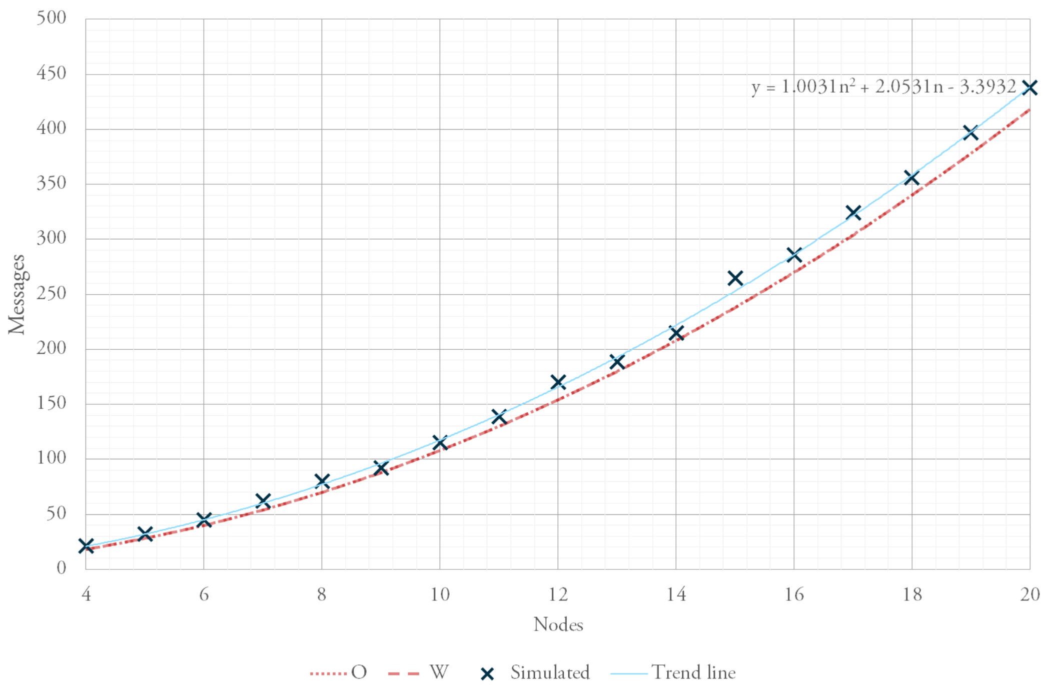


Fig. 7.14 *Comparison between theoretical and empirical message complexity of the Consensus election algorithm.*

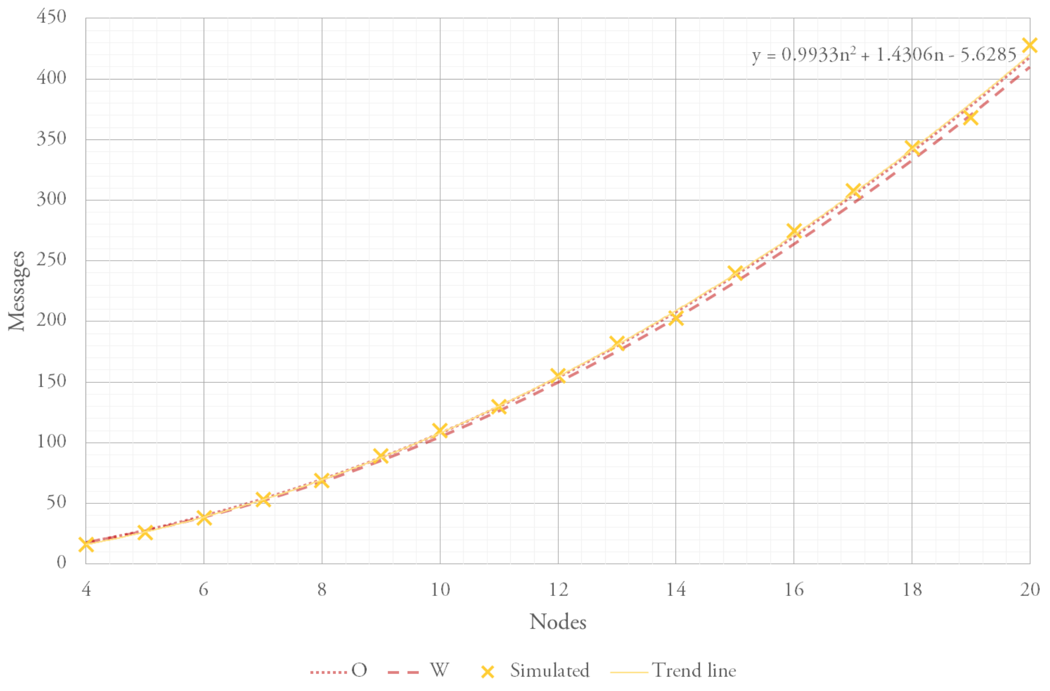


Fig. 7.15 *Comparison between theoretical and empirical message complexity of the Voting election algorithm.*

7.4.2 Mobile and Unreliable Networks

This section presents the results and analysis of coordination message usage in mobile and unreliable networks, under the simulation conditions presented in Section 6.3, Fig. 7.16, Fig. 7.17, and 7.18 show the results for the different election algorithms under the Manhattan Grid mobility model, the Random Walk mobility model, and the RPGM mobility model, respectively.

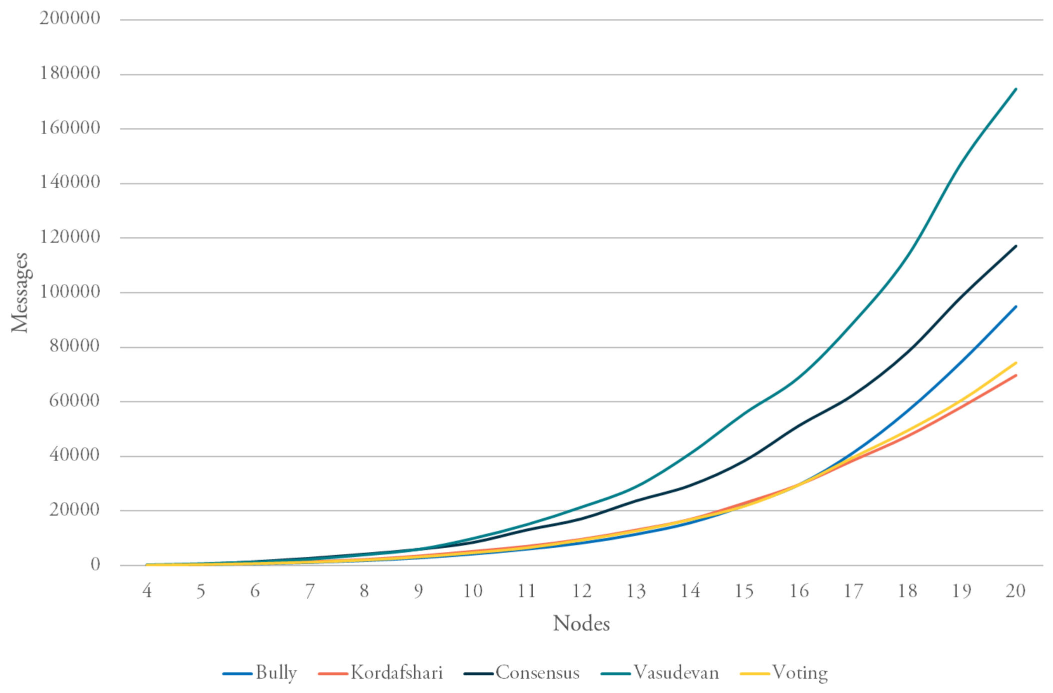


Fig. 7.16 *Coordination messages used during simulation (21600 seconds) by the different election algorithms under the Manhattan Grid mobility model. Data on Table A.10.*

The Kordafshari and Voting algorithms provide the best results regarding coordination messages usage in the three mobility models. For networks of 13/14 nodes or less, the Bully algorithm provides the same good results that the Kordafshari and Voting algorithms as well. However, for larger networks (13/14 nodes or more) the use of coordination messages by the Bully algorithm is significantly increased, even overtaken the Consensus algorithm in the Random Walk and the RPGM mobility models. This proves that the modifications introduced by Kordafshari in the traditional Bully algorithm to reduce the number of messages and avoid redundant elections are effective.

The Vasudevan algorithm provides the worst results regarding message usage. The use of coordination messages of this algorithm is profoundly affected by the nodes participating in the election and by the kind of network topology on which the algorithm is operating. This

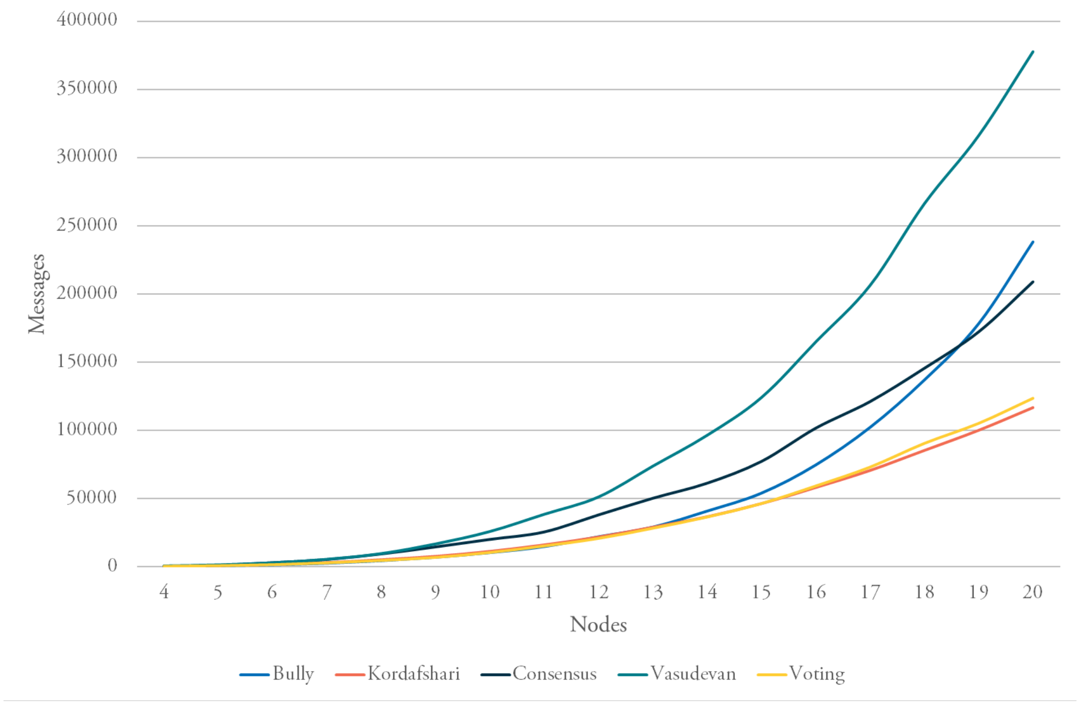


Fig. 7.17 *Coordination messages used during simulation (21600 seconds) by the different election algorithms under the Random Walk mobility model. Data on Table A.11.*

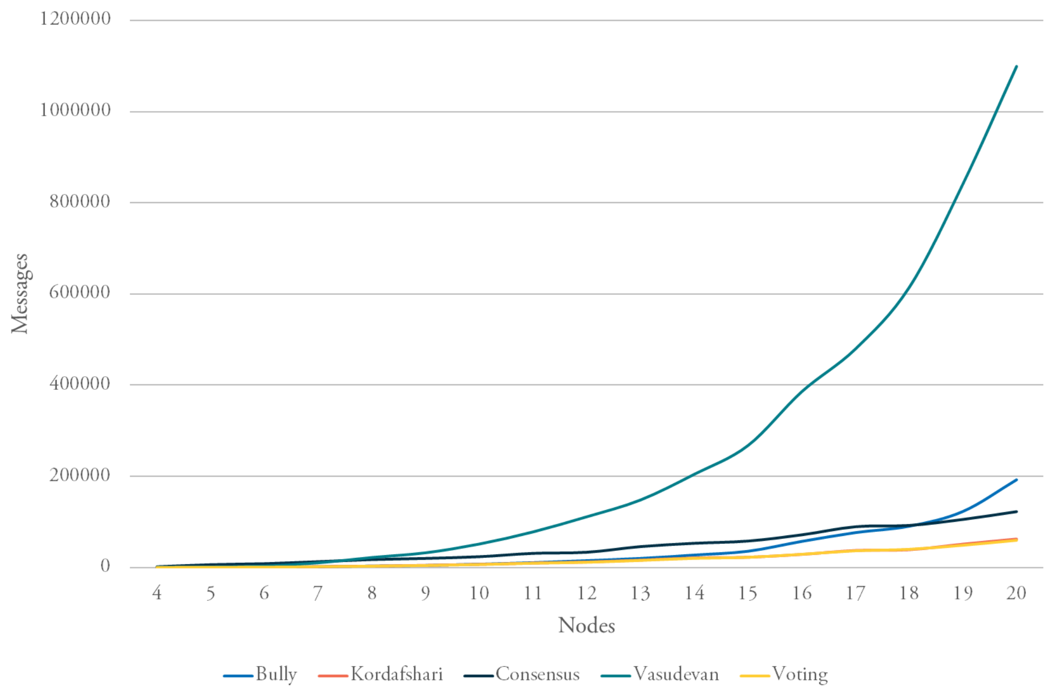


Fig. 7.18 *Coordination messages used during simulation (21600 seconds) by the different election algorithms under the RPGM mobility model. Data on Table A.12.*

can be seen in Fig. 7.18, where under the RPGM mobility model, the Vasudevan algorithm uses eighteen times more messages that the Voting algorithm in a network of twenty nodes. As exposed in Section 4.6.3, the worst case scenario regarding message complexity for the Vasudevan algorithm is caused by the shape of the network, being this a highly interconnected network (i.e., a group of nearby nodes). This is the case of the RPGM mobility model.

The high difference of use of coordination messages of Vasudevan is given by the way in which the concurrent elections are managed, although theoretically, the construction of the spanning tree is more expensive than the other algorithms approach. In this algorithm, the newest election has priority. Thus, the already in process elections are left behind. This approach would lead to inefficient use of messages, mainly if the interrupted elections were in a final stage of construction of the spanning tree.

7.5 Network lifetime

The network lifetime is defined by three measures related with the energy consumption of the nodes: Time until First Node Dies (TFND), Time until Half of the Nodes Die (THND) and Time until Last Node Dies (TLND). In this way, not only the energy consumption or lifetime of a node can be seen, but how the election algorithm affects the network lifetime.

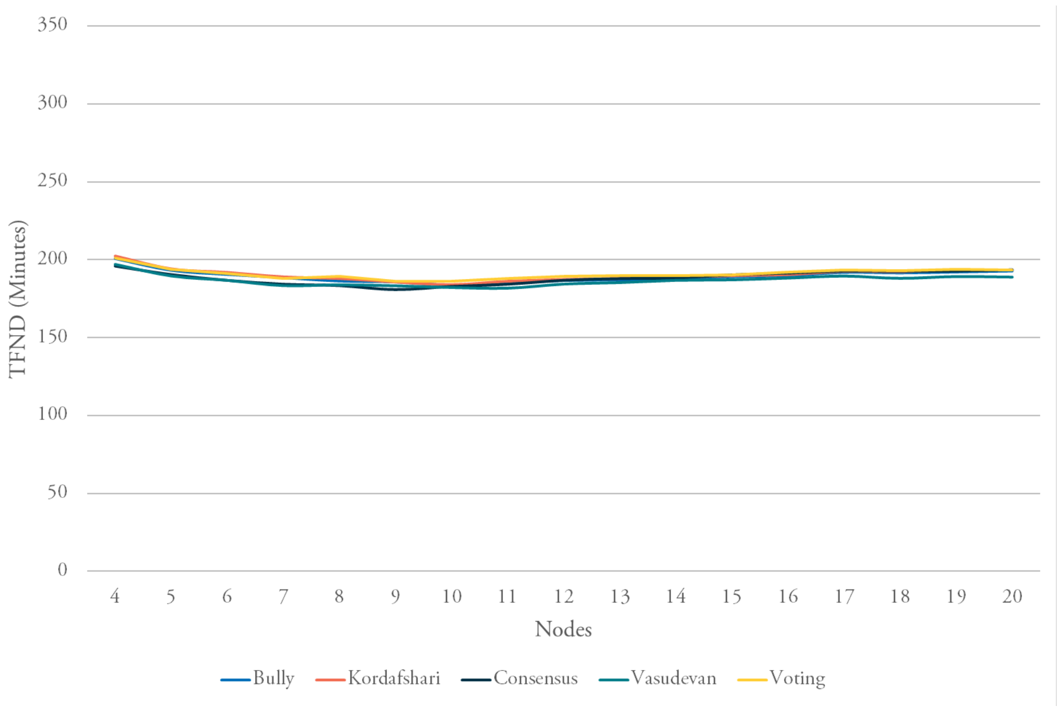


Fig. 7.19 *TFND under the Manhattan Grid mobility model. Data on Table A.13.*

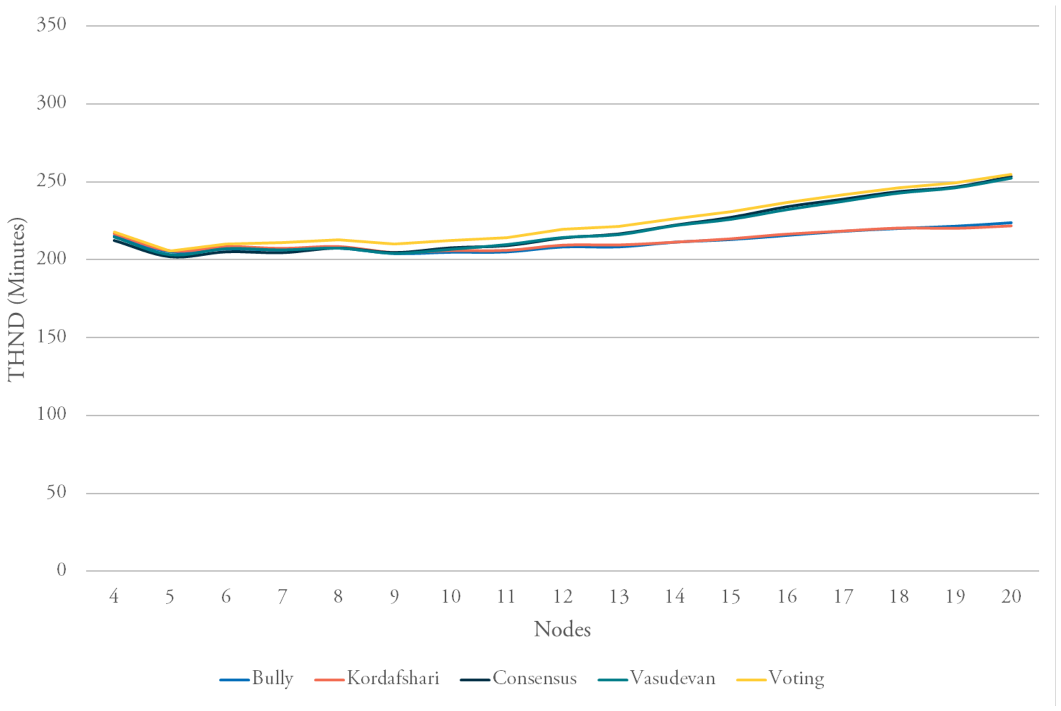


Fig. 7.20 *THND under the Manhattan Grid mobility model. Data on Table A.14.*

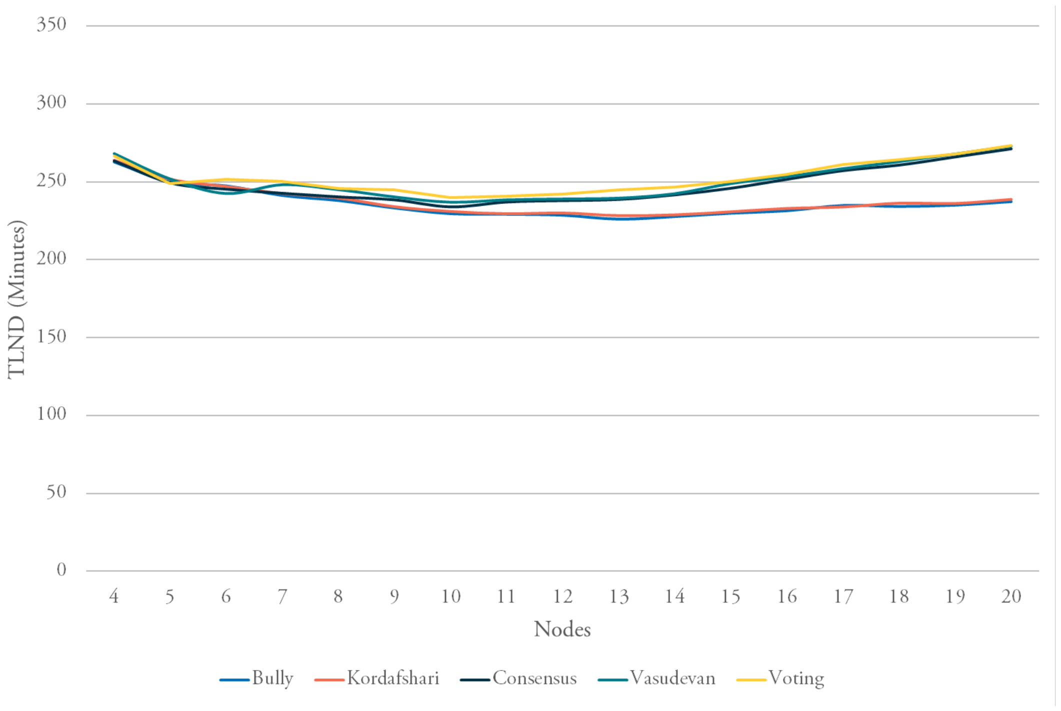


Fig. 7.21 *TLND under the Manhattan Grid mobility model. Data on Table A.15.*

For the Manhattan Grid mobility model, Fig. 7.19 shows the TFND provided by the different election algorithms, Fig. 7.20 shows the THND and Fig. 7.21 shows the TLND. As can be seen, there is no significant difference between the TFND provided by the different algorithms. However, in the THND and TLND, two groups of algorithms can be distinguished: (1) the Voting, Consensus and Vasudevan, which provide the longer network lifetime; and (2) the Bully and Kordafshari algorithms, which provide less network lifetime.

Specifically, between the best TLND in a network of twenty nodes, provided by the Voting algorithm (273.399 minutes), and the worst TLND, provided by the Bully algorithm (237.280 minutes), there is a difference of 36.120 minutes. This is owing to the less percentage of redundant elections that the Voting, Consensus and Vasudevan algorithms cause in comparison with the Bully and Kordafshari algorithms, which leads to a more efficient management of the energy. Also worthy of mention how the network lifetime is increased as the number of nodes is increased. This is because there are more nodes on which spreading the workload of hosting the active service replica, which emphasises how the architecture designed effectively manages the available resources of the network.

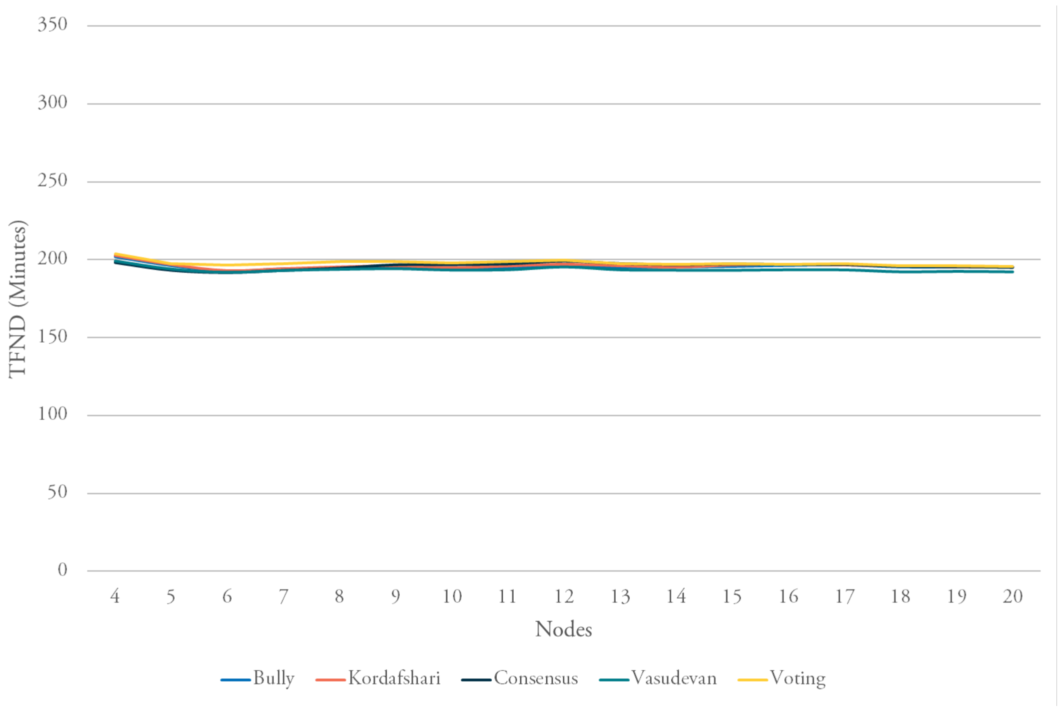


Fig. 7.22 *TFND under the Random Walk mobility model. Data on Table A.16.*

For the Random Walk mobility model, Fig. 7.22 shows the TFND provided by the different election algorithms, Fig. 7.23 shows the THND and Fig. 7.24 shows the TLND. As in the case

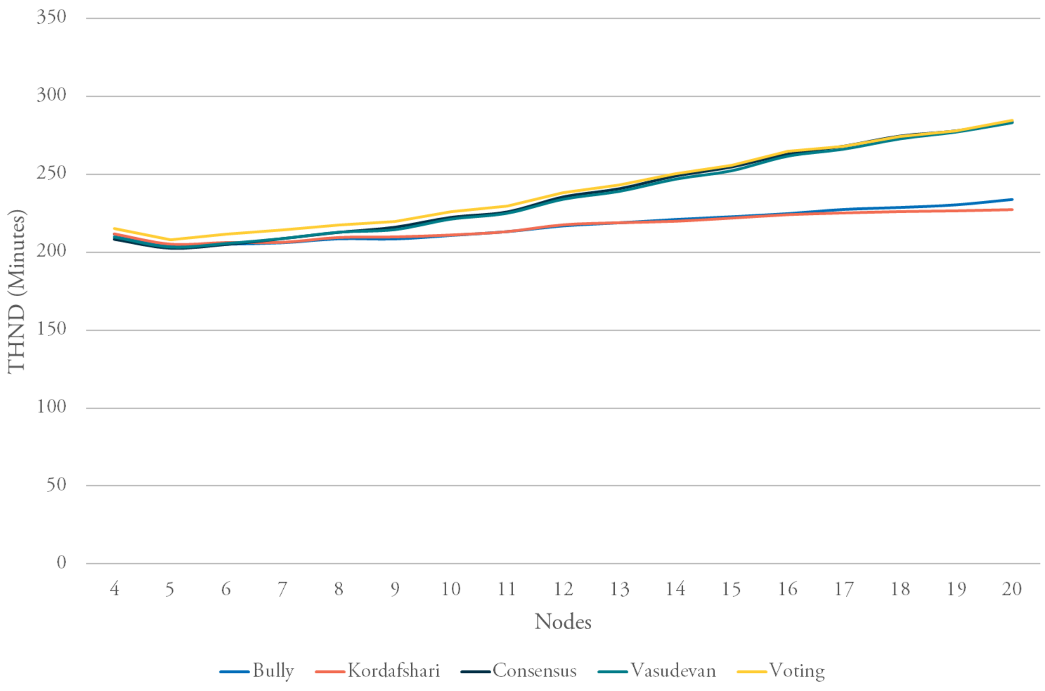


Fig. 7.23 *THND under the Random Walk mobility model. Data on Table A.17.*

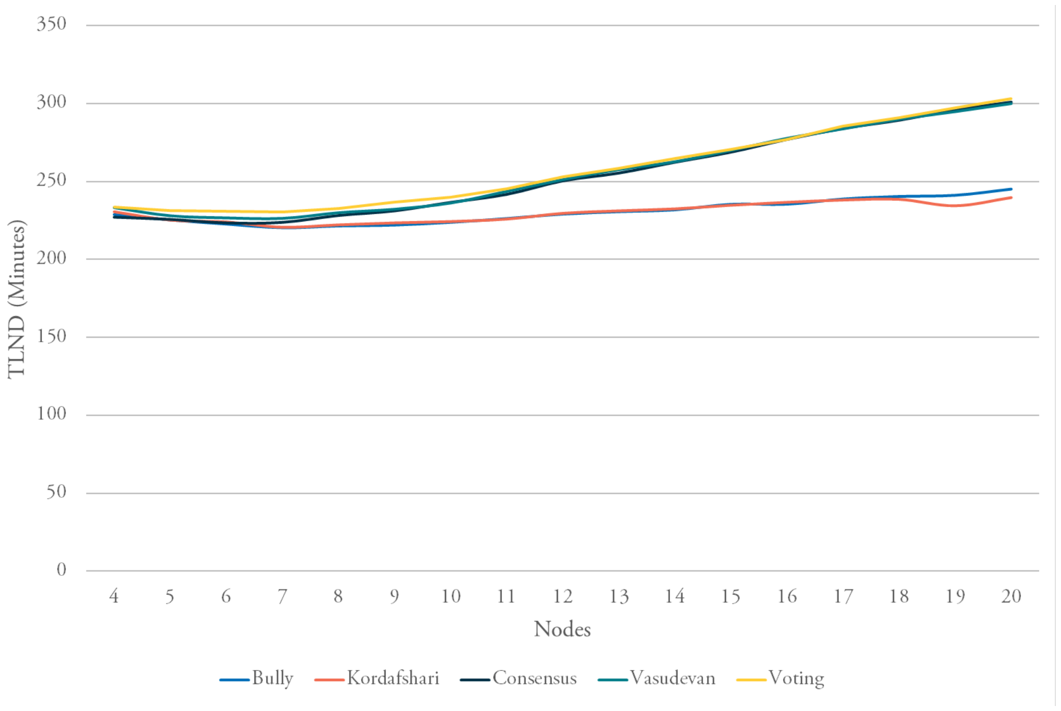


Fig. 7.24 *TLND under the Random Walk mobility model. Data on Table A.18.*

of the Manhattan Grid model, the TNFD does not show significant differences between the different election algorithms.

Similarly than in the Manhattan Grid, regarding THND and TLND, in the Random Walk mobility model, two groups of algorithms can be distinguished: (1) the Voting, Consensus and Vasudevan, which provide the longer network lifetime; and (2) the Bully and Kordafshari algorithms, which provide less network lifetime. However, this difference is more significant in the Random Walk mobility model than in the Manhattan Grid. Between the best TLND in a network of twenty nodes, provided by the Voting algorithm (303.245 minutes), and the worst TLND, provided by the Kordafshari algorithm (245.082 minutes), there is a difference of 63.592 minutes.

