Blockchain Application for Mesh and Mobile Ad Hoc Networks

David Alexis Córdova Morales

Sorbonne Université

École Doctorale Informatique, Télécommunications et Électronique de Paris

Laboratoire d'Informatique de Sorbonne Université (LIP6)

Blockchain Application for Mesh and Mobile Ad Hoc Networks

Présenté par

David Alexis CÓRDOVA MORALES

Thèse de doctorat en Informatique

Présentée et soutenue publiquement le 30/11/2022

Devant un jury composé de:

Stefano SECCI Professeur, Conservatoire National des Arts et Métiers	Rapporteur
Diogo MENEZES FERRAZANI MATTOS Professeur, Universidade Federal Fluminense (Brésil)	Rapporteur
Anne FLADENMULLER Professeur, Sorbonne Université	Examinatrice
Nadjib ACHIR Maître de conférences HDR, Université Sorbonne Paris Nord	Examinateur
Pedro BRACONNOT VELLOSO Maître de conférences, Conservatoire National des Arts et Métiers	Invité (Co-Encadrant)
Guy PUJOLLE Professeur émérite, Sorbonne Université	Co-Directeur
Thi-Mai-Trang NGUYEN Maître de conférences HDR, Sorbonne Université	Directrice de thèse

Sorbonne Université

École Doctorale Informatique, Télécommunications et Électronique de Paris

Laboratoire d'Informatique de Sorbonne Université (LIP6)

Blockchain Application for Mesh and Mobile Ad Hoc Networks

Présenté par

David Alexis CÓRDOVA MORALES

Thèse de doctorat en Informatique

Présentée et soutenue publiquement le 30/11/2022

Devant un jury composé de:

Stefano SECCI Professeur, Conservatoire National des Arts et Métiers	Rapporteur
Diogo MENEZES FERRAZANI MATTOS Professeur, Universidade Federal Fluminense (Brésil)	Rapporteur
Anne FLADENMULLER Professeur, Sorbonne Université	Examinatrice
Nadjib ACHIR Maître de conférences HDR, Université Sorbonne Paris Nord	Examinateur
Pedro BRACONNOT VELLOSO Maître de conférences, Conservatoire National des Arts et Métiers	Invité (Co-Encadrant)
Guy PUJOLLE Professeur émérite, Sorbonne Université	Co-Directeur
Thi-Mai-Trang NGUYEN Maître de conférences HDR, Sorbonne Université	Directrice de thèse

Contents

List of Figures

List of Tables

Remerciements

Les travaux réalisés durant ces trois dernières années furent riches en apprentissage, non seulement d'un point de vue de la recherche scientifique ou technique, mais aussi d'un point de vue humain. En effet, le développent des nouvelles compétences personnelles que j'ai pu apprendre durant mon séjour au sein de l'équipe Phare du laboratoire d'informatique de Paris 6 m'ont permis de découvrir et de développer une nouvelle facette de moi-même capable d'achever des grands objectifs. Cela, n'aurait pas eu été possible sans l'aide des toutes les personnes qui m'ont soutenue tout au long de ma thèse.

Je voudrais tout d'abord commencer par adresser mes sincères remerciements à ma directrice de thèse Thi-Mai-Trang Nguyen pour avoir accepté de diriger ma thèse. Son constant soutien ainsi que son sens de la responsabilité et ses conseils m'ont énormément aidé à surmonter tous les défis qui se sont présentés tout au long de ma thèse. Ces nombreux enseignements, tant dans le plan technique et méthodologique que pour l'exercice de synthèse et de rédaction furent précieux. D'autant même, je voudrais adresser mes remerciements à mon codirecteur de thèse Guy Pujolle, qui m'a beaucoup aidé a garder une vision d'ensemble des enjeux de ma thèse ainsi que de me rappeler constamment les objectifs à atteindre. Son soutien ainsi que son expertise furent déterminants pour mener à bien mon entreprise. Finalement, je voudrais exprimer toute ma gratitude à Pedro Braconnot Velloso, qui par le hasard des faits est devenue mon encadrant de thèse. Sa constante disposition au dialogue et à l'échange des idées fut une constante source d'inspiration grâce à son très solide savoir technique de la matière. Je me considère chanceux à avoir eu l'opportunité d'apprendre de vous trois de si près depuis la position privilégiée du doctorant. Je garderais en mémoire des très bons souvenirs des nombreuses réunions de travail riche en échanges, débat, et rires. C'est grâce à votre expertise, vos conseils et votre soutien que cette thèse arrive à échéance.