Additionally, it can be seen how under the Random Walk mobility model, in general, the lifetime of the network is larger than under the Manhattan Grid. Specifically, for a network of twenty nodes, in average for the five election algorithms, the TLND is 258.702 minutes under the Manhattan Grid, whereas under the RPGM is 277.7586 in average. This is owing to in the RPGM the network partitions are larger. Therefore fewer nodes acting as servers are necessary to maintain the availability of the services.

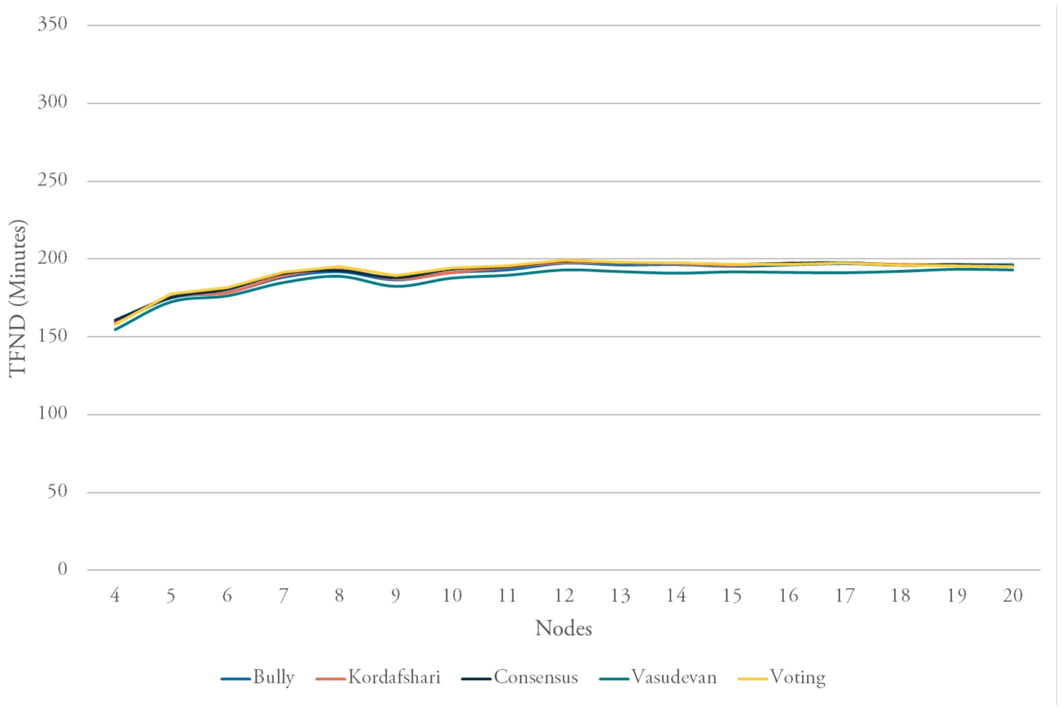


Fig. 7.25 *TFND under the RPGM mobility model. Data on Table A.19.*

Finally, of the RPGM mobility model, Fig. 7.25 shows the TFND provided by the different election algorithms, Fig. 7.26 shows the THND and Fig. 7.27 shows the TLND. As in the

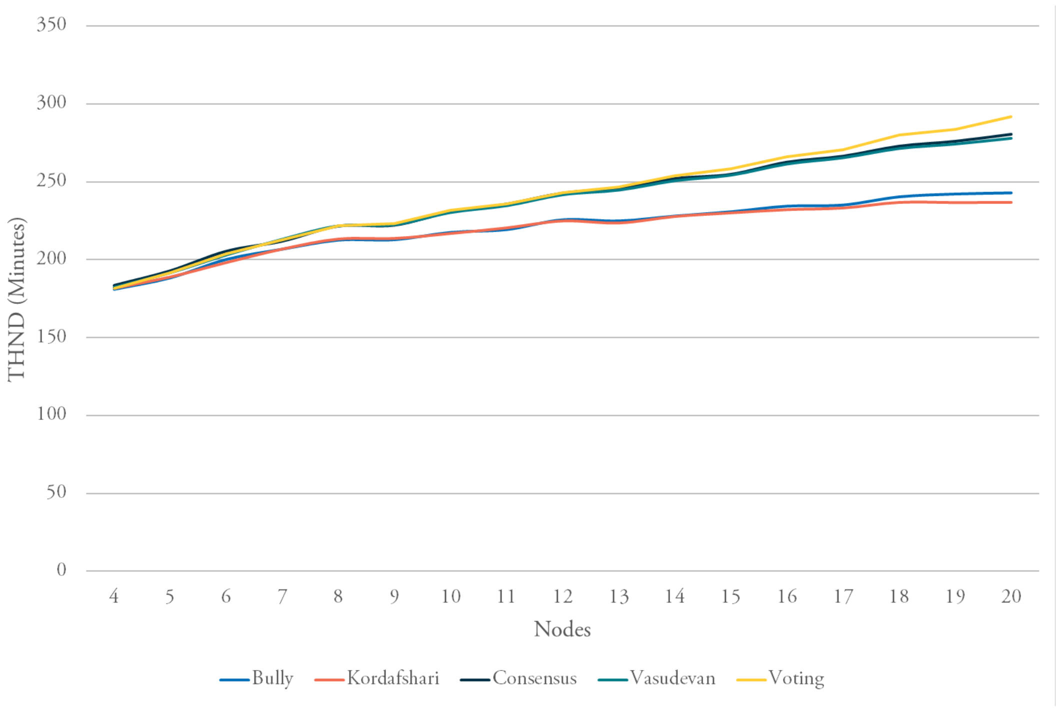


Fig. 7.26 *THND under the RPGM mobility model. Data on Table A.20.*

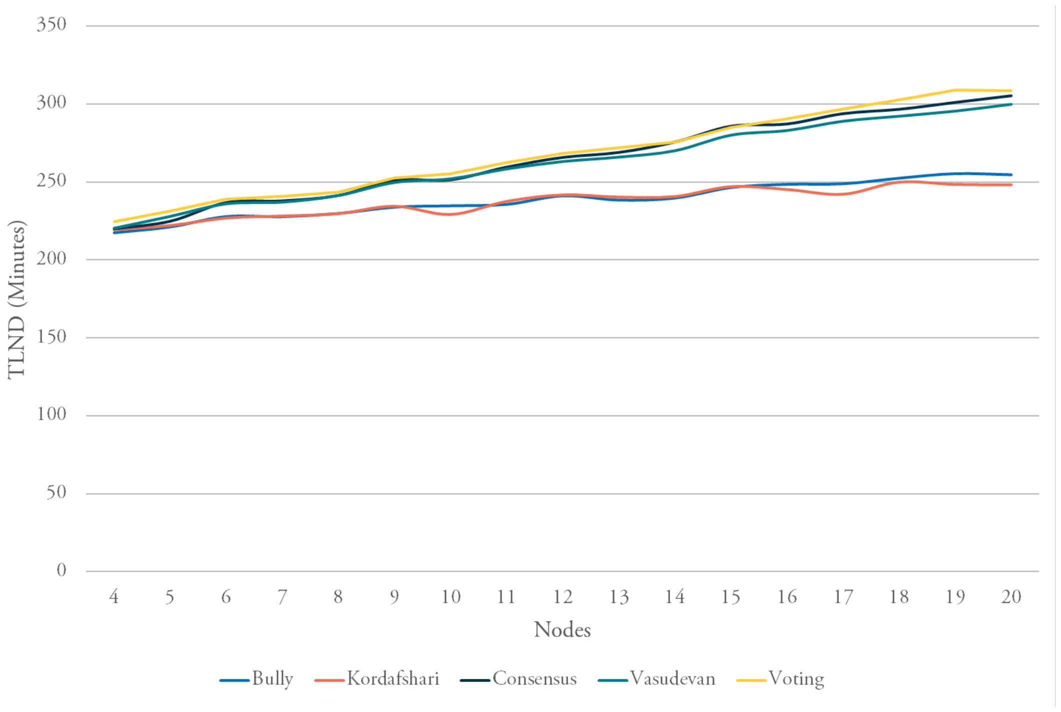


Fig. 7.27 *TLND under the RPGM mobility model. Data on Table A.21.*

previous cases, the TNFD does not show significant differences between the different election algorithms. However, it should be noted how the TFND in the RPGM is short than in the Manhattan Grid and the Random Walk for small network size (from 4 to 12 nodes). This is because, in the RPGM the time that a node spends in local status (i.e., without any reachable neighbour) is shorter than under the other mobility models. Hence, the time in which the service is running is greater, which implies a greater energy consumption.

Similarly than in previous mobility models, regarding THND and TLND, in the Random Walk mobility model, two groups of algorithms can be distinguished: (1) the Voting, Consensus and Vasudevan, which provide the longer network lifetime; and (2) the Bully and Kordafshari algorithms, which provide less network lifetime. The difference between the best TLND in a network of twenty nodes, provided by the Voting algorithm (308.629 minutes), and the worst TLND, provided by the Kordafshari algorithm (248.185 minutes), there is a difference of 60.444 minutes. It should be noted how the Vasudevan algorithm, which under other mobility models provided similar results that the Voting algorithm, under this mobility model, it provides lower TLND than the Voting algorithm. This it is related to the more significant use of coordination messages under the RPGM of this algorithm.

7.6 Efficiency

Oxford dictionary defines the term efficiency as *"the state or quality of being efficient"* [124], and efficient as *"achieving maximum productivity with minimum wasted effort or expense"* [123]. According to this general definition, in this section, the quality attributes (availability and reliability) and the cost measures taken (messages and TLND) are contrasted. It should be noted that messages and TLND measured are directly related to bandwidth and energy usage, respectively.

7.6.1 Service Availability vs Coordination Message Usage

To effectively compare the service availability against the Coordination Messages used, the Equation 7.1 has been used to calculate the efficiency.

$$eff(av, msgs) = \frac{av}{av_{max}} \cdot \left(1 - \frac{msgs}{msgs_{max}}\right) \tag{7.1}$$

where:

av	= represents the percentage of service availability provided by the algorithm
av_{max}	= represents the maximum service availability provided
msgs	= represents the Coordination Messages used by the algorithm
$msgs_{max}$	= represents the maximum Coordination Messages used

Therefore, $av_{max} = 99.76399158$, provided by the Vasudevan algorithm under the RPGM mobility model for a network of four nodes (Table A.3), and $msgs_{max} = 1099211$, used by the Vasudevan algorithm under the RPGM mobility model for a network of twenty nodes (Table A.12).

This normalization of the measures allows us to compare the efficiency of the algorithms not only on one mobility model but also between different mobility models in a range 0 to 1. The maximum efficiency (i.e., $eff(av, msgs) = 1$) is obtained when av reaches av_{max} and $msgs$ reaches 0. Note that therefore, the maximum efficiency is unreachable, as no election algorithm can operate using 0 Coordination Messages. However, the algorithm will be more efficient as $msgs$ tends to 0.

Fig. 7.28 shows the election algorithms efficiency (availability vs messages) under the Manhattan Grid mobility model, Fig. 7.29 under the Random Walk mobility model, and Fig. 7.30 under the RPGM mobility model.

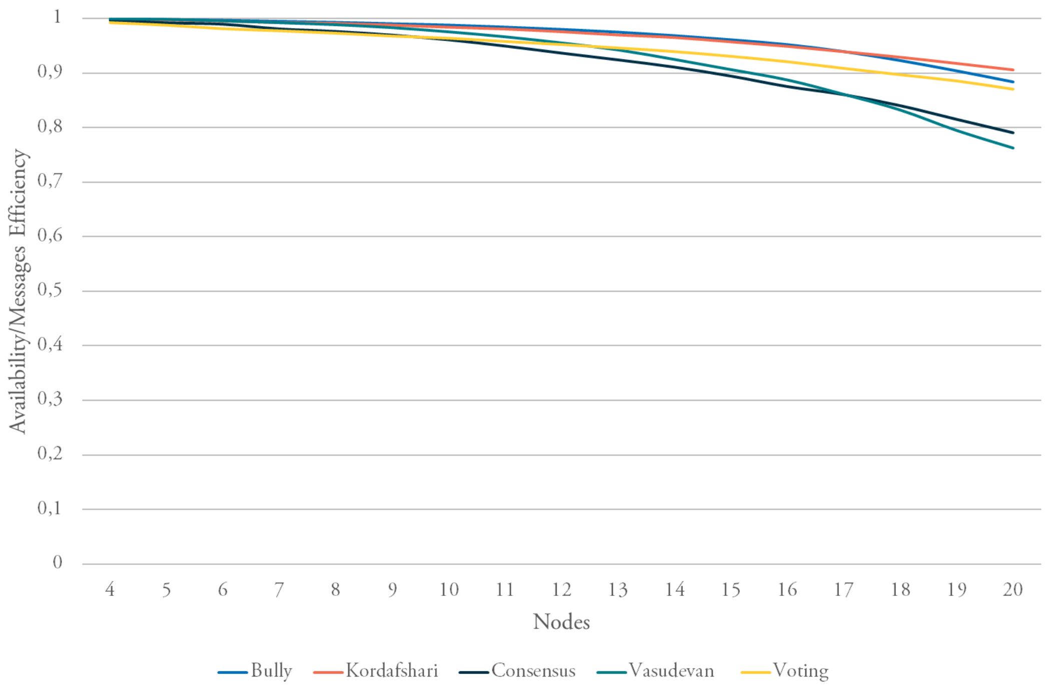


Fig. 7.28 *Election algorithms efficiency (availability vs messages) under the Manhattan Grid mobility model.*

Under the Manhattan Grid (Fig. 7.28) two groups of algorithms can be differentiated: (1) the Bully, Kordafshary and Voting, which provide the best efficiency, and (2) the Consensus and Vasudevan algorithms. Although Vasudevan provides better Service availability, it uses more Coordination Messages than the Consensus algorithm, which results in a worse efficiency in networks of 17 nodes or more.

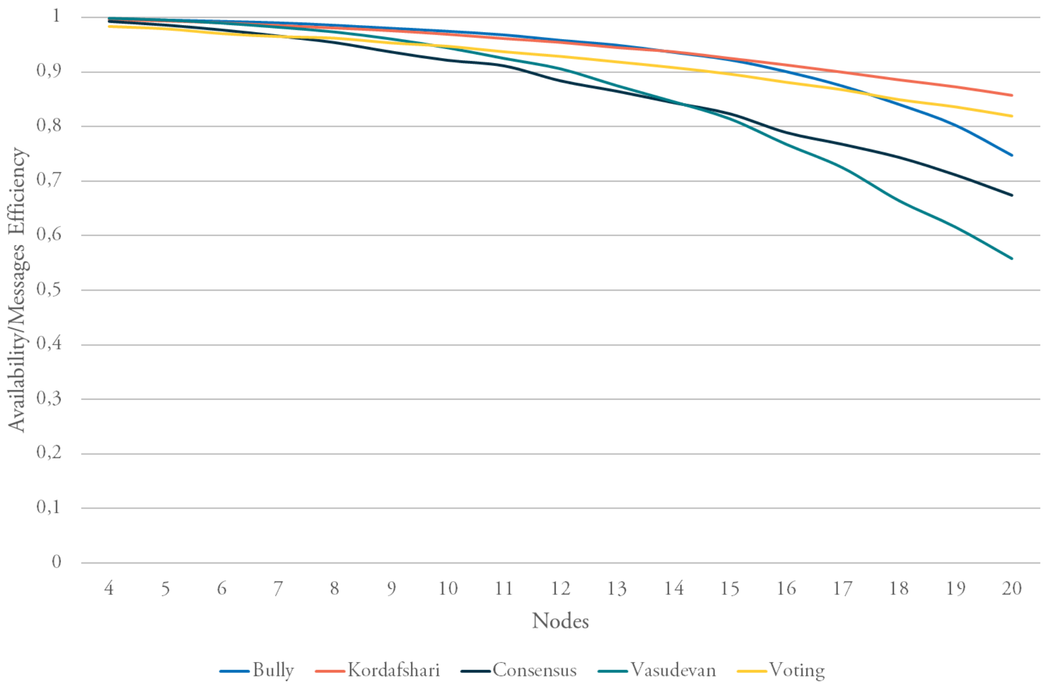


Fig. 7.29 *Election algorithms efficiency (availability vs messages) under the Random Walk mobility model.*

Under the Random Walk mobility model (Fig. 7.29), the results obtained are more uneven. It can be highlighted how the larger network partitions that can be found on the Random Walk have an impact on the efficiency provided by the Bully, Vasudevan and Consensus algorithms. It is especially relevant in the case of the Bully algorithm, which leads the results together Kodafshari algorithm until a network of fifteen nodes. For more extensive networks, the results of the Bully fall below the results of the Voting algorithm.

Under the RPGM mobility model (Fig. 7.30), the Kordafshari and Voting algorithms provide a similar efficiency, which is better than the one provided under the Manhattan Grid and the Random Walk mobility models. The bully algorithm provides similar results to networks of 15 nodes. For larger networks, the efficiency of this algorithm falls to the efficiency levels of the Consensus algorithm. The Vasudevan algorithm provides the worst results, both for service availability and Coordination Messages usage, under this mobility model.

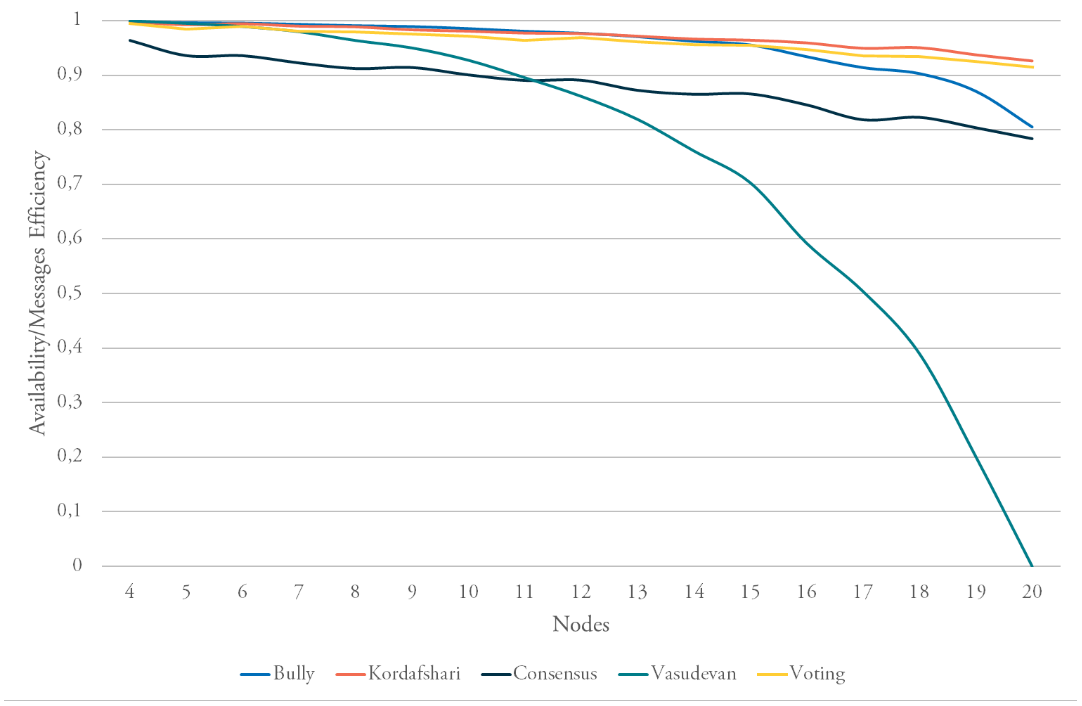


Fig. 7.30 *Election algorithms efficiency (availability vs messages) under the RPGM mobility model.*

7.6.2 Service Availability vs Network lifetime

To effectively compare the service availability against the Network Lifetime (specifically, TLND), Equation 7.2 has been used to calculate the efficiency.

$$eff(av, TLND) = \frac{av}{av_{max}} \cdot \frac{TLND}{TLND_{max}} \tag{7.2}$$

where:

av	= represents the percentage of service availability provided by the algorithm
av_{max}	= represents the maximum service availability provided
TLND	= represents the TLND provided by the algorithm
$TLND_{max}$	= represents the maximum TLND provided

Therefore, $av_{max} = 99.76399158$, provided by the Vasudevan algorithm under the RPGM mobility model for a network of four nodes (Table A.3), and $TLND_{max} = 309.0971667$, provided by the Voting algorithm under the RPGM mobility model for a network of nineteen nodes (Table A.21). The maximum efficiency (i.e., $eff(av, TLND) = 1$) is obtained when av reaches av_{max} and $TLND$ reaches $TLND_{max}$. Contrary to previous case, availability vs messages efficiency, the maximum efficiency is reachable.

Fig. 7.31 shows the election algorithm efficiency (availability vs TLND) under the Manhattan Grid mobility model, Fig. 7.32 under the Random Walk mobility model, and Fig. 7.33 under the RPGM mobility model.

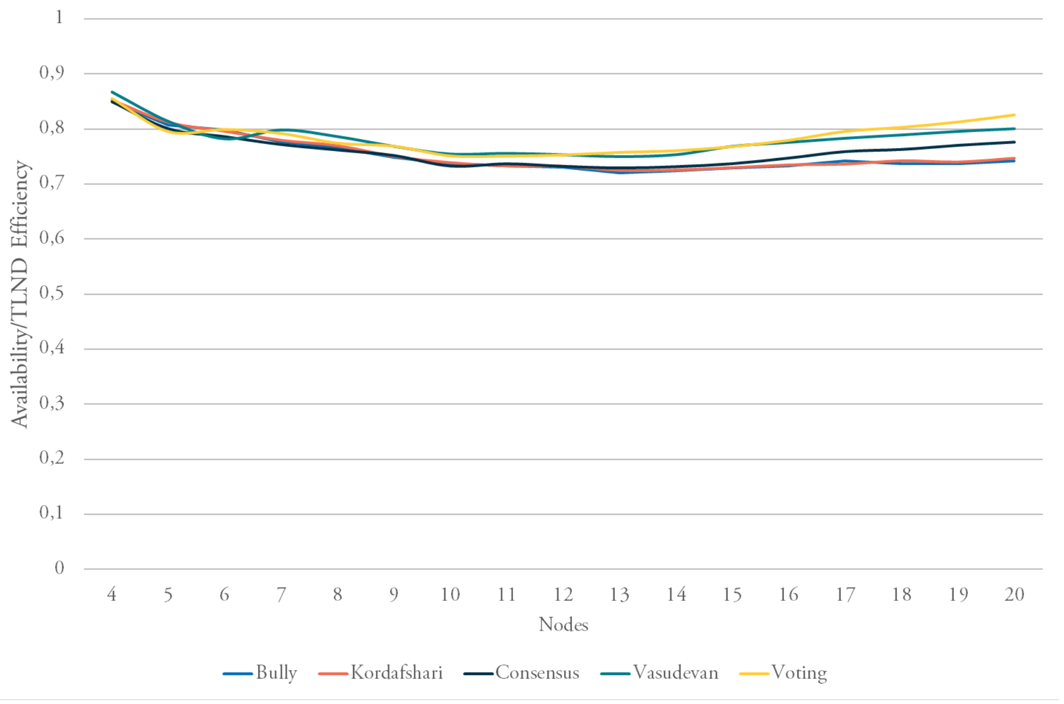


Fig. 7.31 *Election algorithms efficiency (availability vs TLND) under the Manhattan Grid mobility model.*

In the case of the Manhattan Grid (Fig. 7.31) the best efficiency regarding service availability and Network lifetime are provided by the Voting algorithm, followed by the Vasudevan and Consensus algorithm. The Kordafhari and Bully algorithms provide the lowest efficiency under the Manhattan Grid. This is because, although they provide better results regarding service availability, they provided the poorest results regarding Network Lifetime. It should be noted how the efficiency provided by the Consensus algorithm improves as the network size increases. Whereas for networks of thirteen nodes or less, the Consensus algorithm provides a similar efficiency than the Bully and Kordafshari algorithms, this efficiency is increased for networks of thirteen nodes or more. However, although the efficiency provided by the Consensus algorithm increased, it does not reach the efficiency provided by the Vasudevan or the Voting algorithms.

Under the Random Walk mobility model (Fig. 7.32), the Voting algorithm provides, with difference, the best efficiency. The Vasudevan and Consensus algorithms provide a similar efficiency, especially for networks from eighteen to twenty nodes. It should be noted, how Vasudevan efficiency is, for networks of twelve nodes or less, similar to that provided by the

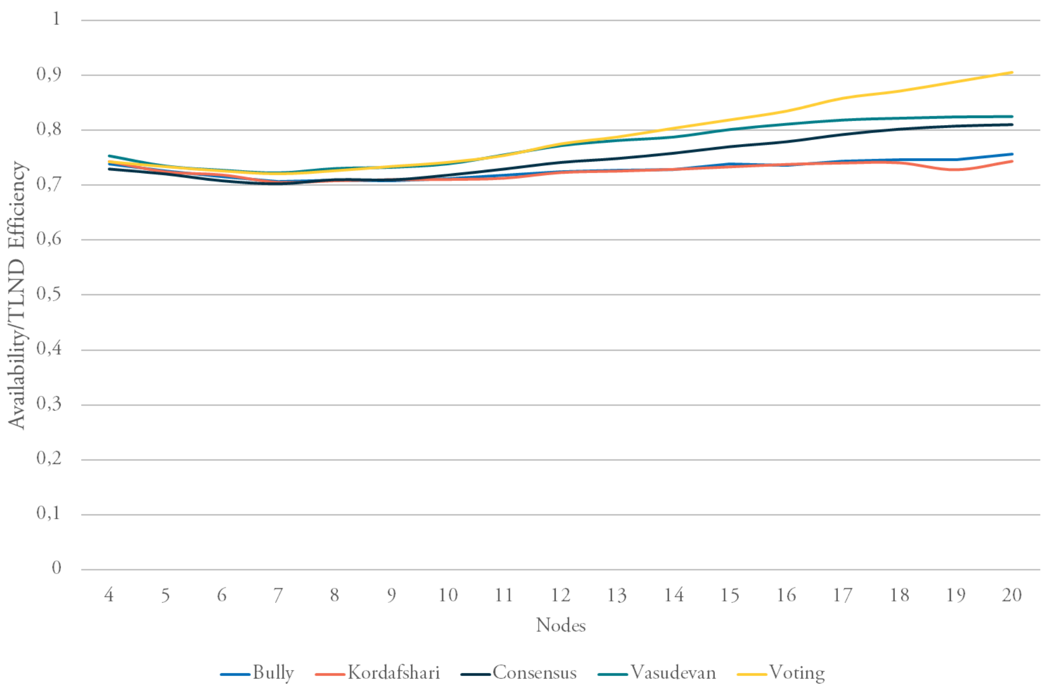


Fig. 7.32 *Election algorithms efficiency (availability vs TLND) under the Random Walk mobility model.*

Voting algorithm. After that, i.e., for networks of twelve nodes or more, the efficiency provided by Vasudevan reduces its growth rate in comparison with the Voting algorithm. The reverse occurs for the Consensus algorithm, which, for networks of nine nodes or less, provides an efficiency similar to that provided by the Bully and Kordafshari algorithms. After that, i.e., for networks of nine nodes or more, the growth rate of Consensus increases, nearly reaching the efficiency provided for the Vasudevan algorithm for a network of twenty nodes.

Finally, under the RPGM mobility model (Fig. 7.33), the Voting algorithm provides the best availability/TLND efficiency again. Under this mobility model, the efficiency of the Vasudevan algorithm falls to the levels of efficiency provided by the Kordafshari and Bully algorithms, providing results lower than those provided by the Consensus algorithm. It should be noted, how under this mobility model, the efficiency growth rate is ever-increasing, to the contrary that in the Manhattan Grid or the Random Walk. This is caused by the direct relationship between the network size and the size of its network partitions under the RPGM mobility model. This is, when a node is added to a network under the RPGM, it directly becomes part of a network partition in a continuous way. This fact does not have to be like that under the Manhattan Grid or the Random Walk mobility models, where the nodes have an individualist behaviour.

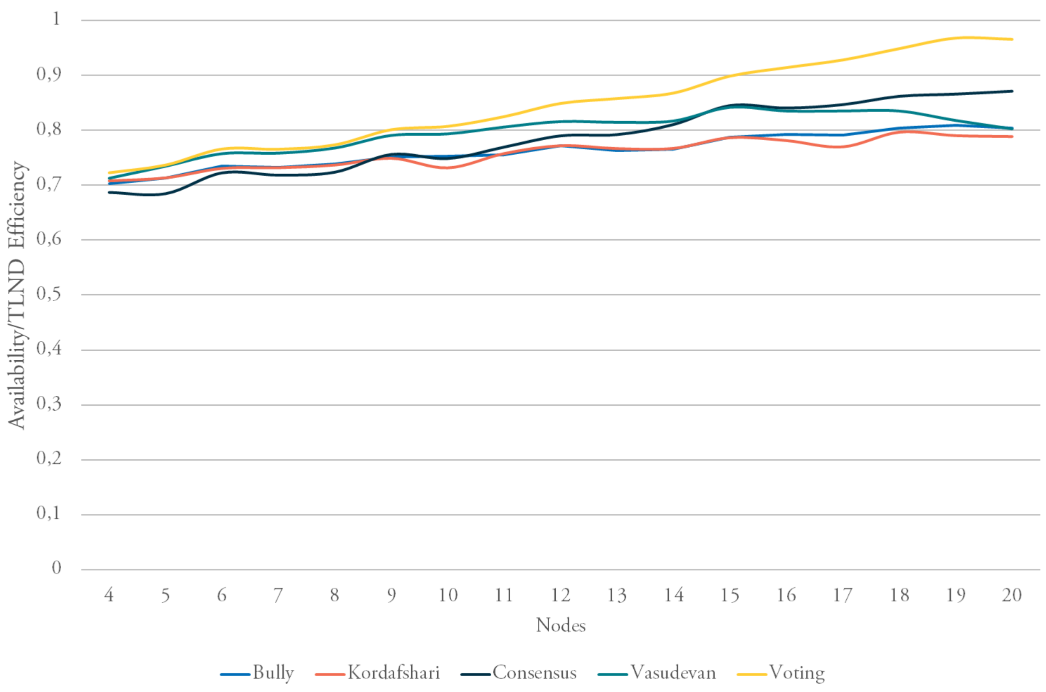


Fig. 7.33 *Election algorithms efficiency (availability vs TLND) under the RPGM mobility model.*

7.6.3 Algorithm Reliability vs Coordination Message Usage

The Equation 7.3 is used to calculate the Redundant Server Elections/Coordination Messages efficiency.

$$eff(rse, msgs) = (1 - \frac{rse}{rse_{max}}) \cdot (1 - \frac{msgs}{msgs_{max}}) \tag{7.3}$$

where:

rse	= represents the percentage of Redundant Server Elections produced by the algorithm
rse_{max}	= represents the maximum Redundant Server Elections produced
msgs	= represents the Coordination Messages used by the algorithm
$msgs_{max}$	= represents the maximum Coordination Messages used

Therefore, $rse_{max} = 18.7511428$, provided by the Kordafshari algorithm under the RPGM mobility model for a network of twenty nodes (Table A.6), $andmsgs_{max} = 1099211$, used by the Vasudevan algorithm under the RPGM mobility model for a network of twenty nodes (Table A.12).

The maximum efficiency (i.e., $eff(rse, msgs) = 1$) is obtained when rse reaches 0 and $msgs$ reaches 0. Note that therefore, as in the case of service availability/Message efficiency, the maximum efficiency is unreachable, as no election algorithm can operate using 0 Coordination Messages. However, the algorithm will be more efficient as *msgs* tends to 0.

Fig. 7.34 shows the election algorithm efficiency (Redundant Server Elections vs Coordination Messages) under the Manhattan Grid mobility model, Fig. 7.35 under the Random Walk mobility model, and Fig. 7.36 under the RPGM mobility model.

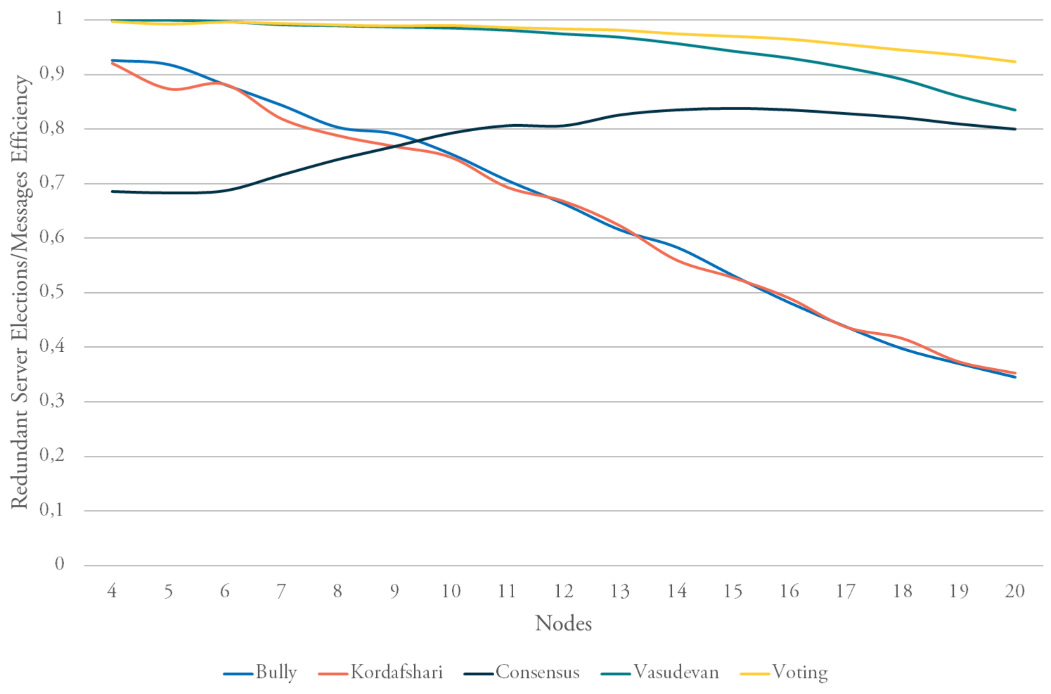


Fig. 7.34 *Election algorithms efficiency (Redundant Server Elections vs Coordination Messages) under the Manhattan Grid mobility model.*

In the case of the Manhattan Grid mobility model (Fig. 7.34), it can be seen how, although the Bully and Kordafshary algorithms provide the worst results. This is particularly relevant in the case of the Kordafhary algorithm, as it uses similar Coordination Messages than the Voting algorithm. However, the Voting algorithm provides, with the same use of Coordination Messages, higher reliability, which results in higher efficiency. In the case of the Vasudevan algorithm, its efficiency is affected by the high use of Coordination Message, despite its high reliability. Finally, the Consensus algorithm, once again, confirms to be more efficient in large networks. Nevertheless, in this case, it reaches its maximum efficiency (0.83) in networks of fourteen nodes.

Under the Random Walk mobility model (Fig. 7.35) the algorithms, in general, show a lower efficiency. In particular, the Consensus algorithm reaches its best efficiency (0.83) in networks of twelve nodes. The remaining algorithms show a decreasing efficiency as network

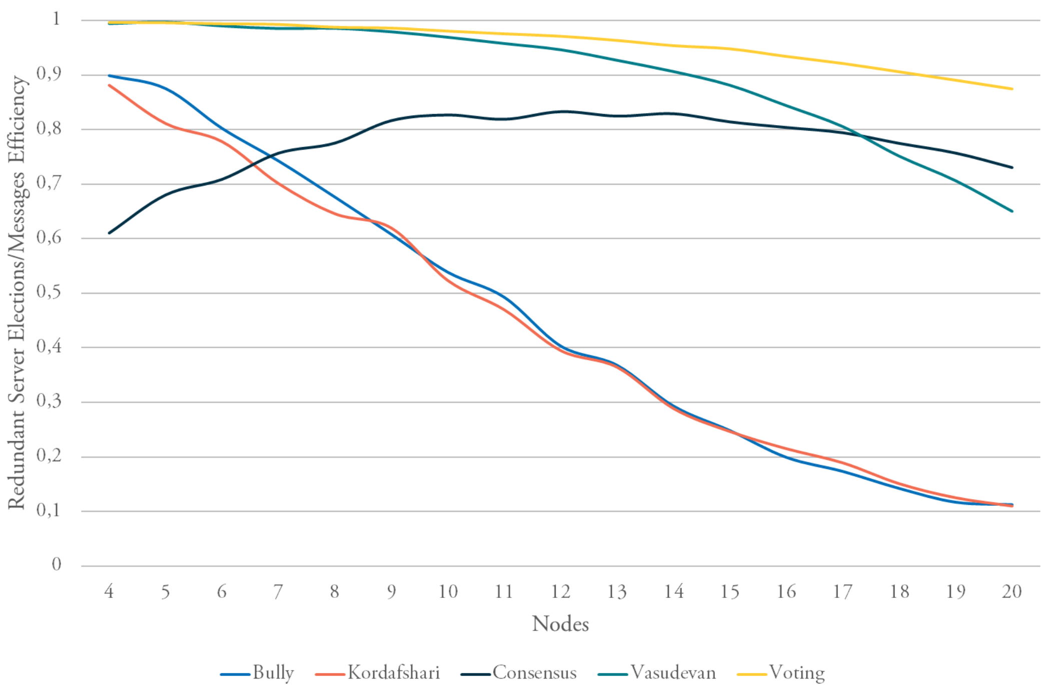


Fig. 7.35 *Election algorithms efficiency (Redundant Server Elections vs Coordination Messages) under the Random Walk mobility model.*

size increases. Whereas the Voting algorithm shows the best efficiency, Kordafshari and Bully show the worst results. Finally, although the Vasudevan algorithm provides an excellent efficiency in small size networks (nine nodes or less), its decreasing rate grows exponentially as size networks increases, falling below the results of the Consensus algorithm for networks of seventeen nodes or more.

It is particularly interesting the case of the RPGM mobility model (Fig. 7.36). In this case, two groups of algorithms can be distinguished: (1) that made up of the Voting and Consensus algorithms; and (2) that composed by the Vasudevan, Bully, and Kordafshari algorithms. Additionally, to provide the best results, the first group of algorithms is also the less affected by the particularities of this mobility model. The Voting and Consensus algorithms show better efficiency under this mobility model for networks of twenty nodes, (0.93 and 0.81, respectively) than under the Random Walk mobility model (0.87 and 0.73, respectively). However, on the contrary, the second group of algorithms, show their worst results, falling the three of them below an efficiency of 0.1 for a network of twenty nodes.

This fact underscores the better efficiency in general of the Voting algorithm in terms of reliability, against Coordination Messages used, and in particular of the Consensus and Voting algorithm when there are large groups of nodes involved in an election process.

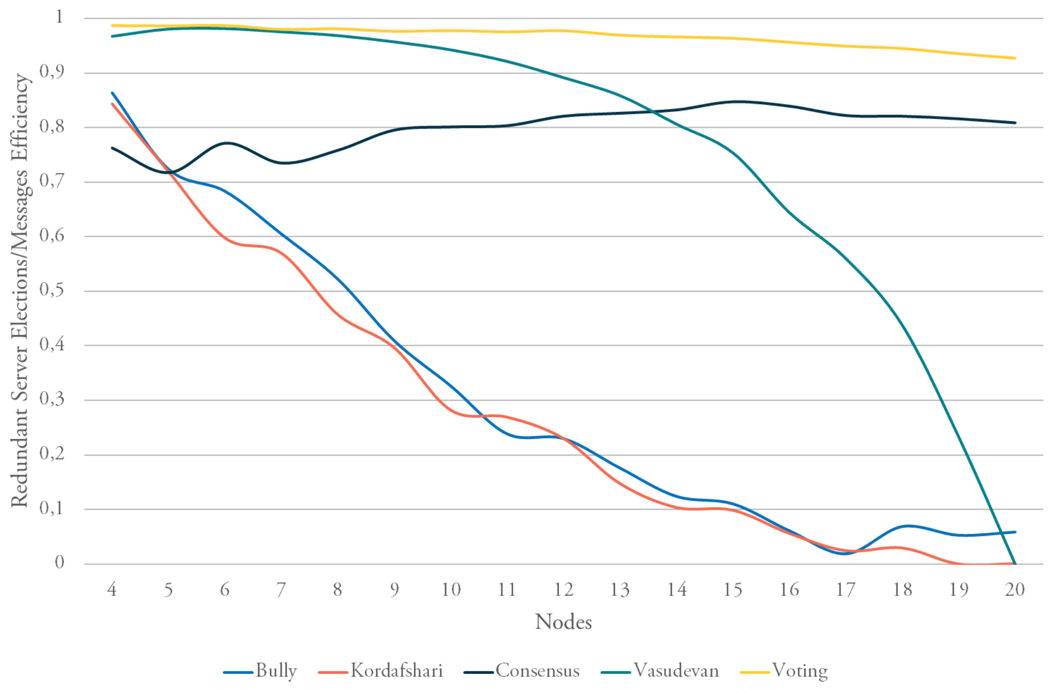


Fig. 7.36 *Election algorithms efficiency (Redundant Server Elections vs Coordination Messages) under the RPGM mobility model.*

7.6.4 Algorithm Reliability vs Network lifetime

The following Equation 7.4 has been used to compare the Redundant Server Elections (i.e., reliability) against the Network Lifetime (TLND).

$$eff(rse, TLND) = (1 - \frac{rse}{rse_{max}}) \cdot \frac{TLND}{TLND_{max}} \tag{7.4}$$

rse	= represents the percentage of Redundant Server Elections produced by the algorithm
rse_{max}	= represents the maximum Redundant Server Elections produced
TLND	= represents the TLND provided by the algorithm
$TLND_{max}$	= represents the maximum TLND provided

Therefore, $rse_{max} = 18.7511428$, provided by the Kordafshari algorithm under the RPGM mobility model for a network of twenty nodes (Table A.6), and $TLND_{max} = 309.0971667$, provided by the Voting algorithm under the RPGM mobility model for a network of nineteen nodes (Table A.21). The maximum efficiency, which is reachable, (i.e., $eff(rse, TLND) = 1$) is obtained when rse reaches 0 and $TLND$ reaches $TLND_{max}$.

Fig. 7.37 shows the election algorithm efficiency (Redundant Server Elections vs TLND) under the Manhattan Grid mobility model, Fig. 7.38 under the Random Walk mobility model, and Fig. 7.39 under the RPGM mobility model.

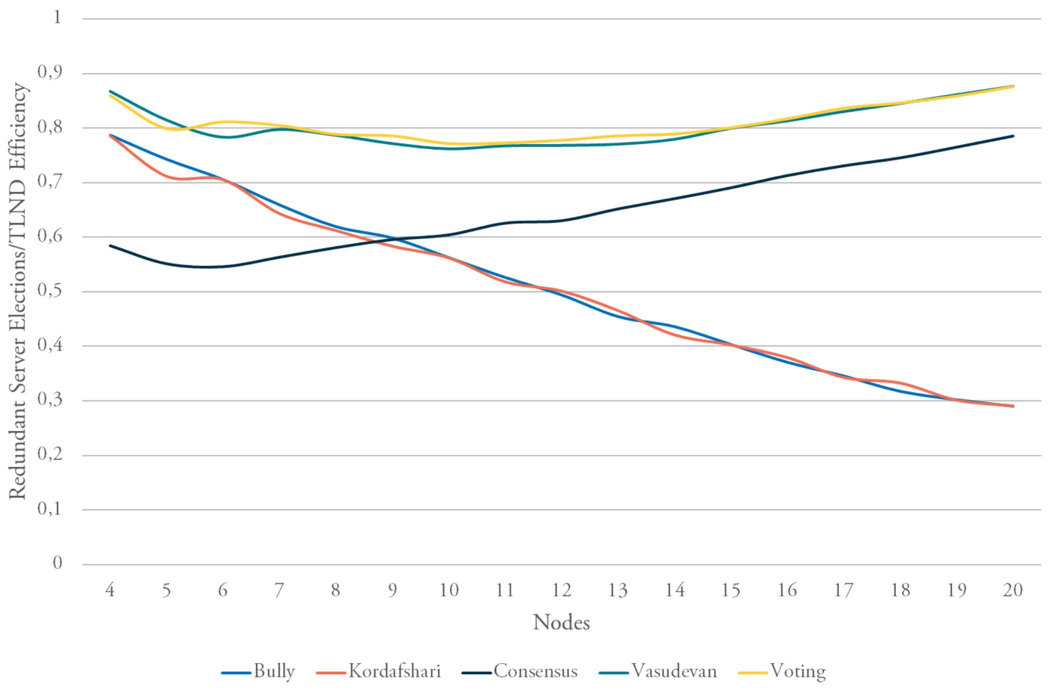


Fig. 7.37 *Election algorithms efficiency (Redundant Server Elections vs TLND) under the Manhattan Grid mobility model.*

In this case, the results obtained are similar for all mobility models. The Kordafshari and Bully present decreasing efficiency, whereas Consensus, Vasudevan, and Voting provide an increasing growth efficiency. In the case of the Consensus algorithm, this is because the algorithm increments its reliability (i.e., it reduces the number of redundant elections) as network size increases. In the case of the Voting and Vasudevan algorithms, the number of redundant server elections can be considered constant regarding network size, whereas the TLND is increased as network size increases since the workload can be distributed among more nodes. The Bully and Kordafshari algorithms provide the worst efficiency. This is because of the direct relationship between the number of servers and energy consumption. Thus, the redundant election of servers results in major energy consumption and a short TLND.

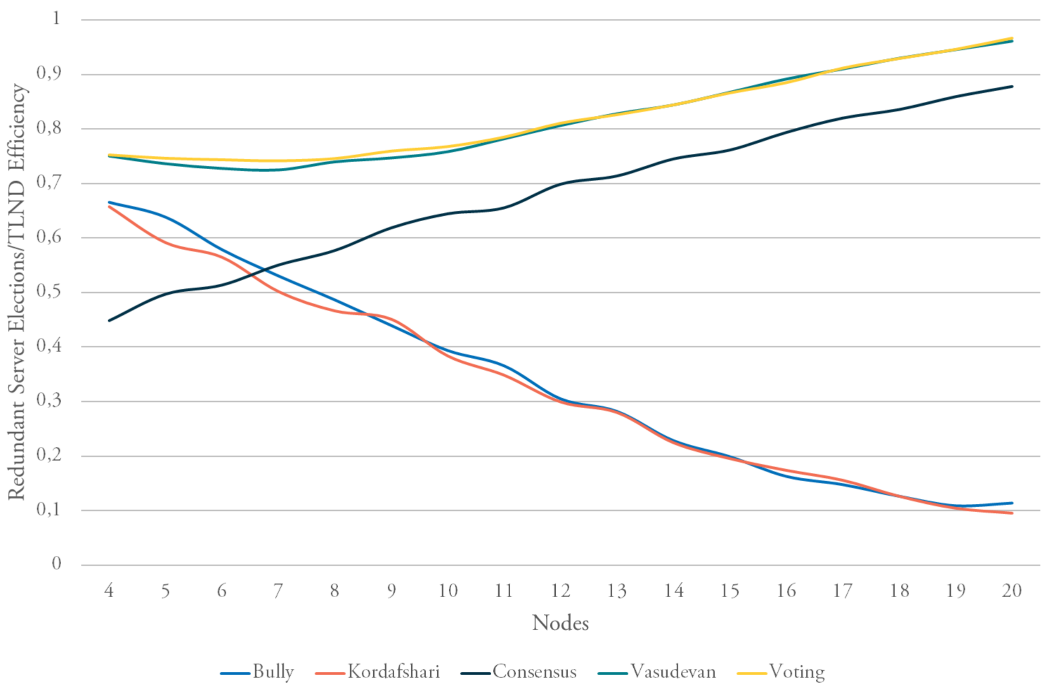


Fig. 7.38 *Election algorithms efficiency (Redundant Server Elections vs TLND) under the Random Walk mobility model.*

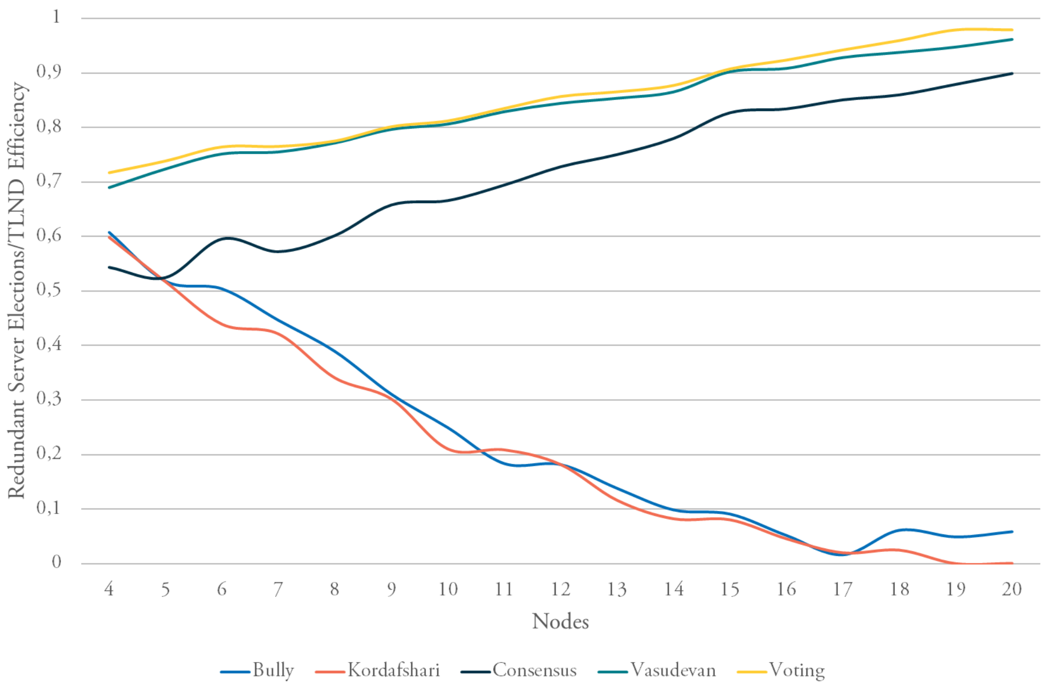


Fig. 7.39 *Election algorithms efficiency (Redundant Server Elections vs TLND) under the RPGM mobility model.*

Table 7.3 *Average of the service availability (av), Coordination Messages (msgs), TLND and Redundant Server Elections (rse) results obtained by the Bully, Kordafshari and Vasudevan election algorithms under the Manhattan Grid (MG), the Random Walk (RW) and the RPGM mobility models for network sizes ranged from 4 to 20 nodes. Additionally, Global average is shown.*

	Bully				Kordafshari				Vasudevan			
	av (%)	msgs	TLND	rse (%)	av (%)	msgs	TLND	rse (%)	av (%)	msgs	TLND	rse (%)
Average MG	98.35	21873.24	236.23	6.39	98.15	19241.12	237.35	6.53	96.41	45895.88	250.38	0.10
Average RW	97.52	53984.24	230.47	10.08	97.28	35931.24	230.07	10.32	93.76	104591.35	255.47	0.13
Average RPGM	98.70	39282.71	238.41	12.88	98.55	18852.00	237.08	13.41	94.02	255929.00	262.40	0.22
Global average	98.19	38380.06	235.04	9.78	97.99	24674.78	234.83	10.09	94.73	135472.08	256.08	0.15

Table 7.4 *Average of the service availability (av), Coordination Messages (msgs), TLND and Redundant Server Elections (rse) results obtained by the Consensus and Voting election algorithms under the Manhattan Grid (MG), the Random Walk (RW) and the RPGM mobility models for network sizes ranged from 4 to 20 nodes. Additionally, Global average is shown.*

	Consensus				Voting			
	av (%)	msgs	TLND	rse (%)	av (%)	msgs	TLND	rse (%)
Average MG	94.60	32522.82	248.30	3.62	95.87	19602.41	252.53	0.15
Average RW	91.43	62099.53	254.19	3.31	94.67	36821.29	257.68	0.18
Average RPGM	91.53	46735.24	264.84	3.07	97.47	18329.29	268.10	0.31
Global average	92.52	47119.20	255.78	3.33	96.00	24917.67	259.43	0.21

7.7 Discussion on Global Performance

In this section the average performance of the election algorithms studied is shown. In Table 7.3 and Table 7.4 it is shown the average of the service availability (av), Coordination Messages (msgs), TLND, and Redundant Server Elections (rse) results, obtained by the Bully, Kordafshari and Vasudevan election algorithms (Table 7.3), and the Consensus and Voting election algorithms (Table 7.4), under the Manhattan Grid (MG), the Random Walk (RW) and the RPGM mobility models for network sizes ranged from 4 to 20 nodes. Additionally, the Global average is shown.

To effectively compare the global results obtained by the algorithms, the data has been normalized in the following way:

- *service availability*: $norm(av) = \frac{av}{av_{max}} \in [0,1]$, where $av_{max} = 98.19$, provided by the Bully algorithm (Table 7.3).
- *Coordination Messages*: $norm(msgs) = \frac{msgs_{min}}{msgs} \in [0,1]$, where $msgs_{min} = 24674.78$, used by the Kordafshari algorithm (Table 7.3).

- *TLND*: $norm(TLND) = \frac{TLND}{TLND_{max}} \in [0,1]$, where $TLND_{max} = 259.43$, provided by the Voting algorithm (Table 7.4).
- *Redundant Server Elections*: $norm(rse) = \frac{rse_{min}}{rse} \in [0,1]$, where $rse_{min} = 0.15$, provided by the Vasudevan algorithm (Table 7.3).

It should be noted that, whereas service availability and TLND are measures to maximize, Coordination Messages and Redundant Server Elections are measured are intended to minimize.

The global results obtained after the normalization can be seen in the radar charts of Fig. 7.40 for the Bully algorithm, Fig. 7.41 for the Kordafshari algorithm, Fig. 7.42 for the Consensus algorithm, Fig. 7.43 for the Vasudevan algorithm, and Fig. 7.44 for the Voting algorithm.

As it can be seen, both the Bully, Kordafshari and Consensus algorithms provide poor global reliability in comparison with the Vasudevan and Voting algorithms. This lack reliability influences the Network lifetime, as it has been pointed out both redundant server elections and coordination messages produce an impact in the energy with reduces the Network lifetime, specifically the TLND. Hence, the Voting and Vasudevan algorithms provide a larger TLND than the Bully, Kordafshari, and Consensus.

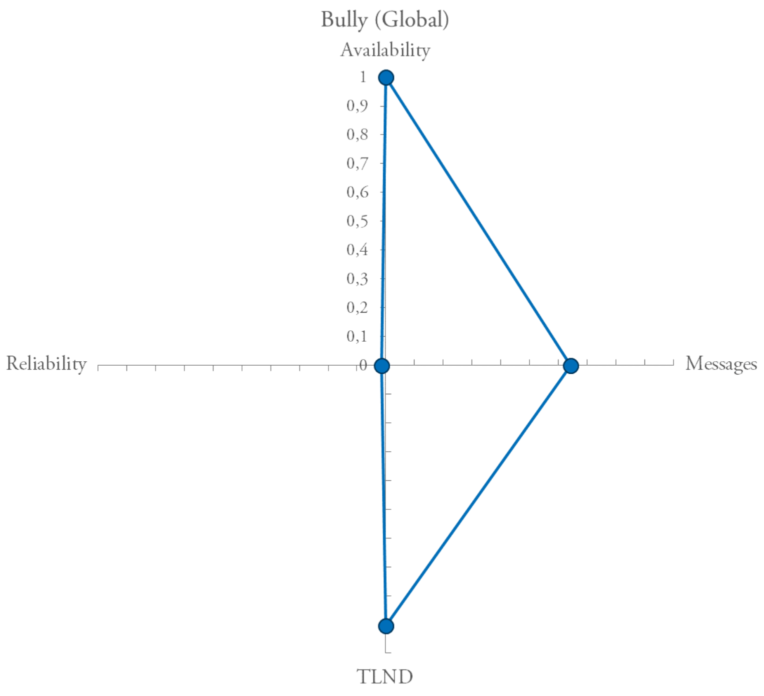


Fig. 7.40 *Global performance of the Bully algorithm (average results over the three mobility models).*

Otherwise, the Bully and Kordafshari algorithms provide better service availability that the others algorithms. The Kordafshari algorithm arises as a modification of the Bully algorithm that intents to reduce the number of Coordination Messages. Therefore, they show similar characteristics, and as it can be seen that Kordafshari effectively reduces the number of Coordination Messages used, without affecting the service availability provided. According to the global results obtained, Kordafshari algorithm reduces the number of Coordination Messages from 38380.06, used by the Bully (Table 7.3), to 24674.78. This represents a reduction of 35.71%.

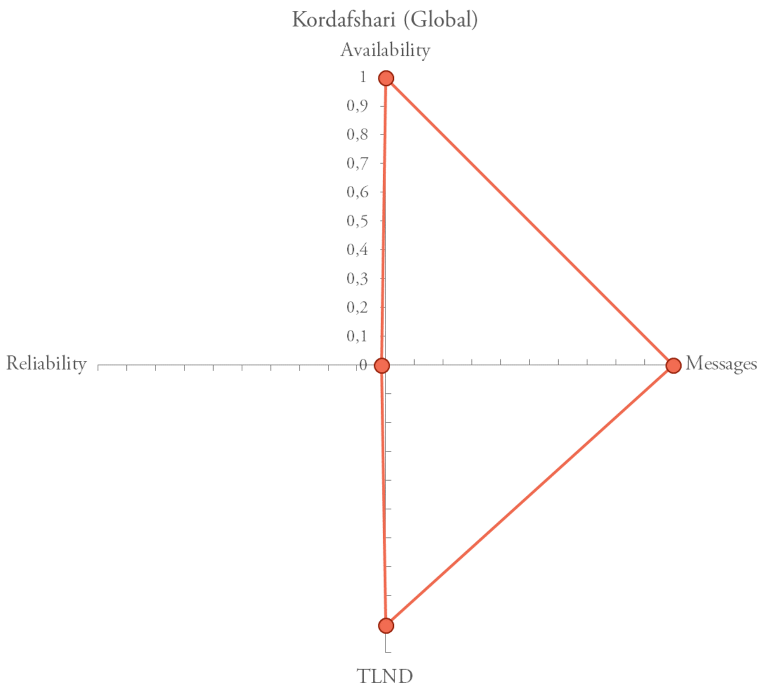


Fig. 7.41 *Global performance of the Kordafshari algorithm (average results over the three mobility models).*

Regarding the Consensus algorithm, although it provides a slight improvement on the reliability in comparison to the Bully and Kordafshari algorithms, globally speaking it does not provide any relevant characteristic that differentiates it over the other algorithms.

The Vasudevan algorithm can be distinguished by the reduced number of Redundant Server Election caused, i.e., by its high reliability. However, to achieve this, it makes extensive use of Coordination Messages, in comparison to the other algorithms evaluated. This has a negative impact on the bandwidth, energy consumption and workload on the system, which could make the algorithm infeasible in specific contexts.

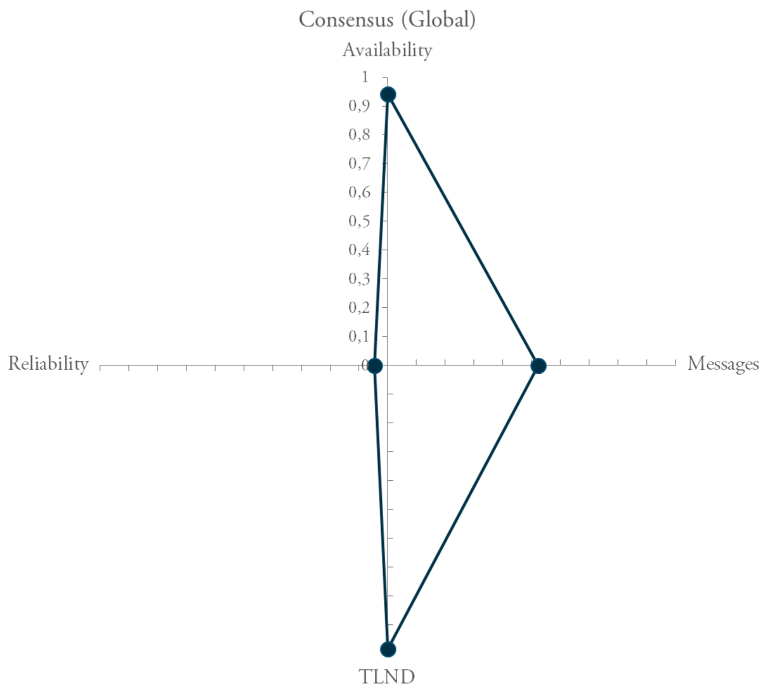


Fig. 7.42 *Global performance of the Consensus algorithm (average results over the three mobility models).*

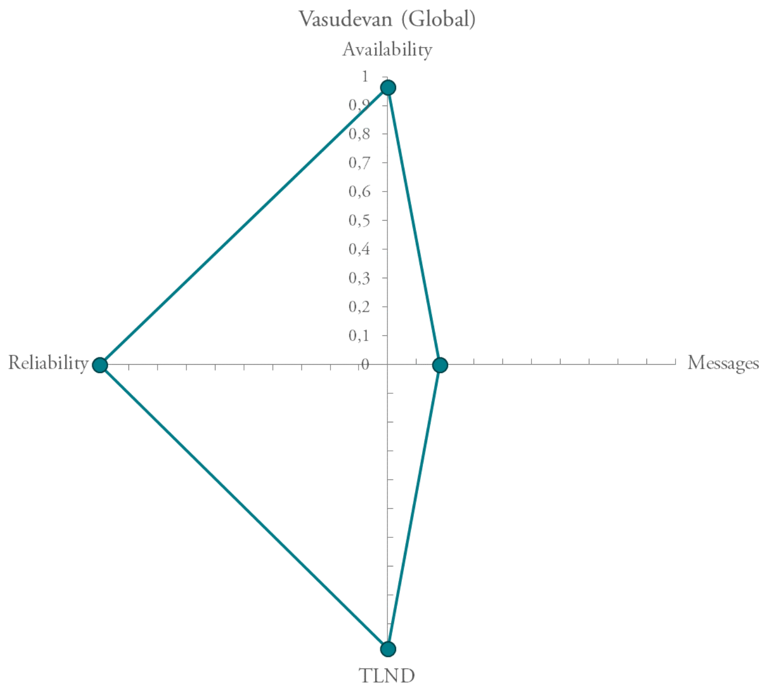


Fig. 7.43 *Global performance of the Vasudevan algorithm (average results over the three mobility models).*

Finally, the Voting algorithm shows results near to optimum regarding service availability and Coordination Messages usage. In global terms, it only provides a 2.19% less of service availability than the Bully algorithm, 98.19% against 96%, and it only uses a 0.98% of Coordination Messages more than the Kordafshari algorithm, 24674.78 against 24917,67. However, it provides better reliability than these algorithms, and in comparison with the Vasudevan algorithm, which only causes 0.15% Redundant Service Elections, the Voting algorithm causes 0.21%. This is, in global terms, a 0.06% more, a slight increase. However, the reliability provided together with the reduced number of Coordination Messages used in comparison to the Vasudevan algorithm (an 81.61% less) make to the Voting algorithm to provide the most extended Network lifetime (TLND), and in global terms, the more balanced performance in comparison to the other election algorithms studied.

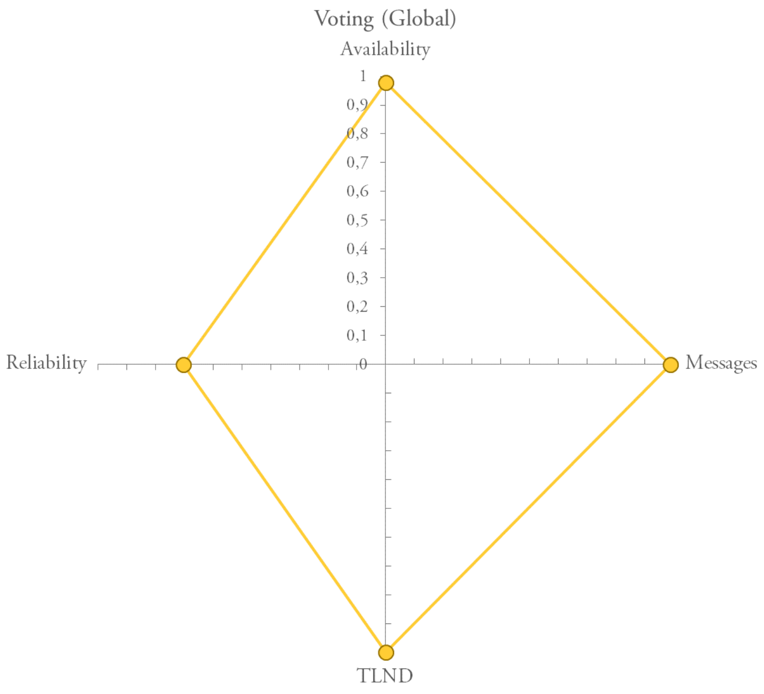


Fig. 7.44 *Global performance of the Voting algorithm (average results over the three mobility models).*

7.8 Summary

This chapter has presented the results obtained during the evaluation of the Proteo architecture in ns-3. The analysis has been carried out regarding service availability, election algorithm reliability, coordination messages usage and network lifetime, considering different mobility models.

In respect to **service availability**, the Bully and Kordafshari algorithms have shown the best performance, independently of the mobility model. The Voting algorithm has shown better performance on the RPGM mobility model than on the others mobility models. The Vasudevan algorithm sees its performance profoundly reduced when the number of nodes increases, both on the RPGM and the Random Walk mobility models. Finally, the Consensus algorithm, generally, provides the worst results.

About **algorithm reliability** (i.e., number of redundant server elections) the Bully and Kordafshari algorithms provide poor reliability and are profoundly affected by the increasing number of nodes in the network. On the contrary, while the number of nodes in the network increases, the Consensus algorithm increases its reliability. Finally, the Voting and Vasudevan algorithms provide excellent results independently of the mobility model or network size.

The Kordafshari and Voting algorithms provide the best results regarding **coordination messages usage** in the three mobility models, whereas the Vasudevan algorithm provides the worst results. The use of coordination messages of this algorithm is profoundly affected by the number of nodes participating in the election and by the kind of network topology on which the algorithm is operating, being the worst case a mesh network topology.

Because of its high reliability and reduced usage of coordination messages, the Voting algorithm provides the best results concerning **network lifetime**.

In *service availability/coordination messages* **efficiency**, the Bully and Kordafshari algorithms provide the best results, except for the RPGM model, in which these are given by the Voting and Kordafshari algorithms. In *service availability/network lifetime* efficiency, the Voting algorithm shows the best performance, regardless of the mobility model.

The Voting algorithm provides the best *algorithm reliability/coordination messages* **efficiency**, in contrast to the Bully and Kordafshari algorithms. Regarding *algorithm reliability/network lifetime*, the Vasudevan and Voting algorithms show the best performance, providing similar results, followed by the Consensus algorithm.

Finally, from a global point of view, the Voting election algorithm proves to show the best behaviour, although Kordafshari provides a slight improvement in service availability at the expense of reliability and Vasudevan provides a slight improvement in algorithm reliability at the expense of coordination message usage.

Part III

Work in Progress and Conclusions

Chapter 8

Work in Progress

Chapter Abstract

In this chapter, the work already in progress to continue developing Proteo architecture is introduced. Nowadays, there are two main lines of research: (1) **addressing synchronization and system interoperability**, for which a service platform has been proposed. It intends to provide a common basis for the consistent management of shared information in ubiquitous collaborative systems, from the software design stage. The services that compose this platform are expected to be included in Proteo architecture. The proposed solution has been applied to the FLERSA tool, devised to transform a *"Content Management System"* (CMS) into its semantic equivalent. As a result, FLERSA tool is provided with quality attributes such as scalability, interoperability, and business agility; (2) **addressing scalability**, for which a study has been performed to know the number of nodes that Proteo can manage using the Consensus election algorithm without degrading its performance. This is essential for applying Proteo in IoT environments, which are devised to support a large number of heterogeneous devices. Additionally, this study aims to stablish if the reliability of TCP could result in an enhanced system operation, despite its high latency and higher consumption of bandwidth in comparison to UDP. As result of the study, it is concluded that TCP presents a better message delivery efficiency, whereas the Proteo architecture does not notably increase the use of bandwidth. Under UDP, the Consensus election algorithm presents a slight improvement in the response time, increasing the service availability provided by the architecture. Under Consensus election algorithm, Proteo architecture proves to operate correctly in networks of 16-18 nodes.

Chapter Contents

8.1 Introduction

This chapter introduces an overview of work already in progress to continue expanding and improving Proteo architecture. The chapter is divided into two sections. First, in Section 8.2, a service platform to support, from the software design stage, the **synchronization** and consistent management of shared resources in dynamic environments is introduced. This is expected to be included in Proteo architecture to address replica synchronization. The platform proposed is also intended to address system **interoperability** (Section 2.5.2).

Secondly, in Section 8.3 the **scalability** of Proteo architecture (Chapter 4) is analysed. To this end, Proteo is evaluated under both the reliable TCP transmission protocol and the non-reliable UDP transmission protocol using the Consensus election algorithm (Section 4.6.1). The analysis has two objectives: (1) to know the number of nodes that Proteo can manage using the Consensus election algorithm without degrading its performance; and (2) to establish if the reliability of TCP could result in an enhanced system operation, despite its high latency and higher consumption of bandwidth in comparison to UDP.

8.2 Towards a Synchronization Solution

Collaborative systems are complex, this is challenging in analysis, modelling, and development [111]. One of the main tasks to be solved in collaborative systems is to maintain data consistency when several users [44] simultaneously shares them. Nowadays, in the absence of standardized methods for the synchronization of the shared data replicas, most of the proposed solutions are planned in an ad-hoc manner. By taking into account the possibility of an increasing number of users and resources to be managed in very dynamic environments, this entails a higher complexity in the correct synchronization of these resources. Thus, it has been proposed a SOA 2.0-based platform, which intends to provide a common basis for the consistent management of shared information in collaborative systems. It consists of two main services (Fig. 8.1):

- **Monitoring Service**. This service gathers all events related with modifications on shared data. This information can fulfil several purposes, e.g., version control or security logs. For the synchronization purposes, this information is required by the specific synchronization algorithm to be applied to know the occurrence and order of the modifications on the resource. The *Monitoring Service* is able to communicate under two different paradigms (SOA 2.0 [90]): (1) the *Publish-Subscribe* paradigm, to know the modifications produced by the users on the shared data, and (2) the *Request-Response* paradigm,

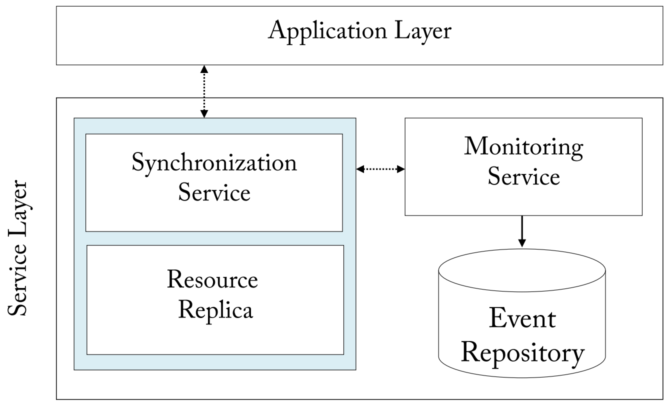


Fig. 8.1 *Architecture of the generic service platform to support sharing and collaboration.*

for example, when the *Synchronization Service* requests to the *Monitoring Service* about the modifications produced on a specific resource in a specific time interval. Thus, the use of an EDA approach [100], specifically the concept of event, allows the developers to provide a reusable service. The *Monitoring Service* has been designed taking advantage of the low coupling in the communications between the sender and receiver provided by EDA. Hence, it can monitor any event. In this way, regarding resource monitoring, in the specialization of the platform it is only necessary to design the structure of the events that will be sent when the shared resources are modified.

- **Synchronization Service**. The service platform proposed aims to be a generic platform that can be applied in any application domain within the collaborative systems. However, as the synchronization algorithms are dependent on the resource type and its specific nature and usage. Namely, the conflicts that could be generated in the concurrent modification of images are not the same that in the modification of, for example, plain text, as well as nor the processes or policies to resolve them. For this reason to provide a general service for the synchronization is not possible. Therefore, in order to provide a reusable service, the Synchronization Service is designed as abstract service, which must be specialized according to each particular resource to be synchronized. This abstract service uses the *Monitoring Service* to obtain information about changes on the different replicas of the shared resource.

In this way, and to provide a more generic service platform, it is designed considering two different levels: (1) the common part related to manage the resource synchronization, which is identified and located into the abstract service and its composition with the *Monitoring*

Service. In this way, the *Synchronization Service*, according to the information received from the Monitoring service, can detect the actions that have been applied to other replicas of the resource, but not applied to its associated replica; (2) the specialization of this service, where, once the inconsistencies are detected, they should be resolved. This level will depend on the requirements associated with the resource kind and the use of the resource in a specific domain (i.e., on the same resource it could be applied different synchronization policies).

8.2.1 Application Scenario: From FLERSA Tool to FLERSA Service

FLERSA [107] has been developed to transform a *"Content Management System"* (CMS) [14] into its semantic equivalent, to partially mitigate the lack of semantic content of the current Web and take advantage of the multiple benefits offered by Semantic Web technologies. The main originality of the tool is to use the manual and multiple-users annotations that can be added at any moment, to learn to annotate documents automatically. Furthermore, these annotations may be related to any pieces of an HTML document (the whole document, a node, a set of nodes, or a text segment). The main functions that the tool enables are the following ones: creation of annotations associated with a range of text, editing/deleting existing annotations, clearing all annotations in the document, permanent storing of annotations, creation of global annotations to a web page (where the scope of the annotations are whole pages) visualization of RDF generated for the page (W3C's RDFa Distiller [171]), ontology-based queries about properties that have been annotated. The inference is made in the taxonomies of concepts when the search is conducted by the annotation properties and automatic generation of semantic annotations.

FLERSA tool has been developed as a module of Joomla![1], for this reason there is a high coupling between the tool and the CMS. As a solution to this limitation, we propose to adapt FLERSA to the new architectural design followed by the SOA 2.0 architecture. This will lead to providing FLERSA with a clear differentiation and independence of the client-side, the CMS (Fig. 8.2).

To adapt an existing tool or service to the proposed architecture, the developer must first clearly identify what the functionality of the tool that must be implemented in the server-side is. In the FLERSA case, this functionality is about the information retrieval, storing and reasoning on the knowledge base, which now will be implemented in the specialization of the Synchronization Service, to take advantage of the architecture proposed. Currently, the services that support this architectural design have been implemented both in C++ and

[1]Joomla Content Management System (CMS), https://www.joomla.org/

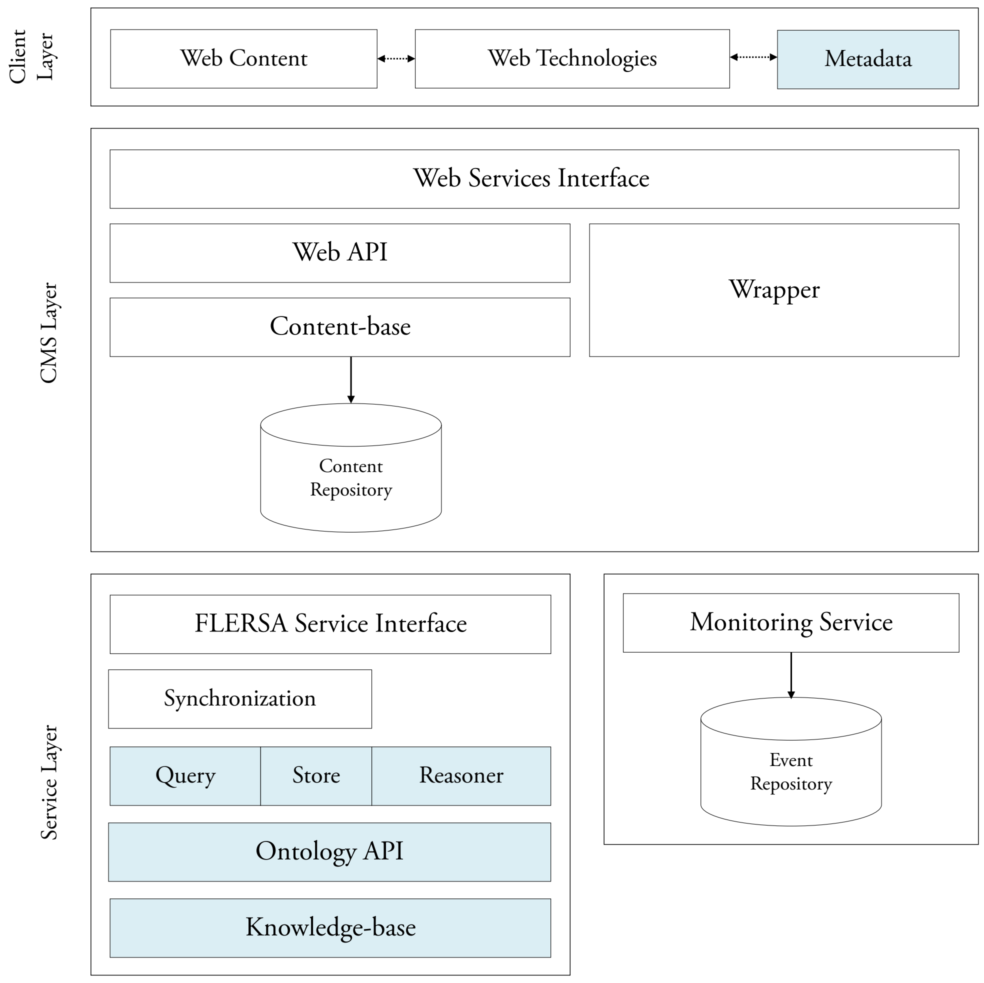


Fig. 8.2 *The SOA architecture for the FLERSA tool.*

in C#. Nevertheless, the programming language is only a technical issue that no affects to the interoperability of the architecture since standard protocols for exchanging information (e.g., SOAP) have been adopted, as well as communication approaches for loosely coupled components (e.g., EDA).

Once the server-side functionality of the tool is implemented, as the specialization of the Synchronization Service, it is necessary to identify which is/are the shared resource(s), the actions that users can perform on it, and the possible inconsistencies that can arise because of that actions. This is one of the most important steps in the adaptation process, owing to the correctness of the resource depends on the correct identification of the possible inconsistencies and the resolution policies applied. In the FLERSA case, the shared resource is a knowledge-base, whereas the actions that can be carried out on the resource are: add, modify and delete semantic annotations. Moreover, these annotations can be performed on the whole document, a node, a subset of nodes, or a text segment, and each of these elements can present several annotations. Therefore, the conflicts during the use of the FLERSA service can be caused by deletions or modifications coming from different users on the same annotation. To solve this kind of conflicts, a version control has been implemented, where a deletion or modification is not permanent and it is possible to revert to the previous version of the annotation.

These all actions are represented as events in the new architectural design. This will facilitate the management and broadcasting of the actions performed on the shared resources along the concerned entities. In the proposed platform, it is used the BlueRose communication middleware [135], which provides a Publish/Subscribe service and an interface for event managing. An event is represented using a pair of topic-attributes, where each attribute is a pair of key-value. In this way, the topic denotes the event type, which is unique in the system, whereas with the set of attributes it is possible to represent information of a wide range of complexity.

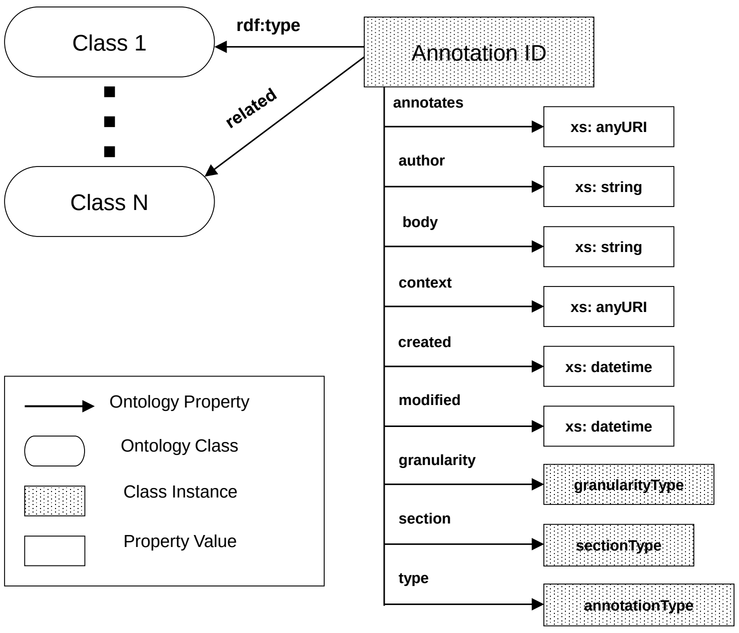


Fig. 8.3 *Graph showing an instance of the class 'Annotation' implemented by FLERSA. Extracted from [107].*

The flexibility and low coupling provided by the EDA approach have made possible to design and implement a Monitoring Service, which is designed and implemented to monitor any event. In the FLERSA case, three types of events taking into account the possible actions that a user can perform have been considered: add, modify or delete an annotation. The generated events contain information related with the user who performs the action, the content related with the annotation (the whole document, a node, a subset of nodes, or a text segment), the timestamp, and information related with the semantic content of the annotation (see Fig. 8.3). These events are stored in a NoSQL database (i.e., a non-relational database) [28], specifically MongoDB[2]. The NoSQL systems arise to address the scalability problems of the traditional databases (i.e., relational) through a more flexible storage structure. Particularly, the absence of data schema allows storing any information as a register with a key-value structure. This makes a NoSQL database ideal for storing any kind of event generated in the system, whereas in a traditional database it would be necessary to create a new table to store each new kind of event. To this regard, the Monitoring Service translates the events from BlueRose format to MongoDB (JSON format) and vice versa.

In the server-side is implemented, as a specialization of the *Synchronization Service*, and together with the identified events, the functionality of the client-side must be additionally implemented. This can be done by creating a new application or adapting an existing one. In the latter, to make use of third-party applications, the common solution is to implement an intermediate entity (known as wrapper) that is capable of translating the petitions of the client to the server and vice versa, and thus it is also capable of adapting existing interfaces to the new service as well. In the FLERSA case, the client-side is located in the CMS module, where a wrapper function is implemented. In this way, if the CMS changes then only a new corresponding implementation of the wrapper function will be required to use the new FLERSA service. Web documents and metadata, like RDFa, are also in the *Client layer*. These resources are downloaded to the device temporally when the client accesses the CMS.

In addition to the benefits initially mentioned obtained through the adaptation of FLERSA to the new architectural design proposed, this SOA architecture also manages a number of additional events at system and infrastructure levels [55], as the battery level of the device or network topology. This, together with replication and caching techniques, allows to provide a context-aware solution, and therefore guarantee the quality attributes of the FLERSA service in AmI environments where the context conditions (e.g., disconnections) are continuous, and they can affect to the proper functioning of the service.

The integration of FLERSA in the proposed SOA architecture has strengthened three of the initial design requirements of the tool [107]: (1) *Requirement 2* - collaborative design/user-centred, as result of the integration FLERSA service allows now the concurrent and distributed edition of HTML documents; (2) *Requirement 5* - evolution of documents (document and

[2]MongoDB, https://www.mongodb.com/

annotation consistency), the *Synchronization Service* and the synchronization policies implemented guarantee the consistency of documents, as well as allow users to know the existence of conflictive modifications and to recover previous versions of documents; (3) *Requirement 8* - Integration, now FLERSA functionality has been encapsulated in a service, which facilitates the reusability and interoperability of the tool.

Case Study

The FLERSA service can be useful in several scenarios in the eHealth domain, where the Semantic Web can help to retrieve information [19], share the patient's medical histories created in different health centres and the semantic interoperability of distributed information systems in eHealth [83] for the collaborative decision support in disease diagnosis [48].

Specifically, the collaboration between various specialists for the diagnosis of patients with strange symptoms [6] is of particular interest to illustrate the usefulness of FLERSA Service. On the one hand, Health is an extensive knowledge area with a complex taxonomy, where any department or research group can define some particular protocol or vocabulary. Thus, to establish semantic relations between concepts or procedures is a mandatory step to achieve collaboration between different specialists or health institutions. On the other hand, it is unusual the existence of medical centres containing every health speciality, being generally more common the existence of specialised centres. Therefore, a tool that allows the distributed collaboration could provide clear benefits.

Fig. 8.4 shows a general scenario about how users can work with FLERSA service and how they can collaborate. In the Figure 5 are depicted two web servers and five users. Several users access the FLERSA service by using different types of devices. The Web users use web browsers to access the service through web servers as front-end, while the rest of users use the service through an application deployed in a mobile device (e.g., smartphone or tablet) as front-end.

For existing applications, like web systems and other applications, a specific wrapper function for each one is needed, which translates the requests that the web client makes to the website to the FLERSA service. In the case of new applications, the wrapper function is not required, given that the interface of FLERSA service can be used directly by these applications.

Specific scenarios can be considered in Fig. 8.4. For instance, if a doctor (e.g., *Web User 1* in Fig. 8.4) creates a new semantic annotation in a HTML document (e.g., a patient report). This action will be propagated as an event through the system, and the *Monitoring Service will register it* and applied by the FLERSA Service to the knowledge base. In this way, this change will be reflected on *the Drupal CMS* as well as to the *Joomla! CMS* and the rest platforms and

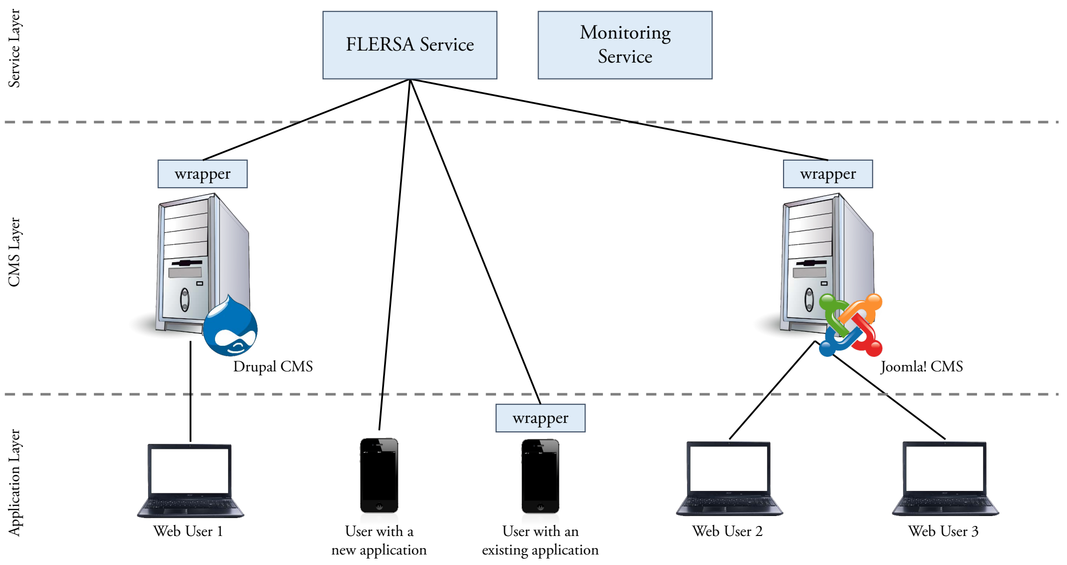


Fig. 8.4 *System architecture for a FLERSA service deployment scenario.*

technologies in the network. That is, FLERSA is now platform independent, which increments its scalability and interoperability. In this situation, the *Web User 1* could be collaborating with specialists that belong to another clinical centre and work on Joomla! (e.g., *Web User 2* and *Web User 3*), or even with mobile users.

The process that a user follows to create an annotation on an HTML document is depicted in the Fig. 8.5. Note that an annotation can be fully or partially overlapped with others. FLERSA resolves this by creating different SPAN sections with unique IDs.

However, under this new configuration, as it has been mentioned, conflicts can occur due to concurrent editing. For instance, if *Web User 1* modifies an annotation on a patient report, while *Web User 2* deletes it, FLERSA Service is in charge of maintaining a consistent version of the shared resource. In this case, the users could have created the conflict because of a distraction, a temporal disconnection of one of them, or because of the changes were not reflected each other in time (i.e., they ignore the action performed by the other user), or intentionally. For this reason, the policy implemented in FLERSA service is to apply the last change but maintaining the previous version of the annotations to avoid information loss, notifying to the users of the existence of a potential conflict. In this way, the users can mediate to decide the correct state of the resource but guarantee that both are consistent versions.

The architecture can react to context changes to guarantee the proper functioning of the service. This feature is of particular interest for mobile users, where, for example, specialists would want work together while they move across the hospital or they are travelling (e.g.,

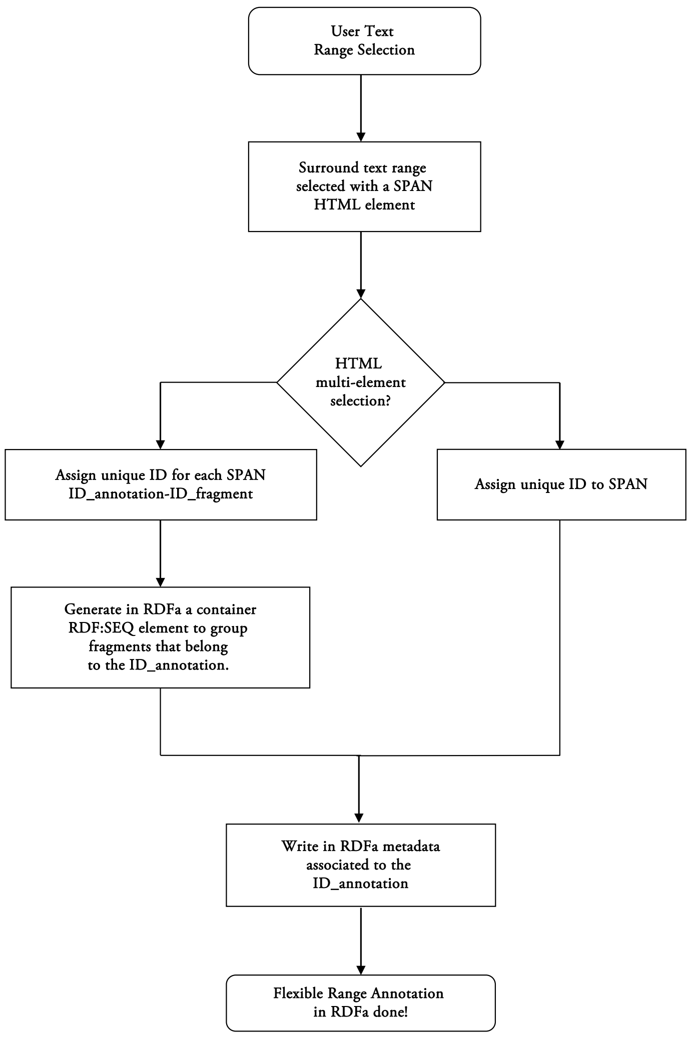

User Text Range Selection
Surround text range selected with a SPAN HTML element
HTML multi-element selection?
Assign unique ID for each SPAN ID_annotation-ID_fragment
Assign unique ID to SPAN
Generate in RDFa a container RDF:SEQ element to group fragments that belong to the ID_annotation.
Write in RDFa metadata associated to the ID_annotation
Flexible Range Annotation in RDFa done!

Fig. 8.5 *Process flow diagram. Extracted from [107].*

by train), where the connection can be lost easily at certain points during that travel. In this case, the FLERSA service receives events regarding the connection loss, and together caching techniques, it can allow users to the user continuing working transparently to the connection lost. The modifications made on the local copy of the resource are stored locally at the device during the disconnection, as a set of events. Later, when the connection is recovered, the FLERSA service will be able to synchronize the modifications that the user has made offline with the modifications of the rest of the users that were online. When FLERSA service (as specialization of the *Synchronization Service*) receives the request of reconnection from the client that was offline, together with the actions that he/she has performed in that period on the cache copy of the resource (as a set of events), it will request to the *Monitoring Service* about what changes have been made in the primary copy of the resource during that period of time by the rest of the users. Once that FLERSA service has the two set of ordered events, it can detect the conflictive events (i.e., the conflictive modifications) and apply the versioning policy described above. Similarly, the application could decide to start working locally when the battery of the device is low and synchronize the changes later, when the energy level is not critical.

8.2.2 Conclusions

This section has presented the evolution of the FLERSA tool towards an SOA proposal for consistent knowledge sharing and collaboration in AmI environments and IoT, and published in [56]. The proposal provides FLERSA tool capabilities such as scalability [176], since the services are loosely coupled; interoperability [145], it is possible to expose any existing data source like as service and to implement workflows that allow exchange of information between different services and platforms through a communication protocol; and business agility [20], thanks to service reusability and the use of access and publication standards. The design of tools (like FLERSA) based on the proposed architecture aims to provide a general solution that opens up new possibilities for the development and deployment of other AmI applications by making use of different technologies and heterogeneous platforms in IoT. For example, the FLERSA service can be now deployed simultaneously in several the Cloud providers and IoT nodes (smartphones or on-board car systems), obtaining benefits such as higher availability, the resources or services will also be available wherever an Internet connection exists; transparency, the user no longer has control of the geographical location of service; and the resources can be increased or decreased as needed.

The SOA 2.0-based architecture for the FLERSA deployment is designed to provide support to the distributed collaboration in environments that exhibit intermittent operations. To this regard, it combines caching and replication techniques together with a context-aware approach to provide a solution to address complex interactions between users in AmI environments in a transparent way. Therefore, it provides a common basis to handle the changing execution context in which AmI and IoT applications can be deployed.

8.3 Scalability Analysis

IoT is devised to support a large number of heterogeneous devices, from simple sensors to smartphones or even computer-based subsystems in cars, sharing information with each other (Section 2.2.2). Additionally, according to [22], applications and systems on IoT are supported by a common base of technologies and protocols. These technologies can be divided into different layers, as can be seen in Figure 8.6: *Application, Service platform & Enabler, Network, Gateway access, Short-range communication and sensing*. In *Service platform & Enabler*, among other approaches, SOA can be found. It is usually based on SOAP or REST protocols, which in turn are based on other protocols, such as CoAP, SMTP or HTTP. Specifically, CoAP and MQTT are protocols specifically designed for IoT. CoAP, *"a specialized web transfer protocol for use with constrained nodes and constrained (e.g., low-power, lossy) networks"* [147]; MQTT [72] a publish/subscribe protocol for WSN. Nevertheless, ultimately, these protocols and technologies usually operate over TCP/IP or UDP/IP. Note that Figure 8.6 is not intended to show a complete list of technologies and protocols.

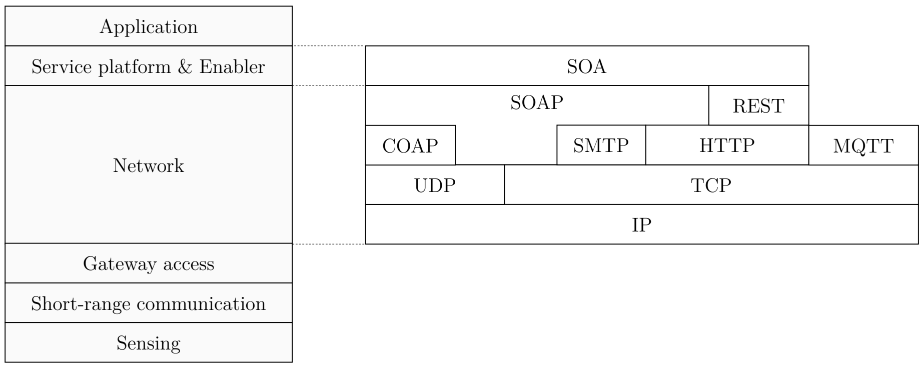


Fig. 8.6 *IoT layers and protocols.*

IoT environments are usually characterized by being resource-constrained, especially regarding energy and bandwidth. Additionally, unreliability in the communication channels may affect the correct system operation, owing to the loss of messages. Under these circumstances, the selection of a particular transmission protocol is crucial as it may have a deep impact on the performance and scalability of the whole system. The TCP transmission protocol, which is connect-oriented and reliable, could help to address the unreliability of this execution environment, thus ensuring the correct operation. However, reduced communication latency, reduced overload in the system, and high data transfer speed of the non-reliable UDP protocol could help to address the problems of a resource-constrained environment and to increment the system scalability.

This section presents a study of the system performance and scalability in terms of service availability and use of bandwidth of the Proteo architecture (Section 4). The analysis is carried out under both the reliable TCP transmission protocol and the non-reliable UDP transmission protocol, using the Consensus election algorithm (Section 4.6.1). The objective is to establish if the reliability of TCP could result in an enhanced system operation, despite its high latency and higher consumption of bandwidth in comparison to UDP.

8.3.1 Evaluation

The simulated scenario consists of:

- A set of mobile nodes ranging from 4 to 25.
- A Random walk mobility model:
  - The speed varies between 0.5-2 m/s.
  - Pauses with a duration that ranges from 60-300 seconds.
- The nodes are equipped with an IEEE 802.11 wireless connection:
  - An approximated range of 250 m.
  - Bandwidth of 1 Mbps.
- The Communication channel is unreliable, the buffer size is limited and the network congestion is possible.
- Mobility area is 1000 m^2.
- Nodes are introduced in a random initial position.
- The time of the simulated execution is one hour.

Under these conditions, two different versions of the Consensus election algorithm (Section 4.6.1) were used: (1) based on a TCP communication, which is reliable, and (2) based on a UDP communication, which is non-reliable (no acknowledgement provided of the message delivery) but requires fewer resources than TCP (e.g., bandwidth). To eliminate the influence of any possible random factor, each configuration was simulated 100 times with 100 different random seeds.

Table 8.1 *KB sent and loss in the node communication for the execution of the host election algorithm, under the TCP and UDP protocols.*

	Nodes	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
TCP	KB sent	0.4973	0.7698	1.4283	1.8298	2.4987	3.5122	4.3226	5.4847	6.5302	8.1340	9.4053
	KB loss	0.0016	0.0035	0.0079	0.0119	0.0209	0.0303	0.0415	0.0643	0.0805	0.1065	0.1395
UDP	KB sent	0.4392	0.6874	1.3532	1.7521	2.4527	3.2260	4.3982	5.4242	6.7014	7.9555	9.2631
	KB loss	0.0089	0.0182	0.0333	0.0566	0.0954	0.1472	0.2156	0.3264	0.4265	0.5908	0.7728
	Nodes	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**
TCP	KB sent	11.0173	12.7699	15.1770	19.2242	34.1736	63.8355	108.4401	154.6433	191.4331	223.0439	259.4148
	KB loss	0.1661	0.3507	1.0883	3.6481	16.8574	44.7002	88.8357	133.5791	168.8155	199.5664	234.4949
UDP	KB sent	10.7583	12.5292	14.7158	17.9600	26.9999	40.6346	57.3161	69.5707	75.0354	78.7156	81.2065
	KB loss	0.9608	1.3157	2.1352	4.1512	11.7903	23.9259	40.3462	51.4914	55.7248	58.8248	60.2697

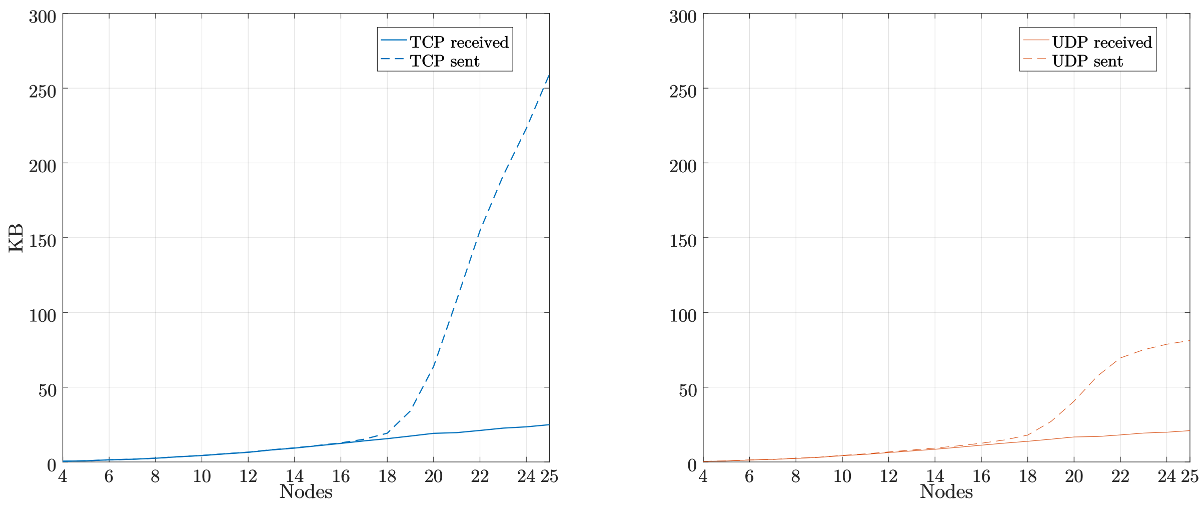


Fig. 8.7 *KB sent and received by the nodes of the network, owing to the execution of the host election algorithm, under the TCP protocol (left) and the UDP protocol (right).*

From the results obtained, two main aspects were evaluated to know if the reliability of TCP results in a better operation of the host election algorithm against UDP: (1) use of bandwidth, and (2) service availability provided by the architecture.

Use of bandwidth

Regarding bandwidth consumption, two distinct intervals were observed: a network composed of up to 16 nodes, where the host election algorithm operates with normality, and a network of more than 16 nodes, where the host election algorithm begins to saturate the bandwidth, with both transportation protocols. This issue is depicted in the chart of Figure 8.7, which shows the

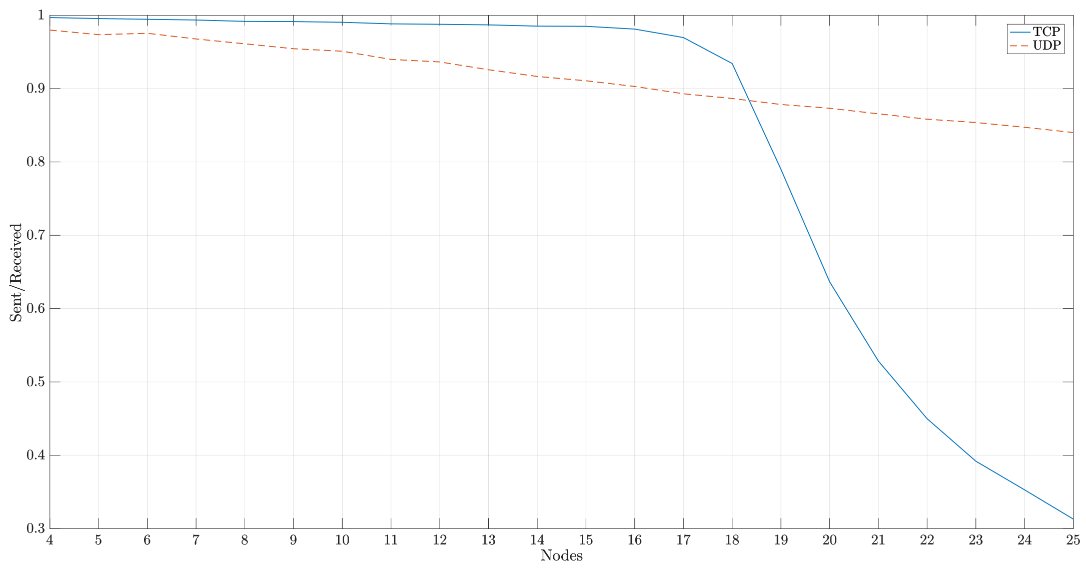


Fig. 8.8 *Efficiency in the message delivery of TCP and UDP protocols.*

amount of KB sent generated by the execution of the host election algorithm, and received by the nodes of the network, under the TCP and UDP protocols.

In the first interval, from 4 to 16 nodes, TCP shows a slight increase, but not relevant, in the KB generated against UDP, as shown in Table 8.1. However, UDP shows a problem with message loss, losing more than 1KB of information in a network of 16 nodes (Table 8.1). In this regard, TCP shows better efficiency in the message delivery near to 1 in the 4-16 interval (Figure 8.8), whereas UDP shows a constant efficiency decrease, losing nearly 10% of the information sent in a network of 15 nodes (Figure 8.9).

However, in the second interval (17-25 nodes), TCP shows a complete saturation of the network. In this case, TCP shows an exponential increment in the KB sent (Figure 8.7), generating 250 KB of traffic approximately in a network of 25 nodes, of which only 24.9198 KB are received. The TCP efficiency in message delivery decreased exponentially (Figure 8.8), and from a network 19 nodes and upwards, the percentage of KB loss of TCP exceeded the UDP loss (Figure 8.9).

Service availability

The availability of the service is affected by the time the host election algorithm takes to choose a host to act as the server. Thus, this decision, which involves all nodes of a network partition, is influenced by the latency in the communication. In this case, owing to the saturation of the network, two distinct intervals are found as well.

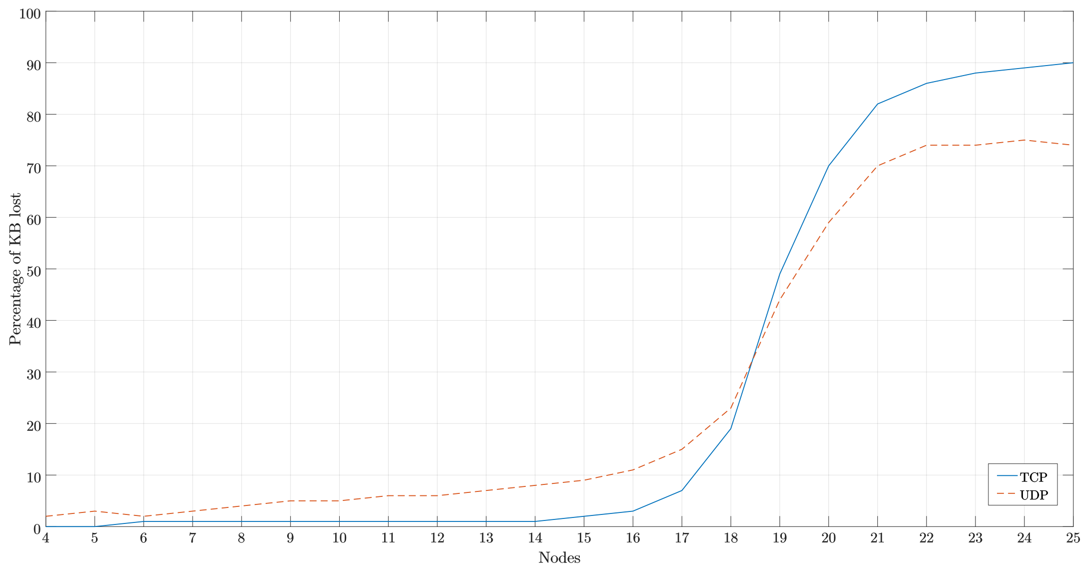


Fig. 8.9 *Percentage of KB loss in the node communications of TCP and UDP protocols.*

Table 8.2 *Service availability provided by the self-adaptive software architecture, under the TCP and UDP communication protocols.*

Nodes	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
TCP	99.81%	99.73%	99.46%	99.37%	99.21%	98.88%	98.77%	98.53%	98.42%	98.00%	97.89%
UDP	99.84%	99.77%	99.45%	99.39%	99.20%	98.99%	98.66%	98.43%	98.22%	98.03%	97.91%
Nodes	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**
TCP	97.55%	97.33%	96.72%	95.55%	90.67%	82.51%	70.88%	61.88%	56.60%	53.16%	49.74%
UDP	97.69%	97.51%	97.26%	97.23%	97.10%	96.93%	97.01%	96.91%	96.84%	96.78%	96.67%

In the first interval, networks composed of less than 16 nodes, the host election algorithm provides a similar service availability with both communication protocols, as shown in Table 8.2. Specifically, except for the case of a network of 10, 11 and 12 nodes, a slightly lower service availability is provided under TCP (Figure 8.10).

In the second interval, networks with more than 16 nodes, TCP begins to saturate the network, and the service availability is drastically reduced as a result of this. However, although UDP also saturates the network, to a lesser degree, the service availability provided is not significantly reduced (Figure 8.10).

8.3.2 Discussion

The evaluation performed has highlighted that the current proposal presents a scalability problem. Under UDP the self-adaptive architecture begins to exhibit a problem with message

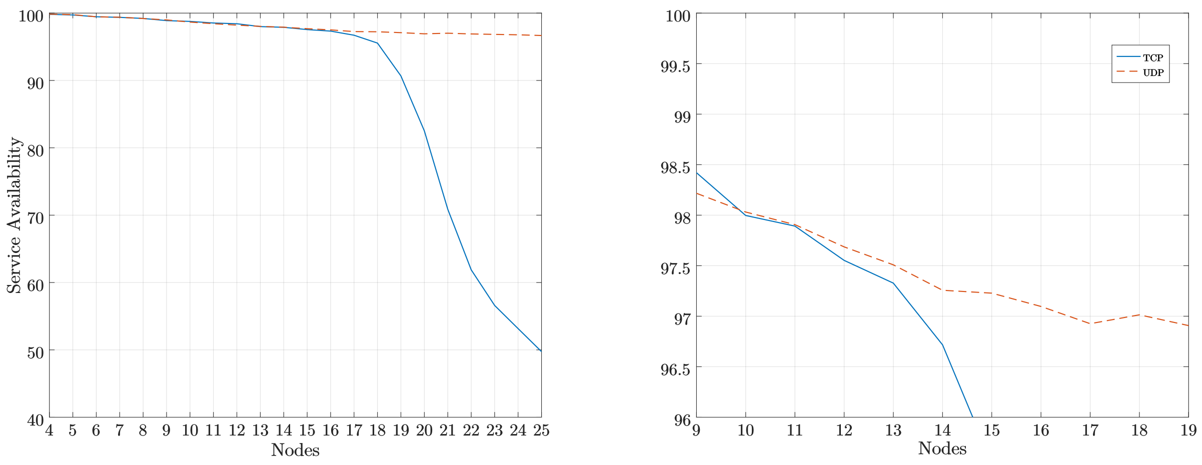


Fig. 8.10 *Service availability provided by the self-adaptive software architecture, under the TCP and UDP communication protocols. Networks from 4 to 25 nodes (left); from 9 to 19 nodes (right).*

loss in networks of 10 nodes or more, whereas, under TCP, the proposed approach operates correctly for networks of 16 nodes or less. In networks of more than 16 nodes, the Consensus election algorithm saturates the network under both protocols, especially under TCP. Moreover, the architecture follows a copy-primary scheme [54] in physical partitions, which in large-scale networks is inefficient, since in large groups the active replica could easily result in a bottleneck. This emphasises the importance of applying device clustering techniques to the architecture to improve its scalability in large ad-hoc systems, such as those that can be found in IoT environments. In this respect, it is worth stressing that the incorporation of device clustering techniques will be transparent and independent for the proposal, as the election algorithm is applicable both to managing physical network partitions and logical partitions.

Furthermore, although TCP presents higher bandwidth requirements than UDP owing to the acknowledge packets, in the interval of networks composed of 4 to 16 nodes, it only has shown a slight increase in traffic generated (of 0.0969KB in average). The average efficiency in message delivery of TCP is 0.9898 in the 4-16 interval, against the 0.9458 presented by UDP. However, the service availability provided by the architecture is lower under TCP than UDP. On average, under UDP the architecture provides a service availability 0.1572% higher, in this interval. This is caused by the lower latency of UDP against TCP, which allows the host election algorithm to provide a better response time. Therefore, in this case, as demonstrated, the bandwidth requirements of TCP are not much higher than the requirements of UDP, whereas the reliability of TCP and lower latency of UDP are relevant features to take into consideration in the design of the self-adaptive architecture.

8.3.3 Conclusions

This section has presented and analysed the results obtained from the study of the behaviour of the Proteo architecture, with the Consensus algorithm, under different transmission protocols: TCP, a reliable protocol, and UDP, a non-reliable transmission protocol. From this study, the following can be concluded: TCP presents, as was expected, a better message delivery efficiency. In a network of 16 nodes, TCP provides a message delivery efficiency of 0.9811, whereas UDP presents 0.9029. Contrary to what was initially expected, the Proteo architecture does not notably increase the use of bandwidth under TCP. In the interval of networks composed of 4 to 16 nodes, it has only shown a slight increase in traffic generated (0.0969KB in average). Under UDP, the host election algorithm presents a slight improvement in the response time, improving the service availability provided by the architecture, as a consequence of the low latency of this transportation protocol.

8.4 Summary

In this chapter, the work already in progress to continue developing Proteo architecture has been presented. Currently, the research work is flowing two principal lines of research:

- *Addressing resource synchronization and system interoperability*: through a service platform which provides to the software engineers the basis for a correct and consistent management of replicated/distributed resources in ubiquitous collaborative systems. This is achieved through the Synchronization and Monitoring services. The services that compose this platform are expected to be included in Proteo architecture to provide a complete solution that also considers the synchronization of the service replicas.

- *Addressing scalability*: for which a study has been performed to know the number of nodes that Proteo can manage using the Consensus election algorithm without degrading its performance. Additionally, this study aims to establish if the reliability of TCP could result in an enhanced system operation, despite its high latency and higher consumption of bandwidth in comparison to UDP. In the study it is concluded that, under Consensus election algorithm, Proteo architecture proves to operate correctly in networks of 16-18 nodes. Thus, to improve its scalability in large ad-hoc systems is necessary to apply clustering techniques. In this respect, the incorporation of device clustering techniques will be transparent and independent for the proposal. The election algorithm is applicable both to managing physical network partitions and logical partitions, and the clustering will be managed in an underlining layer.

Chapter 9

Conclusions and Future Work

9.1 Conclusions

This research work has aimed to address the existing lack of suitable approaches to support the quality attributes of mobile and ubiquitous systems by specifically facing challenges related with dynamic topologies. To this end, a self-adaptive software architecture, named Proteo, has been proposed.

Proteo architecture is composed of three main components, which are replicated on the nodes that compose the distributed system network: (1) *Monitoring Subsystem*, (2) *Context Manager Service*, and (3) *Replica Manager Service*. Their responsibilities are closely related to the phases of MAPE-K autonomic loop of Autonomic Computing: the *Monitoring Subsystem* senses the context information in relation to the node in which is deployed; the *Context Manager Service*, in addition to process this information, is also responsible for analysing it to decide when a new system adaptation is necessary; and the *Replica Manager Service* will be responsible for coming to an agreement with the rest of the *Replica Manager Services* deployed in the system to establish what replica will be activated. This coordination is performed by using a distributed host election algorithm, and the election is based on a dynamic score obtained through a utility function at run-time. This utility function indicates how suitable is a node to host a service replica.

Proteo follows a modular design and its detailed model has been described using SysML. This allows integrating into the architecture different election algorithms and utility functions. Additionally, thanks to the SOA 2.0 approach followed in the design, the elements of the architecture are loosely coupled. In this way, Proteo can be easily extended with new monitoring services, to take into consideration new contextual features to evaluate the feasibility of a node to host a service replica. Similarly, the context events with which Proteo reacts can be extended.

Regarding the election algorithms, two new are proposed within this work: *Consensus* and *Voting*. These election algorithms have been devised to operate in mobile systems with highly dynamic network topologies and under unreliable communication channels.

From the review of the existing proposals related to the dynamic service provisioning in dynamic topology systems presented, the most relevant features for them have been identified and highlighted. From these, Proteo: (1) is reactive to context changes, such as node disconnections; (2) uses local knowledge to the node to take decisions; (3) uses a utility function to evaluate the nodes, which takes into consideration the resources of the nodes (e.g. energy) providing a resource-aware solution.

Additionally, Proteo architecture uses a cross-layer approach to monitoring the network. In this respect, a heuristic method has been proposed to determine the most centric node in the network topology, on the basis of only the information of the routing table of the node. With this more flexible design for managing the dynamics of the mobile environment, Proteo reduces the resource consumption.

Proteo follows a hibernation-based deployment approach. In collaborative systems providing support for tasks in work groups, such as firefighters or rescue and emergency teams, the set of the services is well-known. This makes possible to deploy the replicas of the service in the devices of the group members before system execution. On the one hand, this approach reduces the flexibility of the system, as to deploy a new service or introduce new devices at run-time is not possible. On the other hand, it reduces the requirements for bandwidth and can improve the response time of the configuration process. This is particularly interesting, since the mobile networks currently have important technology restrictions, as a reduced bandwidth. However, these are technical issues that could be resolved in the future, and thus, a hybrid approach would be a better solution concerning pulling together the flexibility of a dynamic replication approach and the response time of a hibernation approach. In this way, the well-known set of services could be deployed before run-time and to add and deploy new services at run-time would be possible as well. This is a technical question, since the current approach of Proteo is transferable between both methods, and the code mobility techniques required to provide the replication and deployment of services at run-time can be implemented in a transparent way for the election process.

To validate and evaluate the proposal, the ns-3 network simulator has been used. To this end, in this work, the ns-3 simulator has been extended with a new module that implements Proteo architecture. Proteo has been evaluated in terms of service availability, election algorithm reliability, coordination messages usage and network lifetime, under three different mobility models: Manhattan Grid, Random Walk, and Reference Point Group (RPGM).

In addition, between the existing approaches, the election algorithms of Bully, Kordafshari, and Vasudevan have been selected and incorporated to Proteo to compare the performance of the new two proposed election algorithms with them. For this purpose, adaptations on the Bully and Kordafshari algorithms have been made. These have been necessary to base the elections, instead of on known IDs, on dynamic scores.

The use of advanced simulation tools has permitted to represent the dynamic nature of distributed and dynamic computer networks, and therefore to handle and evaluate dynamic models. Furthermore, the architecture has been validated and tested under unreliable communication channels, with channel fading caused by the distance, limited buffer sizes, and network congestion. This is a fundamental difference in comparison to the existing studied approaches, which generally assumes reliable communications. From our point of view, this can be considered a strong assumption, since data or message loss is an inherent property of mobile and ad-hoc wireless networks, especially in highly dynamic or large-scale environments.

From the results obtained in the evaluation, it has been observed that:

- In respect to service availability, the Bully and Kordafshari algorithms have shown the best performance, independently of the mobility model. The Voting algorithm has shown better performance on the RPGM mobility model than on the others mobility models. The Vasudevan algorithm sees its performance profoundly reduced when the number of nodes increases, both on the RPGM and Random Walk mobility models. Finally, the Consensus algorithm, generally, provides the worst results.

- About algorithm reliability (i.e., number of redundant server elections) the Bully and Kordafshari algorithms provide poor reliability and are profoundly affected by the increasing number of nodes in the network. On the contrary, as the number of nodes in the network increases, the Consensus algorithm increases its reliability. Finally, the Voting and Vasudevan algorithms provide excellent results independently of the mobility model or network size.

- The Kordafshari and Voting algorithms provide the best results regarding coordination messages usage in the three mobility models, whereas the Vasudevan algorithm provides the worst results. The use of coordination messages of this algorithm is profoundly affected by the number of nodes participating in the election and by the kind of network topology on which the algorithm is operating, being the worst case a mesh network topology.

- Because of its high reliability and reduced usage of coordination messages, the Voting algorithm provides the best results concerning network lifetime.

- In *service availability/coordination messages* efficiency, the Bully and Kordafshari algorithms provide the best results, except for the RPGM model, in which these are given by the Voting and Kordafshari algorithms. In *service availability/network lifetime* efficiency, the Voting algorithm shows the best performance, regardless of the mobility model.

- The Voting algorithm provides the best *algorithm reliability/coordination messages* efficiency, in contrast to the Bully and Kordafshari algorithms. Regarding *algorithm reliability/network lifetime*, the Vasudevan and Voting algorithms show the best performance, providing similar results, followed by the Consensus algorithm.

- Finally, from a general point of view, the Voting election algorithm proves to show the best behaviour, although Kordafshari provides a slight improvement in service availability at the expense of reliability and Vasudevan provides a slight improvement in algorithm reliability at the expense of coordination message usage.

From these results it can be concluded that Proteo architecture, regardless the particular election algorithm proves to improve the quality attributes of the service, specifically availability. **Therefore, the initial hypothesis of this work is verified by the results obtained**. Additionally, although the proposed Voting algorithm shows the better performance in general, this is not the case for all specific scenarios. In this respect, it can be observed that determined algorithms could be more convenient in particular contexts.

However, for Proteo architecture there are still two issues that remain outstanding: (1) the synchronization of the replicas; and (2) the system scalability.

Regarding synchronization, these is an interesting issue that the majority of the existing approaches neither address. However, the increment in the number of resource replicas implies an increase on synchronization messages. This may invalidate some of the proposals that perform an extensive replication of the resource to increment its availability. To this regard, there is already work in progress, and a service platform has been proposed, which provides to the software engineers the basis for a correct and consistent management of replicated/distributed resources in ubiquitous collaborative systems. This is achieved through the *Synchronization* and *Monitoring* services. The services that compose this platform are expected to be included in Proteo architecture to provide a complete solution that also considers the synchronization of the service replicas. That platform has been applied to the FLERSA tool, devised to transform a *"Content Management System"* (CMS) into its semantic equivalent, allowing the collaborative semantic annotation of web resources. As a result, FLERSA tool is improved with attributes such as scalability, interoperability, and business agility.

In respect to system scalability, a study to know the number of nodes that Proteo can manage using the Consensus election algorithm without degrading its performance has been performed. Additionally, this study aimed to establish if the reliability of TCP could result in an enhanced system operation, despite its high latency and higher consumption of bandwidth in comparison to UDP. In the study is concluded that, under Consensus election algorithm, Proteo architecture proves to operate correctly in networks of 16-18 nodes. Thus, to improve its scalability in large ad-hoc systems is necessary to apply clustering techniques. In this respect, the incorporation of different device clustering techniques will be transparent and independent for the proposal. The election algorithm is applicable both to managing physical network partitions and logical partitions, and the clustering will be managed in an underlining layer.

9.2 Future Work

In addition to completely address the synchronization of the replicas and scalability of Proteo architecture, there are additional research lines that should be considered in the future:

- In relation to predictive behaviour, the network partition prediction is of considerable interest in the context of this work. Usually, these approaches monitor the movement of the nodes or the status of the links, to predict the disconnection of a group of nodes. This allows to synchronize or replicate a resource before the change in the topology happens. In this way, the system moves from being a reactive system to a proactive. This is helpful in systems where the replicas are deployed at run-time. However, it can be costly and can result in unnecessary system adaptations.

- Ubiquitous systems, specially IoT, are characterized by possessing a broad range of heterogeneous devices. In this context, the computational requirements of the services, which can be also heterogeneous, could match with the capabilities of certain devices better than with the capabilities of others. Therefore, it would be particularly interesting to provide a semantic mechanism that allows the definition and representation of that grade of matching, in order to consider it in the evaluation of the feasibility of the node to host the service replica. This would result in a more efficient deployment solution in heterogeneous environments.

- In order to extend the applicability of the solution, it would be necessary to incorporate mechanisms to avoid selfish behaviour in the nodes.

- In the same line, the reliability of a software component is directly dependent on the reliability of the components in which it is based. In this context, the reliability of a replica will be influenced by the reliability of the node in which it is deployed. With the purpose of increasing the reliability of the deployment scheme provided by the architecture, methods to measure the reliability of the nodes could be included in the evaluation of the node. For example, if a node frequently disconnects, it will not be a good candidate to be elected as replica host, despite its computational features.

- Finally, as concluded previously in the evaluation of the Proteo architecture, although the proposed Voting algorithm shows the better performance in general, this is not the case for all specific scenarios. This can be understood as that there is no a universal solution that provides the better performance in every possible situation. To this regard, to provide a new level of self-adaptation would be of great interest. This is, for instance, to make possible for the architecture to autonomously use one election algorithm or another according to the particular state of the environment at that specific moment. This could be extended to other elements, as the utility function. In this way, Proteo would be able to self-adapt to the changing requirements of a scenario over time. However, the solution to this problem introduces new challenges. Proteo operates over a distributed system, with no central entity of control and under unreliable communications. If a node decides to change the election algorithm, the other nodes have to do it as well, owing to the coordination between them is not possible when they are using different election algorithms.

Chapter 10

Conclusiones y Trabajo Futuro

10.1 Conclusiones

Este trabajo de investigación ha pretendido abordar la actual carencia de enfoques adecuados para soportar los atributos de calidad de sistemas móviles y ubicuos, enfrentándose a los desafíos relacionados con las topologías de red dinámicas. Con este objetivo, se ha propuesto una arquitectura software auto-adaptativa, denominada Proteo.

La arquitectura Proteo está compuesta de tres componentes principales, los cuales están replicados en los nodos que componen la red del sistema distribuido: (1) *Subsistema de Monitorización*, (2) *Servicio de Gestión del Contexto*, y (3) *Servicio de Gestión de Réplicas*. Las responsabilidades de estos componentes están estrechamente relacionadas con las fases del bucle autonómico MAPE-K, del campo de la Computación Autonómica: el *Subsistema de Monitorización* monitoriza y percibe la información del contexto relacionada con el nodo en el cual se encuentra desplegado; (2) el *Servicio de Gestión del Contexto*, además de procesar la información recibida por el *Subsistema de Monitorización*, es responsable de analizarla y decidir cuándo es necesaria una nueva adaptación en el sistema; y el *Servicio de Gestión de Réplicas*, el cual es el responsable por llegar a un acuerdo con el resto de los *Servicios de Gestión de Réplicas* desplegados en el sistema para establecer cuáles de las réplicas existentes será activada. Esta coordinación se lleva a cabo utilizando un algoritmo distribuido de elección, basándonse en una puntuación dinámica obtenida a través de una función de utilidad, en tiempo de ejecución. Dicha función de utilidad indica cómo de adecuado es un nodo para alojar una réplica activa del servicio.

Proteo sigue un diseño modular. Esto permite integrar en la arquitectura distintos algoritmos de elección y funciones de utilidad. Además, gracias al enfoque SOA 2.0 seguido en el diseño, los elementos de la arquitectura están débilmente acoplados. De esta forma, Proteo puede ser fácilmente extendido con nuevos servicios de monitorización, para tener en cuenta nuevas características contextuales en la evaluación de los nodos. De forma similar, los eventos a los que Proteo reacciona para cambiar el esquema de replicación (por ejemplo, la desconexión del actual servidor) pueden ser modificados o extendidos.

Con respecto a los algoritmos de elección, en este trabajo se han realizado dos nuevas propuestas de algoritmos: *Consenso* y *Voting*. Estos algoritmos de elección se han diseñado con el objetivo de operar en sistemas móviles con topologías de red altamente dinámicas y bajo canales de comunicación no fiables.

A partir de la revisión realizada sobre las propuestas actualmente existentes para el despliegue dinámico de servicios, se han identificado las características más relevantes que estos sistemas deben poseer. De estas características, Proteo: (1) es reactivo a cambios de contexto, como por ejemplo desconexiones de nodos; (2) utiliza información local a nodo para tomar decisiones en cuanto a la auto-adaptación del sistema; (3) utiliza una función de utilidad para evaluar los nodos, con la cual se tiene en cuenta los recursos del nodo, proporcionando una solución consciente del consumo de recursos.

Adicionalmente, la arquitectura Proteo utiliza un enfoque cross-layer para monitorizar la red. A este respecto, se ha propuesto un método heurístico para determinar el nodo más céntrico en la topología de red, en base solo a la información proporcionada por la tabla de ruta del nodo. Con este diseño más flexible para gestionar la dinamicidad del entorno móvil, Proteo reduce el consumo de recursos.

Proteo sigue un enfoque basado en hibernación. En sistemas colaborativos que proporcionan soporte a las tareas de equipos de trabajo, como el caso de bomberos o equipos de emergencia y rescate, el conjunto de servicios es bien conocido y sus réplicas pueden ser desplegadas en los dispositivos de los componentes del equipo antes de la puesta en marcha del sistema. Por una parte, este enfoque reduce la flexibilidad del sistema, debido a que no es posible desplegar nuevos servicios o introducir nuevos dispositivos en tiempo de ejecución. Por otra parte, reduce los requisitos de ancho de banda y puede mejorar el tiempo de respuesta del proceso de adaptación. Esto resulta particularmente interesante, dado que las redes móviles actualmente presentan restricciones tecnológicas importantes, como un ancho de banda reducido. Sin embargo, esto son aspectos tecnológicos que podrán ser resueltos en el futuro, y, por tanto, un enfoque híbrido representaría una mejor solución, combinando la flexibilidad de una replicación dinámica y el tiempo de respuesta de un enfoque de hibernación. De esta forma, el conjunto conocido de servicios puede ser replicado y despegado en los dispositivos antes de ejecutar el sistema y, además, en tiempo de ejecución se pueden desplegar nuevos servicios, previamente no considerados. El diseño seguido en Proteo hace que las técnicas de movilidad de código requeridas para proporcionar el despliegue y replicación de servicios en tiempo de ejecución se pueden implementar de forma transparente al proceso de elección. Por lo que la actual propuesta sería valida en ambos casos.

Para evaluar y validar la propuesta, se ha utilizado el simulador de redes ns-3. Con este fin, en este trabajo, se ha extendido el simulador ns-3 con un nuevo módulo que implementa la arquitectura Proteo. Ésta ha sido evaluada en términos de disponibilidad de servicio, fiabilidad del algoritmo de elección, utilización de mensajes de coordinación y tiempo de vida de la red, bajo tres modelos de movilidad diferentes: Manhattan Grid, Random Walk, y Reference Point Group (RPGM).

Asimismo, entre los enfoques actualmente existentes, se han seleccionado los algoritmos de elección de Bully, Kordafshari y Vasudevan. Estos han sido incorporados en Proteo para comparar el rendimiento de los dos nuevos algoritmos propuestos en este trabajo con ellos. Para hacer esto posible, ha sido necesario realizar modificaciones sobre los algoritmos de Bully y Kordafshari, ya que estos basan la elección en IDs conocidos de los nodos en vez de puntuaciones dinámicas.

El uso de herramientas avanzadas de simulación nos ha permitido representar la naturaleza de redes de computación dinámicas y distribuidas, y por tanto manejar y evaluar modelos dinámicos. Además, la arquitectura ha sido validada y evaluada bajo canales de comunicación no fiables, con desvanecimiento de la señal causada por la distancia, tamaños limitados de buffer y congestión de red. Esto representa una diferencia fundamental en comparación con las propuestas estudiadas, las cuales, generalmente asumen comunicaciones fiables. Desde nuestro punto de vista, esto se puede considerar una suposición importante, dado que la pérdida de información o datos en la transmisión es una propiedad inherente a las redes móviles inalámbricas ad-hoc, especialmente en entornos altamente dinámicos o de gran escala.

De los resultados obtenidos en la evaluación, se ha observado que:

- Con respecto a la disponibilidad del servicio, los algoritmos de Bully y Kordafshari han mostrado el mejor rendimiento, independientemente del modelo de movilidad. El algoritmo de Votación ha mostrado mejores resultados en el modelo de movilidad RPGM que en los otros modelos de movilidad. El rendimiento del algoritmo de Vasudevan se ve profundamente reducido cuando el número de nodos involucrados en la elección se incrementa, tanto en los modelos de RPGM como de Random Walk. Finalmente, el algoritmo de Consenso, generalmente, proporciona los peores resultados en disponibilidad del servicio.

- En lo referente a la fiabilidad del algoritmo (es decir, el número de elecciones redundantes de servidor) los algoritmos de Bully y Kordafshari muestran resultados generalmente pobres, a los cuales les afecta en gran medida el número de nodos de la red. Por el contrario, conforme el número de nodos de la red se incrementa, el algoritmo de Consenso incrementa su fiabilidad. Finalmente, los algoritmos de Votación y Vasudevan proporcionan excelentes resultados, independientemente del modelo de movilidad o el tamaño de la red.

- Los algoritmos de Kordafshari y Votación proporcionan los mejores resultados respecto al uso de mensajes de coordinación en los tres modelos de movilidad estudiados. Por el contrario, el algoritmo de Vasudevan muestra los peores resultados a este respecto. El uso de mensajes de coordinación de este algoritmo se ve profundamente afectado por el número de nodos participando en la elección y por el tipo de topología de la red, siendo el peor caso para el algoritmo de Vasudevan una red de malla ("mesh").

- Debido a su alta fiabilidad y reducido uso de mensajes de coordinación, el algoritmo de Votación proporciona los mejores resultados respecto al tiempo de vida de la red, maximizando el aprovechamiento de energía por parte del sistema, con respecto al resto de algoritmos.
- Los algoritmos de Bully y Kordafshari proporcionan los mejores resultados en relación a la eficiencia de *disponibilidad de servicio/mensajes de coordinación*, excepto para el modelo RPGM, en el cual los mejores resultados los proporcionan los algoritmos de Votación y Kordafshari. Con respecto a la eficiencia de *disponibilidad de servicio/tiempo de vida de la red*, el algoritmo de Votación muestra el mejor rendimiento, independientemente del modelo de movilidad.
- El algoritmo de Votación, también proporciona los mejores resultados con respecto a la eficiencia *fiabilidad del algoritmo/mensajes de coordinación*, al contrario que los algoritmos de Bully y Kordafshari que muestran los peores resultados a este respecto. Respecto a la eficiencia *fiabilidad del algoritmo/tiempo de vida de la red*, los algoritmos Vasudevan y Votación proporcionan, con resultados similares, el mejor rendimiento, seguidos del algoritmo de Consenso.
- Finalmente, desde un punto de vista general, el algoritmo de Votación prueba ser el algoritmo que muestra el mejor comportamiento. Sin embargo, el algoritmo de Kordafshari, con respecto al algoritmo de Votación, proporciona una ligera mejora en la disponibilidad del servicio, a costa de la fiabilidad del algoritmo, y el algoritmo de Vasudevan una ligera mejora en la fiabilidad del algoritmo, a costa de un mayor uso de mensajes de coordinación.

A partir de estos resultados, se puede concluir que: (1) La arquitectura Proteo, independientemente del algoritmo de elección particular, ha demostrado mejorar los atributos de calidad del servicio, específicamente disponibilidad. **Por tanto, la hipótesis inicial de este trabajo se ha verificado por los resultados obtenidos**; y (2) aunque el algoritmo de Voting propuesto muestra en general el mejor rendimiento, esto no es cierto para todos los escenarios específicos. A este respecto, se puede observar que otros algoritmos podrían ser más convenientes en contextos particulares.

Sin embargo, para la arquitectura Proteo quedan dos aspectos que deben ser abordados: (1) la sincronización de las réplicas del servicio; y (2) la escalabilidad del sistema.

La sincronización del recurso replicado (servicio o dato), es una cuestión interesante que la mayoría de los enfoques actualmente existentes tampoco abordan. Sin embargo, el incremento en el número de réplicas del recurso implica un incremento en mensajes de sincronización. Esto podría invalidar algunas de las propuestas existentes, las cuales se basan en una replicación intensiva del recurso para incrementar su disponibilidad. A este respecto, contamos ya con trabajo en desarrollo, en el cual se ha propuesto una plataforma de servicios. Dicha plataforma tiene como objetivo proporcionar a los ingenieros de software una base para la gestión consistente y correcta de los recursos replicados/distribuidos en sistemas ubicuos colaborativos. Esto se ha conseguido a través de un servicio de *Sincronización* y otro de *Monitorización*. Estos servicios se pretenden incluir en la arquitectura Proteo para proporcionar una solución completa que considere también la sincronización de las réplicas. La plataforma propuesta se ha aplicado a la herramienta FLERSA, ideada para transformar un *"Sistema de Gestor de Contenidos"* (CMS) en su equivalente semántico, permitiendo la anotación semántica colaborativa de recursos web. Como resultado, la herramienta FLERSA se completa con capacidades como escalabilidad, interoperabilidad y agilidad comercial.

Con respecto a la escalabilidad del sistema, se ha llevado a cabo un estudio para conocer el número de nodos que puede manejar Proteo, utilizando el algoritmo de Consenso, sin sufrir degradación en su rendimiento. Además, este estudio pretende comprobar si la fiabilidad de TCP puede resultar en un mejor funcionamiento del sistema, a pesar de su mayor latencia y mayor consumo de ancho de banda, en comparación con UDP. En el estudio, se concluye que, bajo el algoritmo de Consenso, la arquitectura Proteo demuestra operar correctamente en redes compuestas por 16-18 nodos. Por tanto, para mejorar su escalabilidad en sistemas ad-hoc mayores es necesario aplicar técnicas de agrupamiento ("clustering"). A este respecto, la incorporación de diferentes técnicas de agrupamiento se puede llevar a cabo de forma transparente a la propuesta actual. El algoritmo de elección es aplicable tanto en particiones físicas como en particiones lógicas, y el agrupamiento y partición de la red se llevaría a cabo en capas inferiores al proceso de elección y despliegue.

10.2 Trabajo Futuro

Además de abordar completamente la sincronización de las réplicas y la escalabilidad de la arquitectura Proteo, existen líneas adicionales de investigación que deberían ser consideradas en el futuro:

- En relación a los comportamientos predictivos, los algoritmos de predicción de particiones son de considerable interés en el contexto de este trabajo. Generalmente, estos enfoques monitorizan el movimiento de los nodos o el estado de los enlaces para predecir la desconexión de un grupo de nodos. Esto permite sincronizar o replicar un recurso antes de que ocurra un cambio en la topología. De esta forma, el sistema pasa de ser reactivo a un sistema proactivo. Esto es útil en sistemas donde las réplicas se despliegan en tiempo de ejecución. Sin embargo, puede ser costoso y resultar en adaptaciones innecesarias si las predicciones no se realizan con determinado grado de certeza.

- Los sistemas Ubicuos, especialmente IoT, están caracterizados por estar compuestos por un largo número de dispositivos heterogéneos. En este contexto, los requisitos computacionales de los servicios desplegados pueden ser también heterogéneos y encajar mejor con las capacidades computacionales de unos dispositivos que con las capacidades de otros. Por tanto, sería de particular interés proporcionar un mecanismo semántico que permitiera definir y representar el grado de adecuación entre los requisitos de un servicio y las capacidades de un dispositivo, con el objetivo de considerar dicho grado en la evaluación de un nodo para alojar una réplica del servicio. Esto resultaría en una solución de despliegue más eficiente en entornos heterogéneos.

- A fin de extender la aplicabilidad y seguridad de la solución a otros escenarios, sería necesario incorporar mecanismos de confianza que eviten comportamientos egoístas en los nodos.

- En esta línea, la fiabilidad de un componente software es directamente dependiente de la fiabilidad de los componentes en los que se basa. En este contexto, la fiabilidad de una réplica se verá afectada por la fiabilidad del nodo en la que está desplegada. Con el propósito de incrementar la fiabilidad del esquema de despliegue proporcionada por la arquitectura, se podrían incluir métodos para medir la fiabilidad de los nodos en su evaluación. Por ejemplo, si un nodo se desconecta de forma frecuente, éste, independientemente de sus capacidades computacionales, no es un buen nodo para alojar la réplica activa del servicio.

- Finalmente, como se concluyó a raíz de la evaluación de la arquitectura Proteo, aunque el algoritmo de Votación muestra el mejor rendimiento en general, esto no es así para todos los casos específicos. Esto se puede entender como que no existe una solución universal que muestre el mejor rendimiento para todas las situaciones posibles. A este respecto, proporcionar un nuevo nivel de auto-adaptación resultaría de gran interés. Esto es, por ejemplo, permitir a la arquitectura autonómicamente utilizar un algoritmo de elección u otro de acuerdo al estado particular del entorno en ese momento. Esto podría extenderse a otros elementos, como la función de utilidad. De esta forma, Proteo podría ser capaz de auto-adaptarse a los posibles requisitos cambiantes de un escenario a lo largo del tiempo. Sin embargo, esto introduce nuevos desafíos. Proteo opera sobre un sistema distribuido, sin una entidad de control central y bajo un entorno no fiable de comunicación. Si un nodo decide cambiar el algoritmo de elección, los otros nodos deben hacerlo también. Esto se debe a que la coordinación entre ellos no es posible cuando usan algoritmos de elección diferentes.

References

[1] Abolhasan, M., Wysocki, T., and Dutkiewicz, E. A review of routing protocols for mobile ad hoc networks. *Ad Hoc Networks*, 2(1):1 – 22, 2004. ISSN 1570-8705. doi:https://doi.org/10.1016/S1570-8705(03)00043-X.

[2] Ahmed, A., Yasumoto, K., Shibata, N., Kitani, T., and Ito, M. DAR: Distributed Adaptive Service Replication for MANETs. In *2009 IEEE International Conference on Wireless and Mobile Computing, Networking and Communications*, pages 91–97, Marrakech, oct 2009. IEEE. ISBN 978-0-7695-3841-9. doi:10.1109/WiMob.2009.25.

[3] Ahmed, A., Yasumoto, K., Ito, M., Shibata, N., and Kitani, T. HDAR: Highly Distributed Adaptive Service Replication for MANETs. *IEICE Transactions on Information and Systems*, E94-D(1):91–103, 2011. ISSN 0916-8532. doi:10.1587/transinf.E94.D.91.

[4] Akbari Torkestani, J. and Meybodi, M. R. A mobility-based cluster formation algorithm for wireless mobile ad-hoc networks. *Cluster Computing*, 14(4):311–324, dec 2011. doi:10.1007/s10586-011-0161-z.

[5] Akyildiz, I. F., Wang, X., and Wang, W. Wireless mesh networks: a survey. *Computer Networks*, 47(4):445 – 487, 2005. ISSN 1389-1286. doi:https://doi.org/10.1016/j.comnet.2004.12.001.

[6] Amato, F., Fasolino, A. R., Mazzeo, A., Moscato, V., Picariello, A., Romano, S., and Tramontana, P. Ensuring semantic interoperability for e-health applications. In *2011 International Conference on Complex, Intelligent, and Software Intensive Systems*, pages 315–320, June 2011. doi:10.1109/CISIS.2011.52.

[7] Arantes, L. and Sopena, J. Easily Rendering Token-Ring Algorithms of Distributed and Parallel Applications Fault Tolerant, year=2013. In *2013 25th International Symposium on Computer Architecture and High Performance Computing*, pages 206–213, Oct . doi:10.1109/SBAC-PAD.2013.11.

[8] Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., and Zaharia, M. A View of Cloud Computing. *Commun. ACM*, 53(4):50–58, 2010. ISSN 0001-0782. doi:10.1145/1721654.1721672.

[9] Aschenbruck, N., Ernst, R., Gerhards-Padilla, E., and Schwamborn, M. BonnMotion: A Mobility Scenario Generation and Analysis Tool. In *Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques*, SIMUTools '10, pages 51:1–51:10, ICST, Brussels, Belgium, Belgium, 2010. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering). ISBN 978-963-9799-87-5. doi:10.4108/ICST.SIMUTOOLS2010.8684.

[10] Attiya, H. and Welch, J. *Distributed computing: fundamentals, simulations, and advanced topics*. John Wiley & Sons, 2nd edition, 2004. ISBN 0471453242.

[11] Atzori, L., Iera, A., and Morabito, G. The Internet of Things: A survey. *Computer Networks*, 54(15):2787 – 2805, 2010. ISSN 1389-1286. doi:https://doi.org/10.1016/j.comnet.2010.05.010.

[12] Avizienis, A., Laprie, J. C., Randell, B., and Landwehr, C. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1):11–33, Jan 2004. ISSN 1545-5971. doi:10.1109/TDSC.2004.2.

[13] Barolli, L., Spaho, E., Ikeda, M., Kulla, E., Xhafa, F., and Younas, M. A fuzzy-based data replication system for QoS improvement in MANETs. In *Proceedings of the 10th International Conference on Advances in Mobile Computing & Multimedia - MoMM '12*, page 224, New York, New York, USA, 2012. ACM Press. ISBN 9781450313070. doi:10.1145/2428955.2428997.

[14] Baxter, S. and Vogt, L. C. Content Management System, March 12 2002. US Patent 6,356,903.

[15] Becker, S., Brogi, A., Gorton, I., Overhage, S., Romanovsky, A., and Tivoli, M. Towards an Engineering Approach to Component Adaptation. In Reussner, R. H., Stafford, J. A., and Szyperski, C. A., editors, *Architecting Systems with Trustworthy Components*, pages 193–215, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg. ISBN 978-3-540-35833-6. doi:10.1007/11786160_11.

[16] Bell, W. H., Cameron, D. G., Millar, A. P., Capozza, L., Stockinger, K., and Zini, F. OptorSim: A Grid Simulator for Studying Dynamic Data Replication Strategies. *The International Journal of High Performance Computing Applications*, 17(4):403–416, 2003. doi:10.1177/10943420030174005.

[17] Bellavista, P., Corradi, A., and Magistretti, E. REDMAN: An optimistic replication middleware for read-only resources in dense MANETs. *Pervasive and Mobile Computing*, 1 (3):279–310, sep 2005. ISSN 15741192. doi:10.1016/j.pmcj.2005.06.002.

[18] Benkaouha, H., Abdelli, A., Ben-Othman, J., Zaffoune, Y., and Mokdad, L. Distributed implementation of a stable storage for MANET checkpointing protocols. In *2016 International Wireless Communications and Mobile Computing Conference (IWCMC)*, pages 672–677. IEEE, sep 2016. ISBN 978-1-5090-0304-4. doi:10.1109/IWCMC.2016.7577137.

[19] Bhatt, M., Rahayu, W., Soni, S. P., and Wouters, C. Ontology Driven Semantic Profiling and Retrieval in Medical Information Systems. *Web Semant.*, 7(4):317–331, December 2009. ISSN 1570-8268. doi:10.1016/j.websem.2009.05.004.

[20] Bieberstein, N., Bose, S., Walker, L., and Lynch, A. Impact of service-oriented architecture on enterprise systems, organizational structures, and individuals. *IBM Systems Journal*, 44(4):691–708, 2005. ISSN 0018-8670. doi:10.1147/sj.444.0691.

[21] BonnMotion. Website. http://sys.cs.uos.de/bonnmotion/. Accessed: 21/04/2018.

[22] Borgia, E. The Internet of Things vision: Key features, applications and open issues. *Computer Communications*, 54:1 – 31, 2014. ISSN 0140-3664. doi:10.1016/j.comcom.2014.09.008.

[23] Bose, J., Hahn, K., Scholz, M., Schweppe, H., and Voisard, A. Using Moving Object Databases to Provide Context Information in Mobile Ad-hoc Networks. In *7th International Conference on Mobile Data Management (MDM'06)*, pages 75–75, May 2006. doi:10.1109/MDM.2006.164.

[24] Brandner, G., Bettstetter, C., and Schilcher, U. Contention-based node selection with applications to relay communications and load balancing. *EURASIP Journal on Wireless Communications and Networking*, 2013(1):211, dec 2013. ISSN 1687-1499. doi:10.1186/1687-1499-2013-211.

[25] Camp, T., Boleng, J., and Davies, V. A survey of mobility models for ad hoc network research. *Special Issue: Mobile Ad Hoc Networking – Research, Trends and Applications*, 2:483–502, 2002.

[26] Canal, C., Murillo, J. M., and Poizat, P. Coordination and Adaptation Techniques for Software Entities. In Malenfant, J. and Østvold, B. M., editors, *Object-Oriented Technology. ECOOP 2004 Workshop Reader*, pages 133–147, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. ISBN 978-3-540-30554-5.

[27] Caragliu, A., Bo, C. D., and Nijkamp, P. Smart Cities in Europe. *Journal of Urban Technology*, 18(2):65–82, 2011. doi:10.1080/10630732.2011.601117.

[28] Cattell, R. Scalable SQL and NoSQL Data Stores. *SIGMOD Rec.*, 39(4):12–27, May 2011. ISSN 0163-5808. doi:10.1145/1978915.1978919.

[29] Chandrakala., C. B., Prema., K. V., and Hareesha., K. S. Improved data availability and fault tolerance in manet by replication. In *2013 3rd IEEE International Advance Computing Conference (IACC)*, pages 324–329, Feb 2013. doi:10.1109/IAdCC.2013.6514244.

[30] Chang, E. and Roberts, R. An improved algorithm for decentralized extrema-finding in circular configurations of processes. *Commun. ACM*, 22(5):281–283, 1979. ISSN 0001-0782. doi:10.1145/359104.359108.

[31] Chang, W. Y., Abu-Amara, H., and Sanford, J. F. *Transforming Enterprise Cloud Services*. Springer Publishing Company, Incorporated, 2014. ISBN 9400790066, 9789400790063.

[32] Chen, B. and Cheng, H. H. A Review of the Applications of Agent Technology in Traffic and Transportation Systems. *IEEE Transactions on Intelligent Transportation Systems*, 11(2):485–497, June 2010. ISSN 1524-9050. doi:10.1109/TITS.2010.2048313.

[33] Chlamtac, I., Conti, M., and Liu, J. J.-N. Mobile ad hoc networking: imperatives and challenges. *Ad Hoc Networks*, 1(1):13 – 64, 2003. ISSN 1570-8705. doi:https://doi.org/10.1016/S1570-8705(03)00013-1.

[34] Choudhury, P., Sarkar, A., and Debnath, N. C. Deployment of Service Oriented architecture in MANET: A research roadmap. In *2011 9th IEEE International Conference on Industrial Informatics*, pages 666–670, 2011. doi:10.1109/INDIN.2011.6034957.

[35] Chu, F. Reducing Ω to ◊W. *Information Processing Letters*, 67(6):289–293, sep 1998. ISSN 00200190. doi:10.1016/S0020-0190(98)00122-7.

[36] Clements, P. C. A Survey of Architecture Description Languages. In *Proceedings of the 8th International Workshop on Software Specification and Design*, IWSSD '96, pages 16–. IEEE Computer Society, 1996. ISBN 0-8186-7361-3. URL http://dl.acm.org/citation.cfm?id=857204.858261.

[37] Cohen, E. and Shenker, S. Replication strategies in unstructured peer-to-peer networks. *ACM SIGCOMM Computer Communication Review*, 32(4):177, 2002. ISSN 01464833. doi:10.1145/964725.633043.

[38] Computer Systems Research Group. UNIX user's manual: supplementary documents: 4.2 Berkeley software distribution: virtual VAX-11 version, 1984. Computer Science Division, Dept. of Electrical Engineering and Computer Science, University of California, Berkeley, Calif.

[39] Conti, M., Maselli, G., Turi, G., and Giordano, S. Cross-layering in mobile ad hoc network design. *Computer*, 37(2):48–51, 2004. ISSN 0018-9162. doi:10.1109/MC.2004.1266295.

[40] Cook, D. J., Augusto, J. C., and Jakkula, V. R. Ambient intelligence: Technologies, applications, and opportunities. *Pervasive and Mobile Computing*, 5(4):277 – 298, 2009. ISSN 1574-1192. doi:https://doi.org/10.1016/j.pmcj.2009.04.001.

[41] Cornuéjols, G., Nemhauser, G. L., and Wolsey, L. A. The uncapacitated facility location problem. Technical report, Carnegie Mellon University, Management Science Research Group, 1983.

[42] Coulouris, G. F., Dollimore, J., and Kindberg, T. *Distributed systems: concepts and design*. Addison-Wesley, 5th edition, 2011. ISBN 7133427729.

[43] Cuka, M., Elmazi, D., Ozera, K., Oda, T., and Barolli, L. Selection of Actor Nodes in Opportunistic Networks: A Fuzzy-Based Approach. In *2017 IEEE 31st International Conference on Advanced Information Networking and Applications (AINA)*, pages 278–284. IEEE, mar 2017. ISBN 978-1-5090-6029-0. doi:10.1109/AINA.2017.118.

[44] Davidson, S. B., Garcia-Molina, H., and Skeen, D. Consistency in a Partitioned Network: A Survey. *ACM Comput. Surv.*, 17(3):341–370, September 1985. ISSN 0360-0300. doi:10.1145/5505.5508.

[45] Derhab, A. and Badache, N. A pull-based service replication protocol in mobile ad hoc networks. *Transactions on Emerging Telecommunications Technologies*, 18(1):1–11, 2007. doi:10.1002/ett.1080.

[46] Dey, A. K. Understanding and Using Context. *Personal Ubiquitous Comput.*, 5(1):4–7, January 2001. ISSN 1617-4909. doi:10.1007/s007790170019.

[47] Dijkstra, E. W., Feijen, W. H. J., and van Gasteren, A. J. M. Derivation of a termination detection algorithm for distributed computations. In Broy, M., editor, *Control Flow and Data Flow: Concepts of Distributed Programming*, pages 507–512. Springer Berlin Heidelberg, 1986. ISBN 978-3-642-82921-5.

[48] Dogdu, E. Semantic web in ehealth. In *Proceedings of the 47th Annual Southeast Regional Conference*, ACM-SE 47, pages 73:1–73:4, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-421-8. doi:10.1145/1566445.1566542.

[49] Dustdar, S. and Juszczyk, L. Dynamic replication and synchronization of web services for high availability in mobile ad-hoc networks. *Service Oriented Computing and Applications*, 1(1):19–33, apr 2007. ISSN 1863-2386. doi:10.1007/s11761-007-0006-z.

[50] Erl, T. *SOA: Principles of Service Design*. Prentice Hall, 1st edition, 2007. ISBN 978-0132344821.

[51] Erl, T. *SOA design patterns*. Prentice Hall PTR, 1st edition, 2008. ISBN 978-0136135166.

[52] Fernando, N., Loke, S. W., and Rahayu, W. Mobile cloud computing: A survey. *Future Generation Computer Systems*, 29(1):84 – 106, 2013. ISSN 0167-739X. doi:10.1016/j.future.2012.05.023.

[53] Garcia-Molina, H. Elections in a distributed computing system. *IEEE Transactions on Computers*, C-31(1):48–59, 1982. ISSN 0018-9340. doi:10.1109/TC.1982.1675885.

[54] Guerraoui, R. and Schiper, A. Software-based replication for fault tolerance. *Computer*, 30(4):68–74, Apr 1997. ISSN 0018-9162. doi:10.1109/2.585156.

[55] Guerrero-Contreras, G., Garrido, J. L., Balderas-Díaz, S., and Rodríguez-Domínguez, C. A context-aware architecture supporting service availability in mobile cloud computing. *IEEE Transactions on Services Computing*, 10(6):956–968, Nov 2017. ISSN 1939-1374. doi:10.1109/TSC.2016.2540629.

[56] Guerrero-Contreras, G., Navarro-Galindo, J. L., Samos, J., and Garrido, J. L. A collaborative semantic annotation system in health: towards a SOA design for knowledge sharing in ambient intelligence. *Mobile Information Systems*, 2017, 2017. doi:10.1155/2017/4759572.

[57] Gupta, I., Renesse, R. v., and Birman, K. P. A Probabilistically Correct Leader Election Protocol for Large Groups. In *Proceedings of the 14th International Conference on Distributed Computing*, DISC '00, pages 89–103, London, UK, UK, 2000. Springer-Verlag. ISBN 3-540-41143-7.

[58] Hamdy, M. and König-Ries, B. A service distribution protocol for mobile ad hoc networks. In *Proceedings of the 5th international conference on Pervasive services - ICPS '08*, page 141, New York, New York, USA, 2008. ACM Press. ISBN 9781605581354. doi:10.1145/1387269.1387293.

[59] Hamdy, M. and König-Ries, B. The Service Distribution Protocol for MANETs - Criteria and Performance Analysis. In Kim T., F. W. C., editor, *Communications in Computer and Information Science*, volume 48, pages 467–479. Springer Berlin Heidelberg, 2009. ISBN 9783642051968. doi:10.1007/978-3-642-05197-5_34.

[60] Hamdy, M., Derhab, A., and König-Ries, B. A Comparison on MANETs' Service Replication Schemes: Interest versus Topology Prediction. In Özcan, A., Chaki, N., and Nagamalai, D., editors, *Communications in Computer and Information Science*, volume 84 of *Communications in Computer and Information Science*, pages 202–216. Springer Berlin Heidelberg, Ankara, 2010. ISBN 9783642141706. doi:10.1007/978-3-642-14171-3_17.

[61] Hamed, T., Ernst, J. B., and Kremer, S. C. *A Survey and Taxonomy of Classifiers of Intrusion Detection Systems*, pages 21–39. Springer International Publishing, Cham, 2018. doi:10.1007/978-3-319-58424-9_2.

[62] Hara, T. Replica Allocation Methods in Ad Hoc Networks with Data Update. *Mobile Networks and Applications*, 8(4):343–354, 2003. ISSN 1383469X. doi:10.1023/A:1024523411884.

[63] Hara, T. and Madria, S. K. Consistency Management Strategies for Data Replication in Mobile Ad Hoc Networks. *IEEE Transactions on Mobile Computing*, 8(7):950–967, jul 2009. ISSN 1536-1233. doi:10.1109/TMC.2008.150.

[64] Herrmann, K. Self-organized Service Placement in Ambient Intelligence Environments. *ACM Trans. Auton. Adapt. Syst.*, 5(2):6:1–6:39, May 2010. ISSN 1556-4665. doi:10.1145/1740600.1740602.

[65] Hill, M. D. What is Scalability? *SIGARCH Comput. Archit. News*, 18(4):18–21, December 1990. ISSN 0163-5964. doi:10.1145/121973.121975.

[66] Hirsch, D. and Madria, S. A Resource-Efficient Adaptive Caching Scheme for Mobile Ad Hoc Networks. In *2010 29th IEEE Symposium on Reliable Distributed Systems*, pages 64–71, New Delhi, oct 2010. IEEE. ISBN 978-0-7695-4250-8. doi:10.1109/SRDS.2010.16.

[67] Hirschfeld, R., Costanza, P., and Nierstrasz, O. M. Context-oriented programming. *Journal of Object technology*, 7(3):125–151, 2008. doi:10.5381/jot.2008.7.3.a4.

[68] Hong, X., Gerla, M., Pei, G., and Chiang, C.-C. A group mobility model for ad hoc wireless networks. In *Proceedings of the 2Nd ACM International Workshop on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, MSWiM '99, pages 53–60. ACM, 1999. ISBN 1-58113-173-9. doi:10.1145/313237.313248.

[69] Horizon 2020. Work Programme 2018-2020 - Secure societies - Protecting freedom and security of Europe and its citizens. http://ec.europa.eu/research/participants/data/ref/h2020/wp/2018-2020/main/h2020-wp1820-security_en.pdf. Accessed: 16/05/2018.

[70] Hossain, E., Chow, G., Leung, V. C., McLeod, R. D., Mišić, J., Wong, V. W., and Yang, O. Vehicular telematics over heterogeneous wireless networks: A survey. *Computer Communications*, 33(7):775 – 793, 2010. ISSN 0140-3664. doi:https://doi.org/10.1016/j.comcom.2009.12.010.

[71] Huerta-Canepa, G. and Lee, D. A Virtual Cloud Computing Provider for Mobile Devices. In *Proceedings of the 1st ACM Workshop on Mobile Cloud Computing &#38; Services: Social Networks and Beyond*, MCS '10, pages 6:1–6:5, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0155-8. doi:10.1145/1810931.1810937.

[72] Hunkeler, U., Truong, H. L., and Stanford-Clark, A. MQTT-S - A publish/subscribe protocol for Wireless Sensor Networks. In *Communication Systems Software and Middleware and Workshops, 2008. COMSWARE 2008. 3rd International Conference on*, pages 791–798, Jan 2008. doi:10.1109/COMSWA.2008.4554519.

[73] IBM. An architectural blueprint for Autonomic Computing. *IBM White Paper*, 31:1–6, 2006.

[74] IGI Global. Availability definition. https://www.igi-global.com/dictionary/service-availability/44258. Accessed: 22/04/2018.

[75] Inaba, T., Elmazi, D., Liu, Y., Sakamoto, S., Barolli, L., and Uchida, K. Integrating Wireless Cellular and Ad-Hoc Networks Using Fuzzy Logic Considering Node Mobility and Security. In *2015 IEEE 29th International Conference on Advanced Information Networking and Applications Workshops*, pages 54–60. IEEE, mar 2015. ISBN 978-1-4799-1775-4. doi:10.1109/WAINA.2015.116.

[76] ISO/IEC 15288:2002. Systems and software engineering – System life cycle processes. https://www.iso.org/standard/27166.html. Accessed: 25/04/2018.

[77] ISO/IEC/IEEE42010:2011. Systems and software engineering — Architecture description. https://www.iso.org/standard/50508.html. Accessed: 21/04/2018.

[78] Issariyakul, T. and Hossain, E. *Introduction to Network Simulator NS2*. Springer Publishing Company, Incorporated, 2nd edition, 2011. ISBN 1461414059, 9781461414056.

[79] Jacquet, P., Muhlethaler, P., Clausen, T., Laouiti, A., Qayyum, A., and Viennot, L. Optimized link state routing protocol for ad hoc networks. In *Proceedings. IEEE International Multi Topic Conference, 2001. IEEE INMIC 2001. Technology for the 21st Century.*, pages 62–68, 2001. doi:10.1109/INMIC.2001.995315.

[80] Jae-Ho Choi, Kyu-Sun Shim, SangKeun Lee, and Kun-Lung Wu. Handling Selfishness in Replica Allocation over a Mobile Ad Hoc Network. *IEEE Transactions on Mobile Computing*, 11(2):278–291, feb 2012. ISSN 1536-1233. doi:10.1109/TMC.2011.57.

[81] JSim. Website. http://www.physiome.org/jsim/. Accessed: 21/04/2018.

[82] Kakousis, K., Paspallis, N., and Papadopoulos, G. A. A Survey of Software Adaptation in Mobile and Ubiquitous Computing. *Enterp. Inf. Syst.*, 4(4):355–389, November 2010. ISSN 1751-7575. doi:10.1080/17517575.2010.509814.

[83] Kataria, P. and Juric, R. Sharing e-Health Information through Ontological Layering. In *2010 43rd Hawaii International Conference on System Sciences*, pages 1–10, Jan 2010. doi:10.1109/HICSS.2010.338.

[84] Kephart, J. O. and Chess, D. M. The vision of Autonomic Computing. *Computer*, 36(1): 41–50, Jan 2003. ISSN 0018-9162. doi:10.1109/MC.2003.1160055.

[85] Ketfi, A., Belkhatir, N., and Cunin, P.-Y. Automatic Adaptation of Component-based Software: Issues and Experiences. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications - Volume 3*, PDPTA '02, pages 1365–1371. CSREA Press, 2002. ISBN 1-892512-89-0.

[86] Kim, S.-K., Yoon, J.-H., Lee, K.-J., Choi, J.-H., and Yang, S.-B. A scalable mobility-based replica allocation scheme in a mobile ad-hoc network. *Telecommunication Systems*, 60(2):239–250, oct 2015. ISSN 1018-4864. doi:10.1007/s11235-015-0026-5.

[87] Knodel, J. and Naab, M. *What Is Architecture Evaluation?*, pages 21–34. Springer International Publishing, Cham, 2016. ISBN 978-3-319-34177-4. doi:10.1007/978-3-319-34177-4_3.

[88] Knodel, J. and Naab, M. *How to Perform the Solution Adequacy Check (SAC)?*, pages 59–72. Springer International Publishing, Cham, 2016. doi:10.1007/978-3-319-34177-4_6.

[89] Kordafshari, M. S., Gholipour, M., Mosakhani, M., Haghighat, A. T., and Dehghan, M. Modified bully election algorithm in distributed systems. In *Proceedings of the 9th WSEAS International Conference on Computers*, ICCOMP'05, pages 10:1–10:6. World Scientific and Engineering Academy and Society (WSEAS), 2005. ISBN 960-8457-29-7.

[90] Krill, P. Make way for SOA 2.0. www.infoworld.com/t/architecture/make-way-soa-20-420, 2006. Accessed: 24/04/2018.

[91] Kumar, P. J. and Ilango, P. An Optimized Replica Allocation Algorithm Amidst of Selfish Nodes in MANET. *Wireless Personal Communications*, 94(4):2719–2738, jun 2017. ISSN 0929-6212. doi:10.1007/s11277-016-3928-y.

[92] Lacage, M. and Henderson, T. R. Yet another network simulator. In *Proceeding from the 2006 Workshop on ns-2: The IP Network Simulator*, WNS2 '06. ACM, 2006. ISBN 1-59593-508-8. doi:10.1145/1190455.1190467.

[93] Laplante, P. A., Zhang, J., and Voas, J. What's in a Name? Distinguishing between SaaS and SOA. *IT Professional*, 10(3):46–50, 2008. ISSN 1520-9202. doi:10.1109/MITP.2008.60.

[94] Leite, J., Oquendo, F., and Batista, T. SysADL: A SysML Profile for Software Architecture Description. In *Software Architecture*, pages 106–113. Springer Berlin Heidelberg, 2013. ISBN 978-3-642-39031-9. doi:10.1007/978-3-642-39031-9_9.

[95] Little, M. C. and McCue, D. L. The replica management system: a scheme for flexible and dynamic replication. In *Proceedings of 2nd International Workshop on Configurable Distributed Systems*, pages 46–57, 1994. doi:10.1109/IWCDS.1994.289936.

[96] Liu, S., Ölveczky, P. C., and Meseguer, J. Formal Analysis of Leader Election in MANETs Using Real-Time Maude. In *Software, Services, and Systems*, pages 231–252. Springer International Publishing, 2015. doi:10.1007/978-3-319-15545-6_16.

[97] Malavolta, I., Lago, P., Muccini, H., Pelliccione, P., and Tang, A. What Industry Needs from Architectural Languages: A Survey. *IEEE Transactions on Software Engineering*, 39(6):869–891, June 2013. ISSN 0098-5589. doi:10.1109/TSE.2012.74.

[98] Malpani, N., Welch, J. L., and Vaidya, N. 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 - DIALM '00*,

pages 96–103, New York, New York, USA, 2000. ACM Press. ISBN 1581133014. doi:10.1145/345848.345871.

[99] Marques, A., Mira da Silva, F., and Rocha, R. P2P over Mobile Ad-hoc Networks. In *2009 6th IEEE Annual Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks Workshops*, pages 1–3. IEEE, jun 2009. ISBN 978-1-4244-3938-6. doi:10.1109/SAHCNW.2009.5172952.

[100] Meier, R. Communication Paradigms for Mobile Computing. *SIGMOBILE Mob. Comput. Commun. Rev.*, 6(4):56–58, October 2002. ISSN 1559-1662. doi:10.1145/643550.643555.

[101] Michelson, B. M. Event-driven architecture overview. *Patricia Seybold Group*, 2, 2006.

[102] Miorandi, D., Sicari, S., Pellegrini, F. D., and Chlamtac, I. Internet of things: Vision, applications and research challenges. *Ad Hoc Networks*, 10(7):1497 – 1516, 2012. ISSN 1570-8705. doi:10.1016/j.adhoc.2012.02.016.

[103] Mohammed, N., Otrok, H., Wang, L., Debbabi, M., and Bhattacharya, P. Mechanism Design-Based Secure Leader Election Model for Intrusion Detection in MANET. *IEEE Transactions on Dependable and Secure Computing*, 8(1):89–103, jan 2011. ISSN 1545-5971. doi:10.1109/TDSC.2009.22.

[104] Mohler, B. J., Thompson, W. B., Creem-Regehr, S. H., Pick, H. L., and Warren, W. H. Visual flow influences gait transition speed and preferred walking speed. *Experimental Brain Research*, 181(2):221–228, 2007. ISSN 1432-1106. doi:10.1007/s00221-007-0917-0. URL https://doi.org/10.1007/s00221-007-0917-0.

[105] Muldoon, C., O'Hare, G. M., O'Grady, M. J., Tynan, R., and Trigoni, N. Distributed constraint optimisation for resource limited sensor networks. *Science of Computer Programming*, 78(5):583 – 593, 2013. ISSN 0167-6423. doi:https://doi.org/10.1016/j.scico.2012.10.005.

[106] Nadareishvili, I., Mitra, R., McLarty, M., and Amundsen, M. *Microservice Architecture: Aligning Principles, Practices, and Culture*. " O'Reilly Media, Inc.", 2016. ISBN 978-1491956250.

[107] Navarro-Galindo, J. L. and Samos, J. The FLERSA tool: adding semantics to a web content management system. *International Journal of Web Information Systems*, 8(1): 73–126, 2012. doi:10.1108/17440081211222609.

[108] Nayyar, A. and Singh, R. A Comprehensive Review of Simulation Tools for Wireless Sensor Networks (WSNs). *Journal of Wireless Networking and Communications*, 5(1): 19–47, 2015. ISSN 2167-7336. doi:10.5923/j.jwnc.20150501.03.

[109] NetSim. Website. http://www.boson.com/netsim-cisco-network-simulator. Accessed: 21/04/2018.

[110] Neyem, A., Ochoa, S. F., Pino, J. A., and Franco, R. D. A reusable structural design for mobile collaborative applications. *Journal of Systems and Software*, 85(3):511 – 524, 2012. ISSN 0164-1212. doi:https://doi.org/10.1016/j.jss.2011.05.046.

[111] Noguera, M., Hurtado, M. V., Rodríguez, M. L., Chung, L., and Garrido, J. L. Ontology-driven analysis of UML-based collaborative processes using OWL-DL and CPN. *Science of Computer Programming*, 75(8):726 – 760, 2010. ISSN 0167-6423. doi:https://doi.org/10.1016/j.scico.2009.05.002.

[112] ns-3. Software Architecture. https://www.nsnam.org/docs/design.pdf, . Accessed: 26/04/2018.

[113] ns-3. Detailed description of Two-Ray Ground propagation loss model. https://www.nsnam.org/doxygen/classns3_1_1_two_ray_ground_propagation_loss_model.html#details, . Accessed: 26/04/2018.

[114] ns-3. Website. https://www.nsnam.org/, . Accessed: 21/04/2018.

[115] O'Brien, L., Merson, P., and Bass, L. Quality Attributes for Service-Oriented Architectures. In *Proceedings of the International Workshop on Systems Development in SOA Environments*, SDSOA '07, page 3, Washington, DC, USA, 2007. IEEE Computer Society. ISBN 0-7695-2960-7. doi:10.1109/SDSOA.2007.10.

[116] O'Grady, M. J., Muldoon, C., Dragone, M., Tynan, R., and O'Hare, G. M. P. Towards evolutionary ambient assisted living systems. *Journal of Ambient Intelligence and Humanized Computing*, 1(1):15–29, Mar 2010. ISSN 1868-5145. doi:10.1007/s12652-009-0003-5.

[117] OMNeT++. Website. https://www.omnetpp.org/. Accessed: 21/04/2018.

[118] OPNET. Website. https://www.riverbed.com/gb/products/steelcentral/opnet.html. Accessed: 21/04/2018.

[119] Oquendo, F., Leite, J., and Batista, T. Specifying Architecture Behavior with SysADL. In *2016 13th Working IEEE/IFIP Conference on Software Architecture (WICSA)*, pages 140–145, 2016. doi:10.1109/WICSA.2016.40.

[120] Oquendo, F. pi-ADL: An Architecture Description Language Based on the Higher-order Typed pi-calculus for Specifying Dynamic and Mobile Software Architectures. *SIGSOFT Softw. Eng. Notes*, 29(3):1–14, May 2004. ISSN 0163-5948. doi:10.1145/986710.986728.

[121] Oquendo, F., Leite, J., and Batista, T. Executing Software Architecture Descriptions with SysADL. In *Software Architecture*, pages 129–137. Springer International Publishing, 2016. ISBN 978-3-319-48992-6. doi:10.1007/978-3-319-48992-6_9.

[122] Organisation for Economic Co-operation and Development. OECD Glossary of Statistical Terms - Ubiquitous Computing Definition. https://stats.oecd.org/glossary/detail.asp?ID=6093, 2003. Accessed: 15/06/2018.

[123] Oxford Dictionary. Efficient definition. https://en.oxforddictionaries.com/definition/efficient, . Accessed: 01/05/2018.

[124] Oxford Dictionary. Efficiency definition. https://en.oxforddictionaries.com/definition/efficiency, . Accessed: 01/05/2018.

[125] Pandey, R. K. Architectural Description Languages (ADLs) vs UML: A Review. *SIGSOFT Softw. Eng. Notes*, 35(3):1–5, 2010. ISSN 0163-5948. doi:10.1145/1764810.1764828.

[126] Papazoglou, M. P., Traverso, P., Dustdar, S., and Leymann, F. Service-Oriented Computing: State of the Art and Research Challenges. *Computer*, 40(11):38–45, Nov 2007. ISSN 0018-9162. doi:10.1109/MC.2007.400.

[127] Park, S. A Safe Election Protocol based on an Unreliable Failure Detector in Distributed Systems. *Indian Journal of Science and Technology*, 8(34), 2015. ISSN 0974-5645. doi:10.17485/ijst/2015/v8i34/86665.

[128] Park, V. D. and Corson, M. S. A highly adaptive distributed routing algorithm for mobile wireless networks. In *INFOCOM '97. Sixteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Driving the Information Revolution., Proceedings IEEE*, volume 3, pages 1405–1413, 1997. doi:10.1109/INFCOM.1997.631180.

[129] Pei, G., Gerla, M., and Chen, T.-W. Fisheye state routing: a routing scheme for ad hoc wireless networks. In *2000 IEEE International Conference on Communications. ICC 2000. Global Convergence Through Communications. Conference Record*, volume 1, pages 70–74, 2000. doi:10.1109/ICC.2000.853066.

[130] Perkins, C. E. and Bhagwat, P. Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers. *SIGCOMM Comput. Commun. Rev.*, 24(4):234–244, 1994. ISSN 0146-4833. doi:10.1145/190809.190336. URL http://doi.acm.org/10.1145/190809.190336.

[131] Preuveneers, D., Yasar, A.-U.-H., and Berbers, Y. Architectural styles for opportunistic mobile communication: requirements and design patterns. In *Mobility Conference*, 2008. doi:10.1145/1506270.1506326.

[132] Psannis, K., Xinogalos, S., and Sifaleras, A. Convergence of Internet of things and mobile cloud computing. *Systems Science & Control Engineering*, 2(1):476–483, 2014. doi:10.1080/21642583.2014.913213.

[133] QualNet. Website. https://web.scalable-networks.com/qualnet-network-simulator-software. Accessed: 21/04/2018.

[134] Raychoudhury, V., Cao, J., Niyogi, R., Wu, W., and Lai, Y. Top k-leader election in mobile ad hoc networks. *Pervasive and Mobile Computing*, 13:181–202, aug 2014. ISSN 15741192. doi:10.1016/j.pmcj.2013.10.003.

[135] Rodríguez-Domínguez, C., Benghazi, K., Noguera, M., Garrido, J. L., Rodríguez, M. L., and Ruiz-López, T. A Communication Model to Integrate the Request-Response and the Publish-Subscribe Paradigms into Ubiquitous Systems. *Sensors*, 12(6):7648–7668, 2012. ISSN 1424-8220. doi:10.3390/s120607648.

[136] Roman, G. C., Julien, C., and Huang, Q. Network abstractions for context-aware mobile computing. In *Proceedings of the 24th International Conference on Software Engineering. ICSE 2002*, pages 363–373, May 2002. doi:10.1145/581384.581385.

[137] Russell, S. J. and Norvig, P. *Artificial Intelligence: A Modern Approach*. Pearson, 3rd edition, 2009. ISBN 978-0136042594.

[138] Ryu, B.-G., Choi, J.-H., and Lee, S. Impact of node distance on selfish replica allocation in a mobile ad-hoc network. *Ad Hoc Networks*, 11(8):2187–2202, nov 2013. doi:10.1016/j.adhoc.2013.05.001.

[139] Ryu, B.-G., Ryu, W.-J., Lee, Y.-K., and Lee, S. Selfish replica allocation in a mobile ad hoc network with data update. In *2015 International Conference on Big Data and Smart Computing (BIGCOMP)*, pages 142–149. IEEE, feb 2015. ISBN 978-1-4799-7303-3. doi:10.1109/35021BIGCOMP.2015.7072824.

[140] Sabat, S. and Kadam, S. Adaptive Energy aware reputation based leader election for IDS in MANET. In *2014 International Conference on Communication and Signal Processing*, pages 891–894. IEEE, apr 2014. ISBN 978-1-4799-3358-7. doi:10.1109/ICCSP.2014.6949972.

[141] Salehie, M. and Tahvildari, L. Self-adaptive Software: Landscape and Research Challenges. *ACM Trans. Auton. Adapt. Syst.*, 4(2):14:1–14:42, 2009. ISSN 1556-4665. doi:10.1145/1516533.1516538.

[142] Satyanarayanan, M. Fundamental Challenges in Mobile Computing. In *Proceedings of the Fifteenth Annual ACM Symposium on Principles of Distributed Computing*, PODC '96, pages 1–7, New York, NY, USA, 1996. ACM. ISBN 0-89791-800-2. doi:10.1145/248052.248053.

[143] Satyanarayanan, M. Pervasive computing: vision and challenges. *IEEE Personal Communications*, 8(4):10–17, Aug 2001. ISSN 1070-9916. doi:10.1109/98.943998.

[144] Schmidt, A., Beigl, M., and Gellersen, H.-W. There is more to context than location. *Computers & Graphics*, 23(6):893 – 901, 1999. ISSN 0097-8493. doi:10.1016/S0097-8493(99)00120-X.

[145] Seo, C. and Zeigler, B. P. Devs namespace for interoperable devs/soa. In *Proceedings of the 2009 Winter Simulation Conference (WSC)*, pages 1311–1322, Dec 2009. doi:10.1109/WSC.2009.5429701.

[146] Shannon, R. E. Introduction to the Art and Science of Simulation. In *Proceedings of the 30th Conference on Winter Simulation*, WSC '98, pages 7–14, Los Alamitos, CA, USA, 1998. IEEE Computer Society Press. ISBN 0-7803-5134-7.

[147] Shelby, Z., Hartke, K., and Bormann, C. The constrained application protocol (CoAP). 2014.

[148] Shepherd, C. Theoretical design of primary and secondary cells. Part 3 - battery discharge equation. Technical report, NAVAL RESEARCH LAB WASHINGTON DC, 1963.

[149] Shi, K. and Chen, H. RHPMAN: Replication in Highly Partitioned Mobile Ad Hoc Networks. *International Journal of Distributed Sensor Networks*, 10(6):819372, jun 2014. ISSN 1550-1477. doi:10.1155/2014/819372.

[150] Shila, D. M., Shen, W., Cheng, Y., Tian, X., and a. X. S. Shen. AMCloud: Toward a Secure Autonomic Mobile Ad Hoc Cloud Computing System. *IEEE Wireless Communications*, 24(2):74–81, April 2017. ISSN 1536-1284. doi:10.1109/MWC.2016.1500119RP.

[151] Sommer, N. L. Service Provision in Disconnected Mobile Ad Hoc Networks. In *Mobile Ubiquitous Computing, Systems, Services and Technologies, 2007. UBICOMM '07. International Conference on*, pages 125–130, Nov 2007. doi:10.1109/UBICOMM.2007.24.

[152] Sørensen, C.-F., Wu, M., Sivaharan, T., Blair, G. S., Okanda, P., Friday, A., and Duran-Limon, H. A Context-aware Middleware for Applications in Mobile Ad Hoc Environments. In *Proceedings of the 2Nd Workshop on Middleware for Pervasive and Ad-hoc Computing*, MPAC '04, pages 107–110, New York, NY, USA, 2004. ACM. ISBN 1-58113-951-9. doi:10.1145/1028509.1028510.

[153] Sun, X., Zheng, J., Liu, Q., and Liu, Y. Dynamic Data Replication Based on Access Cost in Distributed Systems. *2009 Fourth International Conference on Computer Sciences and Convergence Information Technology*, pages 829–834, 2009. doi:10.1109/ICCIT.2009.198.

[154] SysML. Forum FAQ website. http://sysmlforum.com/sysml-faq/. Accessed: 21/04/2018.

[155] Szyperski, C. *Component Software: Beyond Object-oriented Programming*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1998. ISBN 0-201-17888-5.

[156] Tanenbaum, A. S. and Van Steen, M. *Distributed systems: principles and paradigms*. Prentice-Hall, 2nd edition, 2007. ISBN 153028175X.

[157] Taylor, H., Yochem, A., Phillips, L., and Martinez, F. *Event-driven architecture: how SOA enables the real-time enterprise*. Pearson Education, 2009. ISBN 978-0321322111.

[158] Thönes, J. Microservices. *IEEE Software*, 32(1):116–116, 2015. ISSN 0740-7459. doi:10.1109/MS.2015.11.

[159] Tortorella, M. *Reliability, maintainability, and supportability: best practices for systems engineers*. John Wiley & Sons, 2015. ISBN 9781118858882. doi:10.1002/9781119058823.

[160] Tremblay, O., Dessaint, L. A., and Dekkiche, A. I. A generic battery model for the dynamic simulation of hybrid electric vehicles. In *2007 IEEE Vehicle Power and Propulsion Conference*, pages 284–289, 2007. doi:10.1109/VPPC.2007.4544139.

[161] Tsuchiya, P. F. The landmark hierarchy: A new hierarchy for routing in very large networks. *SIGCOMM Comput. Commun. Rev.*, 18(4):35–42, 1988. ISSN 0146-4833. doi:10.1145/52325.52329.

[162] Universal Mobile Telecommunicatios System (UMTS). Selection procedures for the choice of radio transmission technologies of the UMTS. Technical report, European Telecommunications Standards Institute (ETSI), 1998.

[163] Vasudevan, S., DeCleene, B., Immerman, N., Kurose, J., and Towsley, D. Leader election algorithms for wireless ad hoc networks. In *Proceedings DARPA Information Survivability Conference and Exposition*, pages 261–272. IEEE Comput. Soc, 2003. ISBN 0-7695-1897-4. doi:10.1109/DISCEX.2003.1194890.

[164] Vasudevan, S., Kurose, J., and Towsley, D. Design and analysis of a leader election algorithm for mobile ad hoc networks. In *Proceedings of the 12th IEEE International Conference on Network Protocols, 2004. ICNP 2004.*, pages 350–360. IEEE, 2004. ISBN 0-7695-2161-4. doi:10.1109/ICNP.2004.1348124.

[165] Venkanna, U. and Leela Velusamy, R. TEA-CBRP: Distributed cluster head election in MANET by using AHP. *Peer-to-Peer Networking and Applications*, 9(1):159–170, jan 2016. ISSN 1936-6442. doi:10.1007/s12083-014-0320-0.

[166] Wang, K. H. and Li, B. Efficient and guaranteed service coverage in partitionable mobile ad-hoc networks. In *Proceedings.Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 2, pages 1089–1098, 2002. doi:10.1109/INFCOM.2002.1019357.

[167] Weiser, M. The Computer for the 21 st Century. *Scientific American*, 265(3):94–105, 1991. ISSN 00368733, 19467087. URL http://www.jstor.org/stable/24938718.

[168] Weiser, M. and Brown, J. S. *The Coming Age of Calm Technology*, pages 75–85. Springer New York, New York, NY, 1997. ISBN 978-1-4612-0685-9. doi:10.1007/978-1-4612-0685-9_6.

[169] Weyns, D. and Ahmad, T. Claims and Evidence for Architecture-Based Self-adaptation: A Systematic Literature Review. In Drira, K., editor, *Software Architecture*, pages 249–265, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. ISBN 978-3-642-39031-9. doi:10.1007/978-3-642-39031-9_22.

[170] Wooldridge, M. *An introduction to multiagent systems*. John Wiley & Sons, 2nd edition, 2009. ISBN 978-0-470-51946-2.

[171] World Wide Web Consortium (W3C). RDFa Distiller. http://www.w3.org/2007/08/pyRdfa/, 2016. Accessed: 14/05/2018.

[172] Wu, W., Cao, J. J., Raynal, M., Weigang Wu, Cao, J. J., Raynal, M., Wu, W., Cao, J. J., Raynal, M., Weigang Wu, Cao, J. J., and Raynal, M. Eventual clusterer: A modular approach to designing hierarchical consensus protocols in MANETs. *IEEE Transactions on Parallel and Distributed Systems*, 20(6):753–765, jun 2009. ISSN 10459219. doi:10.1109/TPDS.2008.266.

[173] Xia, F., Ahmed, A. M., Yang, L. T., Ma, J., and Rodrigues, J. J. Exploiting Social Relationship to Enable Efficient Replica Allocation in Ad-hoc Social Networks. *IEEE Transactions on Parallel and Distributed Systems*, 25(12):3167–3176, dec 2014. ISSN 1045-9219. doi:10.1109/TPDS.2013.2295805.

[174] Xu, L., O'Grady, M. J., O'Hare, G. M. P., and Collier, R. Reliable multihop intra-cluster communication for wireless sensor networks. In *2014 International Conference on Computing, Networking and Communications (ICNC)*, pages 858–863, 2014. doi:10.1109/ICCNC.2014.6785450.

[175] Xu, L., Collier, R., and O'Hare, G. M. P. A survey of clustering techniques in wsns and consideration of the challenges of applying such to 5g iot scenarios. *IEEE Internet of Things Journal*, 4(5):1229–1249, 2017. doi:10.1109/JIOT.2017.2726014.

[176] Yang, T.-H., Sun, Y. S., and Lai, F. A Scalable Healthcare Information System Based on a Service-oriented Architecture. *Journal of Medical Systems*, 35(3):391–407, Jun 2011. ISSN 1573-689X. doi:10.1007/s10916-009-9375-5.

[177] Yang Zhang, Liangzhong Yin, Jing Zhao, and Guohong Cao. Balancing the Trade-Offs between Query Delay and Data Availability in MANETs. *IEEE Transactions on Parallel and Distributed Systems*, 23(4):643–650, apr 2012. ISSN 1045-9219. doi:10.1109/TPDS.2011.222.

[178] Zhang, Y., Ray, S., Cao, G., Porta, T. L., and Basu, P. Data replication in mobile tactical networks. In *2011 - MILCOM 2011 Military Communications Conference*, pages 797–803, Baltimore, MD, nov 2011. IEEE. ISBN 978-1-4673-0081-0. doi:10.1109/MILCOM.2011.6127774.

Appendix A

Simulation Result Data

Table A.1 *Service Availability (%) provided by the different election algoritms evaluated under Manhattan Grid mobility model.*

	Service Availability (%) - Manhattan Grid					
Nodes	Bully	Kordafshari	No Adaptive	Consensus	Vasudevan	Voting
4	99.65213753	99.65676006	94.75450404	99.44895305	99.73836604	99.00407454
5	99.60214675	99.51946884	86.66188395	99.01283545	99.65832394	98.56814677
6	99.49991525	99.36806203	78.54540518	98.84112148	99.45072151	97.95594173
7	99.33709802	99.14474043	71.05993974	98.09917162	99.20026825	97.61198593
8	99.21727499	99.05685552	64.76969407	97.76750652	98.97250412	97.24647141
9	99.05138865	98.86023921	60.34934938	97.24987393	98.61588503	96.81585844
10	98.93246388	98.63324098	55.90482946	96.59757772	98.20982068	96.55990355
11	98.71994906	98.47081862	51.60159657	95.85820154	97.75107238	96.13147071
12	98.50339138	98.19291263	48.10439408	94.90538248	97.19039014	95.81075588
13	98.29225606	97.90785165	45.60627226	94.22422036	96.54790334	95.4705995
14	98.02203365	97.7849438	43.82438657	93.36129212	95.84870155	95.16206476
15	97.82362763	97.51722743	41.69223933	92.45213795	95.24076151	94.71459697
16	97.61639705	97.28877863	40.13350667	91.58185252	94.44047049	94.38703425
17	97.3740077	97.08964431	38.96589806	91.01205304	93.45071172	94.02704171
18	97.06821475	96.85469129	38.1781266	90.23007153	92.55745561	93.68756253
19	96.76205141	96.6584913	37.31587735	89.32628073	91.58650869	93.49998223
20	96.50252558	96.4906255	36.86376665	88.26880763	90.44869837	93.12159554

Table A.2 *Service Availability (%) provided by the different election algoritms evaluated under Random Walk mobility model.*

	Service Availability (%) - Random Walk					
Nodes	Bully	Kordafshari	No Adaptive	Consensus	Vasudevan	Voting
4	99.55921321	99.39779356	90.98992854	99.07861426	99.65806306	98.14583406
5	99.34875669	99.19835115	82.28395507	98.46342772	99.40369724	97.74834842
6	99.17437927	98.93165618	73.71376892	97.74313085	98.98800488	96.94305481
7	99.00506066	98.56263178	66.29885416	96.84112217	98.46884091	96.51968752
8	98.72531759	98.29538504	60.25550193	95.98275621	97.9654705	96.37078031
9	98.40737434	98.01994779	56.17304739	94.69697122	97.30184089	95.70002714
10	98.15983605	97.68946314	53.09431376	93.63791669	96.48934628	95.39484285
11	97.89010264	97.32885606	49.51572068	93.0716211	95.61975098	94.80224345
12	97.52910643	97.13467516	47.87295849	91.3316869	94.78293493	94.42447741
13	97.26252704	96.81168981	45.76114289	90.40191872	93.56021799	94.05389573
14	96.99198812	96.708356	44.48904329	89.15191964	92.49116382	93.70963317
15	96.76072296	96.35411422	43.80683374	88.36387398	91.57242418	93.35471059
16	96.41118985	96.16513844	42.67066614	86.7522682	90.08842975	92.91769691
17	96.15336837	95.91694885	42.41295835	86.03751461	88.97839445	92.69555365
18	95.79623421	95.78673242	41.42136056	85.50146177	87.44258365	92.3385549
19	95.54147421	95.80141218	40.86763913	84.16871188	86.24459248	92.23869424
20	95.19314684	95.6878467	41.14387852	83.04247493	84.82433697	92.08773482

Table A.3 *Service Availability (%) provided by the different election algoritms evaluated under RPGM mobility model.*

	Service Availability (%) - RPGM					
Nodes	Bully	Kordafshari	No Adaptive	Consensus	Vasudevan	Voting
4	99.61073894	99.56982093	70.23662344	96.31570721	99.76399158	99.25959107
5	99.43684688	99.07828552	59.30329701	93.91785499	99.40968226	98.2078339
6	99.40544892	99.31425107	49.36963004	94.09562697	98.99656326	98.77017773
7	99.24955641	98.94886708	44.79322611	93.1349446	98.67666749	97.96803527
8	99.13692031	98.92490089	42.5217614	92.48252105	98.12085897	97.93138655
9	99.09935312	98.53807161	38.41411575	92.90206775	97.67154649	97.76869045
10	98.97664252	98.47110551	37.18016133	91.85365614	97.10070577	97.54491088
11	98.79502728	98.37703945	36.00381072	91.43926876	96.20644217	96.97975308
12	98.81800838	98.49983595	34.07594869	91.68704864	95.63949204	97.64543669
13	98.65637362	98.40323793	32.90218992	90.84650461	94.46874694	97.25650788
14	98.45851331	98.30300272	32.85410389	90.69945109	93.31704305	97.14855399
15	98.54872405	98.20954299	31.83666164	91.20152713	92.71795178	97.21008219
16	98.31441358	98.28522407	31.37131645	90.26210447	91.00663922	97.01045028
17	98.02342301	98.06485156	31.41274763	88.88582042	89.14172197	96.51280189
18	98.19718785	98.32257076	31.11460286	89.62241617	88.15072937	96.74818713
19	97.8054458	98.1228391	31.03945317	88.71250542	85.37938314	96.56397074
20	97.40125044	97.98585491	31.1663234	88.01221594	82.55875567	96.48466498

Table A.4 *Redundant Server Elections (%) caused by the different election algoritms evaluated under Manhattan Grid mobility model.*

	Redundant Server Elections (%) - Manhattan Grid				
Nodes	Bully	Kordafshari	Consensus	Vasudevan	Voting
4	1.387406617	1.489361702	5.895078421	0	0.053191489
5	1.52650569	2.363080345	5.936712405	0	0.137400385
6	2.215845688	2.200605964	5.851839844	0.031660598	0.0630219
7	2.912621359	3.376541919	5.294800127	0.124468416	0.103423312
8	3.665700052	3.945709412	4.74266906	0.132460078	0.139041347
9	3.877426849	4.299055092	4.272028517	0.142200831	0.152788388
10	4.547997074	4.65126267	3.778207608	0.110394022	0.109252878
11	5.442269196	5.664974619	3.450834879	0.096172979	0.151785125
12	6.221243133	6.120459104	3.396899438	0.118598383	0.15079707
13	7.09657943	6.934568616	2.922415743	0.105234875	0.138488686
14	7.652160108	8.088922471	2.663922602	0.114287891	0.192318678
15	8.578209572	8.644021461	2.471061489	0.12835302	0.194631417
16	9.467365932	9.309477639	2.326182847	0.143342459	0.160748834
17	10.22206604	10.25425794	2.279936055	0.129286539	0.174244419
18	10.8928373	10.59445862	2.179990107	0.12196234	0.198727686
19	11.30138151	11.35528628	2.078240891	0.11812737	0.179775748
20	11.66366897	11.6909853	1.963706751	0.136991551	0.179956502

Table A.5 *Redundant Server Elections (%) caused by the different election algoritms evaluated under Random Walk mobility model.*

	Redundant Server Elections (%) - Random Walk				
Nodes	Bully	Kordafshari	Consensus	Vasudevan	Voting
4	1.889866406	2.221496243	7.302631579	0.097911227	0.064412238
5	2.332214765	3.52901823	5.984359062	0.0331785	0.066028392
6	3.689399329	4.142072054	5.42359058	0.139972006	0.089223753
7	4.796880078	5.569686298	4.49293138	0.184701611	0.088083554
8	6.019766397	6.587136929	4.088506339	0.106000553	0.155699043
9	7.277381605	7.052612416	3.237384506	0.10585371	0.147296065
10	8.556899877	8.835588356	2.960878131	0.139774125	0.190273658
11	9.386849825	9.812592628	3.032457851	0.1411529	0.201599203
12	11.03434178	11.19444274	2.575323368	0.140660526	0.192648532
13	11.66403043	11.73289694	2.547356847	0.119278366	0.209709379
14	13.04179254	13.15047967	2.288533555	0.122511033	0.252893398
15	13.85433594	13.924672	2.331956089	0.12454134	0.192758182
16	14.74402918	14.49323529	2.141435487	0.137486312	0.238920813
17	15.16731577	14.96250586	2.014534817	0.16034146	0.24757276
18	15.7042811	15.68392547	2.006874613	0.161353258	0.240361401
19	16.13278146	16.16961617	1.919733411	0.158972791	0.286506182
20	16.05814625	16.44607491	1.839055604	0.177162657	0.275495843

Table A.6 *Redundant Server Elections (%) caused by the different election algoritms evaluated under RPGM mobility model.*

	Redundant Server Elections (%) - RPGM				
Nodes	Bully	Kordafshari	Consensus	Vasudevan	Voting
4	2.553191489	2.932193036	4.428485937	0.606060606	0.241254524
5	5.178907721	5.261509552	5.220883534	0.339820822	0.24607813
6	5.930044012	7.534406345	4.174228675	0.294384058	0.226705962
7	7.392514692	8.048289738	4.807546022	0.28592927	0.345086272
8	8.934782609	10.15531661	4.300847458	0.217910138	0.304781923
9	11.0479798	11.28580464	3.558594092	0.260869565	0.36761488
10	12.59145449	13.42920354	3.396624473	0.208333333	0.307606264
11	14.23574054	13.65821808	3.243395166	0.15459364	0.302351624
12	14.38222321	14.39159975	2.872228089	0.1506337	0.228504623
13	15.39275578	15.94806705	2.579946777	0.144806705	0.310335522
14	16.36790082	16.76953437	2.350410637	0.178395202	0.29650437
15	16.61240613	16.86393126	1.975609756	0.07390408	0.304741782
16	17.54458731	17.67211605	1.917044266	0.147132587	0.329489292
17	18.37302107	18.27495593	1.968480345	0.12982981	0.334801231
18	17.34642582	18.17743158	1.94861611	0.148737164	0.361934477
19	17.63929232	18.7511428	1.824019859	0.162436056	0.395224597
20	17.41733327	18.73859426	1.680794381	0.16033316	0.365124485

Table A.7 *MTBF (seconds) of the different election algoritms evaluated under Manhattan Grid mobility model.*

	MTBF (seconds) - Manhattan Grid				
Nodes	Bully	Kordafshari	Consensus	Vasudevan	Voting
4	-	-	3906	-	-
5	-	-	2099.625	-	-
6	3517.125	3623.75	1116	-	-
7	1544.1	1127.464286	1009.225	-	-
8	851.2222222	826.0227273	865.2954545	-	-
9	693.4305556	680.6590909	743.3846154	-	-
10	482.640625	461.1785714	661.9166667	-	-
11	341.4861111	295.1959459	548.234375	-	-
12	252.2102273	259.3826531	475.0961538	-	-
13	193.5076923	196.5	488.725	-	-
14	153.4583333	142.5609756	443.5892857	-	6233.5
15	123.1186869	125.6173469	435.0769231	-	3694.833333
16	101.4944444	105.8253968	427.733871	5194	5512
17	83.7533557	84.08396947	391.2564103	4907.25	3703
18	74.09493671	75.58083832	379.8947368	6055.75	2750.083333
19	66.79868421	63.57981221	353.98125	8114	2981.5625
20	60.46052632	59.28196347	350.7714286	3470.4375	3001.8125

Table A.8 *MTBF (seconds) of the different election algoritms evaluated under Random Walk mobility model.*

	MTBF (seconds) - Random Walk				
Nodes	Bully	Kordafshari	Consensus	Vasudevan	Voting
4	-	-	1982.583333	-	-
5	2104.8125	1653.458333	1361.25	-	-
6	1062.6875	1153.4375	1055.21875	-	-
7	601.375	577.8375	777.625	-	-
8	415.375	389.44	664.8947368	-	-
9	288.8636364	286.7	668.90625	-	-
10	191.119403	191.4375	570.8026316	-	-
11	152.0958904	142.1084337	489.25	-	7734.75
12	115.3581081	113.0328947	489.4021739	-	8718.25
13	97.0984252	96.55381944	481.2421875	-	3821
14	79.57432432	79.76174497	461.0403226	-	3123.333333
15	68.7019774	69.41820988	413.0540541	-	3604.416667
16	59.73690476	60.69955157	449.8870968	6547	2653.75
17	56.27589641	57.1468254	441.1838235	4122.5	2543.2
18	52.11209964	52.90520833	435.3014706	4022.875	2245.6
19	49.39074074	50.61389961	447.9791667	4187.375	2205
20	48.57264151	48.95567376	473.4513889	3061.5625	2233.25

Table A.9 *MTBF (seconds) of the different election algoritms evaluated under RPGM mobility model.*

	MTBF (seconds) - RPGM				
Nodes	Bully	Kordafshari	Consensus	Vasudevan	Voting
4	-	-	-	-	-
5	1818.25	2179.5	2058.666667	-	-
6	1818.416667	1450	2863.5	-	-
7	998.1944444	996.3125	1504.9375	-	-
8	670.65625	608.1111111	1220.431818	-	-
9	444.8863636	488.1	1161.84375	-	-
10	372.4469697	344.7878788	1025.4	-	-
11	270.5343137	302.125	1062.666667	-	-
12	254.0243902	247.1890244	1046.75	-	-
13	199.4221311	191.4035088	1091.909091	-	-
14	178.5101351	162.1566265	1102.583333	-	-
15	168.4240506	178.4756098	1111.222222	-	-
16	145.3422619	142.2984694	1313.95	-	-
17	129.0679825	122.9740566	1033.977273	-	6737.5
18	144.1852941	141.3297872	1075.525	-	4944.875
19	124.6659664	112.75	1030.769231	-	4481.875
20	104.6916058	103.4810606	1035.633333	-	4455.75

Table A.10 *Number of Coordination Messages used by the different election algoritms evaluated under Manhattan Grid mobility model.*

	Coordination Messages - Manhattan Grid				
Nodes	Bully	Kordafshari	Consensus	Vasudevan	Voting
4	150	184	206	291	173
5	335	415	549	687	381
6	659	829	1463	1339	735
7	1127	1464	2699	2353	1261
8	1822	2249	4207	3945	2031
9	2818	3552	5954	5955	3173
10	4237	5247	8480	9938	4713
11	6037	7108	13102	15082	6536
12	8269	9650	17155	21434	9353
13	11449	13025	23689	28863	12659
14	15678	16955	29369	41042	16837
15	21950	22935	38392	55756	21788
16	29599	29613	51235	68911	29560
17	41347	38530	62520	88942	39635
18	56629	47426	78135	113297	49439
19	74789	58179	98605	147707	60641
20	94950	69738	117128	174688	74326

Table A.11 *Number of Coordination Messages used by the different election algoritms evaluated under Random Walk mobility model.*

	Coordination Messages - Random Walk				
Nodes	Bully	Kordafshari	Consensus	Vasudevan	Voting
4	326	382	450	599	353
5	703	882	1035	1385	769
6	1429	1717	3012	2920	1531
7	2542	3045	5310	5465	2821
8	4385	5143	9381	9662	4586
9	6944	7634	14519	16722	6845
10	10329	11282	20034	25844	10435
11	14779	16060	25510	38462	15229
12	21994	21808	38008	51191	20769
13	29052	29009	50220	73771	28239
14	40808	36659	61270	96415	36460
15	53982	46338	77256	124124	46368
16	74634	58119	101648	164854	59225
17	102217	70620	121138	206313	73071
18	136969	85308	145575	266263	90497
19	178324	100092	172310	316279	105190
20	238315	116733	209016	377784	123574

Table A.12 *Number of Coordination Messages used by the different election algoritms evaluated under RPGM mobility model.*

	Coordination Messages - RPGM				
Nodes	Bully	Kordafshari	Consensus	Vasudevan	Voting
4	210	287	1747	465	246
5	518	657	6295	1473	605
6	918	1131	8500	3278	995
7	1727	2053	12899	10408	1901
8	3230	3225	17406	22083	3019
9	4594	4899	20099	32402	4487
10	7303	6978	23793	51610	6551
11	10671	9730	31178	78243	9458
12	15092	12151	33578	111402	11528
13	20062	16336	46031	148193	15664
14	27476	21164	53196	204523	20213
15	35818	22531	58140	267613	22440
16	57332	28793	71696	385545	29103
17	76667	37468	89488	479338	36758
18	90684	39079	92339	614101	40018
19	122928	51441	105488	840905	48862
20	192576	62561	122626	1099211	59750

Table A.13 *TFND (minutes) under the different election algoritms evaluated under Manhattan Grid mobility model.*

	TFND (minutes) - Manhattan Grid				
Nodes	Bully	Kordafshari	Consensus	Vasudevan	Voting
4	200.76	202.3613333	195.89325	197.0294167	201.2541667
5	193.2170833	194.1498333	190.45575	189.5835	193.9890833
6	190.5573333	191.85675	186.764	186.7729167	191.1590833
7	188.4825	189.0048333	184.312	183.3799167	187.8044167
8	186.323	187.9039167	183.3638333	183.8683333	189.11125
9	185.7624167	185.6970833	180.8230833	183.2615	186.2676667
10	183.7005	183.9770833	182.8701667	182.1730833	186.0325833
11	185.1125	186.18125	184.2685833	181.73125	188.111
12	186.64425	187.283	186.81075	184.3584167	189.1216667
13	187.08725	187.9841667	187.8998333	185.32675	189.662
14	187.6083333	188.7588333	188.2781667	186.7559167	189.8633333
15	189.0581667	189.7520833	190.2924167	187.107	190.1520833
16	189.7889167	190.4488333	191.3929167	188.2375833	192.19225
17	191.9019167	192.5314167	192.7416667	189.48475	193.3498333
18	191.55025	192.2998333	192.60425	188.0478333	192.83475
19	192.1198333	193.22725	192.7279167	189.193	193.65675
20	192.7644167	193.4905833	193.5634167	188.8573333	193.4064167

Table A.14 *THND (minutes) under the different election algoritms evaluated under Manhattan Grid mobility model.*

	THNA (minutes) - Manhattan Grid				
Nodes	Bully	Kordafshari	Consensus	Vasudevan	Voting
4	215.1805	216.7224167	212.33375	214.5995833	217.5933333
5	205.0831667	205.7315833	201.9333333	203.2070833	205.5435
6	206.8711667	208.3839167	205.0749167	207.0876667	210.1754167
7	206.2999167	207.2778333	204.5158333	206.2835833	210.7658333
8	207.4285833	208.4145833	207.46875	207.4495	212.93925
9	203.9566667	204.60025	204.6273333	204.0889167	210.0175
10	204.7800833	205.8879167	207.6535	206.6865833	212.51375
11	205.00525	205.9865	209.0151667	209.7256667	214.1619167
12	208.2043333	209.3654167	213.945	214.272	219.6459167
13	208.2154167	209.4733333	216.5825833	216.0774167	221.3425833
14	211.2431667	211.2730833	222.1893333	221.865	226.4553333
15	212.9233333	213.4505	227.2111667	225.9411667	230.9889167
16	215.626	216.43975	234.0065	232.1558333	236.7055
17	218.245	218.3935	238.849	237.4568333	241.4713333
18	220.072	220.3125833	243.7428333	242.77525	246.2558333
19	221.48525	220.1583333	246.643	246.14	249.2940833
20	223.7321667	221.7351667	253.4238333	252.33475	254.8660833

Table A.15 *TLND (minutes) under the different election algoritms evaluated under Manhattan Grid mobility model.*

	TLND (minutes) - Manhattan Grid				
Nodes	Bully	Kordafshari	Consensus	Vasudevan	Voting
4	262.7306667	263.8375	263.45625	268.1075833	266.4278333
5	249.9193333	251.6878333	249.3398333	251.8080833	248.81275
6	247.3845833	247.0151667	245.2655	242.5049167	251.5758333
7	241.2320833	242.4736667	242.6315	248.133	250.1429167
8	237.9696667	239.6165833	240.3225833	244.9568333	245.5625
9	233.19	234.0214167	238.4135	240.2993333	244.8429167
10	229.5540833	230.9988333	233.92375	236.9165833	239.8981667
11	229.16225	229.50975	237.02925	238.4346667	240.8268333
12	228.5614167	230.00275	237.9089167	238.9763333	242.3141667
13	226.03225	228.2971667	238.6830833	239.51675	244.687
14	227.6945	228.8109167	241.6163333	242.3948333	246.47175
15	229.8316667	230.7783333	245.8485	248.8686667	250.05625
16	231.4495	232.9204167	251.59925	253.2855	254.6753333
17	234.8139167	233.8770833	257.16975	258.4546667	260.9539167
18	234.1669167	236.31275	260.717	262.941	264.309
19	235.0148333	236.1171667	265.95675	267.89475	268.0521667
20	237.27975	238.7174167	271.1639167	272.9501667	273.399

Table A.16 *TFND (minutes) under the different election algoritms evaluated under Random Walk mobility model.*

	TFND (minutes) - Random Walk				
Nodes	Bully	Kordafshari	Consensus	Vasudevan	Voting
4	201.8935	202.7266667	198.1645	199.1919167	203.7913333
5	196.196	196.8953333	193.2048333	194.0045833	197.5291667
6	192.6526667	192.9858333	191.7560833	191.9460833	196.4605833
7	194.2065	194.285	193.14475	193.11625	197.4514167
8	194.26725	195.37675	194.5898333	193.8415	198.6321667
9	194.4601667	196.05675	196.7465	194.271	198.7539167
10	193.8669167	194.9703333	196.8511667	193.36075	197.83125
11	195.1416667	195.9795833	197.3480833	193.539	198.84825
12	196.2806667	197.0725	199.0665	195.37725	199.7531667
13	195.02925	196.1231667	197.537	193.50825	197.2974167
14	195.3195	195.177	196.7695	193.27025	197.0648333
15	195.7113333	196.70325	197.3096667	193.2286667	197.28425
16	196.3868333	197.105	196.9154167	193.53925	197.1680833
17	196.6675	196.6361667	196.9085	193.4845	197.4634167
18	195.3380833	195.9483333	195.6896667	192.2335833	196.1086667
19	195.349	195.4419167	195.5163333	192.5134167	195.8575
20	195.4388333	195.2586667	195.0415833	192.217	195.6859167

Table A.17 *THND (minutes) under the different election algoritms evaluated under Random Walk mobility model.*

	THNA (minutes) - Random Walk				
Nodes	Bully	Kordafshari	Consensus	Vasudevan	Voting
4	211.5348333	211.7688333	208.3289167	209.7653333	215.10975
5	204.36025	205.1900833	202.5795	203.3509167	207.9044167
6	205.1405833	206.22875	204.9840833	205.6190833	211.48075
7	206.0979167	206.3653333	208.7156667	208.7848333	214.1161667
8	208.5995833	209.4663333	212.77325	212.8208333	217.3334167
9	208.4996667	209.8108333	216.04525	214.5038333	219.6715
10	210.7861667	211.0659167	222.3665833	221.2285833	226.0445
11	213.1883333	213.2259167	225.8666667	224.9976667	229.4703333
12	216.8819167	217.6041667	235.4764167	233.9789167	238.1574167
13	218.88325	218.9123333	240.76275	239.0485	243.2424167
14	221.0424167	219.84625	248.9014167	246.8738333	250.383
15	222.7719167	221.9024167	254.6821667	252.2893333	255.6721667
16	224.7790833	224.0723333	262.98575	261.6028333	264.6415
17	227.4625	225.195	268.0393333	266.1474167	268.0383333
18	228.7020833	226.06175	274.5394167	272.7291667	274.1661667
19	230.4206667	226.5635833	277.8864167	277.105	278.0260833
20	233.85125	227.3101667	283.4975833	283.0906667	284.7745

Table A.18 *TLND (minutes) under the different election algoritms evaluated under Random Walk mobility model.*

	TLND (minutes) - Random Walk				
Nodes	Bully	Kordafshari	Consensus	Vasudevan	Voting
4	228.7595833	230.5470833	227.0539167	233.1580833	233.3755833
5	225.2951667	225.2015	225.6646667	227.9850833	231.49075
6	222.6560833	224.0280833	223.4035833	226.6045833	230.9376667
7	220.31125	220.6389167	223.7524167	226.29375	230.2911667
8	221.4453333	222.16875	228.1529167	229.93775	232.4458333
9	222.0131667	223.4075833	231.1420833	232.1809167	236.5973333
10	223.776	224.2820833	236.5775	236.1518333	239.76
11	226.2204167	225.8448333	241.6958333	243.5820833	245.3238333
12	229.0515833	229.5498333	250.3675833	251.1451667	253.1709167
13	230.5809167	231.1294167	255.346	257.5514167	258.2775
14	231.7780833	232.40025	262.3016667	262.6331667	264.4899167
15	235.3564167	234.7411667	268.7874167	269.89075	270.5395833
16	235.3999167	236.6188333	276.9761667	277.636	277.08725
17	238.7518333	238.06425	283.9778333	283.7350833	285.5545833
18	240.3616667	238.5149167	289.27	289.8836667	290.9798333
19	241.11275	234.406	295.8946667	294.7510833	296.9609167
20	245.0820833	239.6525	300.88625	299.92725	303.24525

Table A.19 *TFND (minutes) under the different election algoritms evaluated under RPGM mobility model.*

	TFND (minutes) - RPGM				
Nodes	Bully	Kordafshari	Consensus	Vasudevan	Voting
4	158.9196667	159.8389167	160.7681667	154.6795	158.024
5	175.0833333	175.72525	175.3723333	172.5065833	177.5841667
6	178.5310833	178.4216667	180.57725	176.3401667	181.5649167
7	188.3188333	189.3008333	190.7716667	184.8956667	191.4230833
8	191.8070833	193.13975	192.5426667	188.9054167	195.1005833
9	186.5799167	187.0834167	188.07675	182.441	189.3346667
10	191.2974167	191.3834167	193.523	187.7578333	194.4675833
11	193.04625	194.6395833	195.1004167	189.577	195.53725
12	197.3285	197.80025	198.71375	193.0099167	199.3345
13	196.1856667	197.3039167	197.5961667	191.9126667	197.8431667
14	196.4633333	196.992	197.3501667	190.907	197.2545833
15	195.32825	195.9326667	196.5600833	191.7490833	196.4243333
16	196.22275	197.3295833	197.2014167	191.37875	196.7443333
17	197.02325	197.5234167	197.6506667	191.1996667	197.4305833
18	196.0215	196.4999167	196.1063333	192.0925833	195.8825
19	196.3758333	196.1935833	196.3890833	193.43075	195.8515
20	196.3220833	195.5763333	195.5409167	193.0063333	195.0475833

Table A.20 *THND (minutes) under the different election algoritms evaluated under RPGM mobility model.*

	THNA (minutes) - RPGM				
Nodes	Bully	Kordafshari	Consensus	Vasudevan	Voting
4	181.01	181.6578333	183.5369167	182.2925833	181.7369167
5	188.36825	188.9341667	192.9151667	191.6494167	191.4263333
6	200.1856667	198.1464167	205.4590833	203.0485	203.9768333
7	206.77475	206.836	211.9315833	213.1585	212.8704167
8	212.5720833	213.2654167	221.5549167	221.6675833	221.8993333
9	212.8421667	213.6655833	222.163	222.4785	223.0373333
10	217.543	216.8423333	230.7794167	230.3050833	231.90875
11	219.3430833	220.42475	235.73125	234.6878333	235.6721667
12	225.6433333	224.8676667	242.7875	241.7118333	242.9894167
13	224.90225	223.5580833	246.0469167	244.6555833	246.5759167
14	228.028	227.7339167	252.1060833	250.6828333	253.74125
15	230.80125	230.0445833	254.8305833	254.2628333	258.301
16	234.3753333	232.10225	262.6251667	261.4213333	266.1153333
17	235.1175833	233.1738333	266.4736667	265.4185833	270.4855833
18	240.3415833	236.7798333	272.8619167	271.3826667	279.8698333
19	242.1610833	236.65825	276.0199167	274.2265	283.5605833
20	242.8950833	236.7895	280.5253333	277.9296667	291.9001667

Table A.21 *TLND (minutes) under the different election algoritms evaluated under RPGM mobility model.*

	TLND (minutes) - RPGM				
Nodes	Bully	Kordafshari	Consensus	Vasudevan	Voting
4	217.3985	219.26375	219.9325833	220.3223333	224.54275
5	221.1586667	222.0615	224.799	227.9044167	231.3400833
6	227.8123333	226.8664167	236.78025	235.95475	239.1425
7	227.6996667	228.10125	237.8681667	237.0311667	240.9768333
8	229.79925	229.67875	241.3446667	241.3064167	243.577
9	233.8051667	234.4163333	250.97775	249.6633333	252.66425
10	234.69475	229.1286667	251.3580833	252.01375	255.2046667
11	235.5958333	237.5231667	259.5136667	258.343	262.3011667
12	241.0865	241.6891667	265.7040833	263.0799167	268.0973333
13	238.35525	240.277	268.9115	265.90075	272.00725
14	239.6455	240.6465	275.57025	269.9900833	275.4621667
15	246.4205	246.9804167	285.73625	279.9863333	285.0711667
16	248.434	245.1340833	287.1625833	282.98575	290.58775
17	248.8080833	242.1036667	293.7808333	288.9179167	296.5381667
18	252.39375	249.85025	296.57475	292.1444167	302.3785833
19	255.28875	248.3811667	300.9949167	295.4241667	309.0971667
20	254.5863333	248.18525	305.2543333	299.78225	308.6295