Je tiens à remercier mes rapporteurs de thèse, Steffano Secci et Diogo Menezes Ferrazani Mattos pour leur temps consacré à la lecture et au rapport de cette thèse. Leurs critiques constructives ont permis d'améliorer la qualité de cette thèse. De même, je voudrais remercier Anne Fladenmuller et Nadjib Achir d'avoir accepté de former part de mon jury de thèse. Leur expertise et leur œil critique ne font qu'additionner au travail qui est présenté dans cette thèse.

De même, je voudrais remercier toutes les personnes qui direct ou indirectement ont participé à la réalisation de cette thèse. Alexandre Laubé, pour m'avoir donné le coup de

pouce initial dont j'avais besoin pour commencer à mener les expérimentations qui sont décrites dans cette thèse, Alexandre Guerre, pour avoir pris le challenge d'implémenter notre solution pour créer une plateforme fonctionnelle, Khaldoun Al Agha, pour sa précieuse participation à la gestion et facilitation du projet dans lequel cette thèse est contextualisée, ainsi que pour ses travaux initiaux qui furent essentiels au développement des idées originales présenté dans cette thèse, et Brigitte Kervella pour ses corrections et ses conseils.

Je remercie également le Conseil National de Science et Technologie (CONACYT) du gouvernement du Mexique, qui a financé en grande mesure le développement de cette thèse, sans leur généreuse contribution, cette thèse n'aurait pas eu lieu.

Je voudrais aussi dédier quelques lignes à toutes ces personnes qui me sont chères, et qui par le fait de leur existence, m'ont apporté du bonheur, du rire, des aventures, du soutien, de la complicité, de l'amour, et même du désamour et du chagrin. Sans eux, Paris ne serait pas une fête.

Finalement, mes plus profonds remerciements vont à ma famille, à mon père et à ma mère qui m'ont toujours soutenu et qui malgré la distance ont toujours été présents à chaque jalon de mon parcours pour m'offrir leur amour et leur soutien. À mon frère et ma sœur dont notre complicité transcende les frontières, et à Ada, ma nièce chérie qui m'inspire de l'optimisme envers le futur.

Abstract

Blockchain is a technology that maintains a single record of information in a decentralized and distributed manner while ensuring data security. This technology, which is behind the most popular cryptocurrency, Bitcoin, is changing the way we think about information records in distributed systems. Indeed, the cryptographic feature set and the distributed nature of the technology make blockchain one of the most secure tools available today for maintaining a record of information.

The first applications to have adopted this technology are in the field of finance, where it is now possible to carry out transactions directly between users without going through a central authority. However, other fields have also shown interest in this technology, such as medicine, for the secure sharing of medical records; art and music, for the tracking of copyrights and royalties; governance, for secure voting, IoT, etc.

However, to benefit from such technology it is necessary to count with high reliability and connectivity, as provided by the Internet. In mesh and mobile ad hoc networks, it is often necessary to deploy its own infrastructure and services where the infrastructure of operators are not available due to the geography of the site or to an exceptional situation as is the case of natural disasters, war zones or monitoring of protected areas to achieve specific missions. The dynamism of these networks makes it difficult to use a blockchain to maintain a record of information. Indeed, the mobility of nodes can cause partitions in the network that may or may not be desired; nodes can appear and disappear, partitions can split or reunite depending on the mobility of the nodes. This poses a problem for a traditional blockchain, as partitions in the network lead to forks (competing chains) that are often resolved by choosing the longest chain and ignoring other competing chains. For the use cases of mesh and mobile ad hoc networks that we seek to solve, the concurrent chains constructed by the effect of network partitions can be considered as legitimate chains carrying information related to a given network partition. It is therefore important to include these chains in the information register.

In this thesis manuscript, we propose the **Blockgraph**, a blockchain-like technology capable of dealing with network partitions for mobile mesh and ad hoc networks. The Blockgraph, takes the form of a direct acyclic graph based on node mobility and inherits all the security properties of the blockchain. In addition, we propose **C4M**, a RAFT-based consensus algorithm that has been adapted to work with the Blockgraph and is also tolerant to network partitions.

To validate our solutions, we first implemented the Blockgraph and C4M in the discrete event simulator, NS-3. We performed a first performance study of each system,

then we implemented the Blockgraph in real proof-of-concept mesh routers to validate the effectiveness of our solution. Finally we performed a performance study and presented our conclusions.

Résumé

La blockchain est une technologie qui permet de maintenir un unique registre d'information de façon décentralisée et distribuée tout en garantissant la sécurité des données. Cette technologie, qui est à l'origine de la cryptomonnaie la plus populaire, le Bitcoin, est en train de changer la façon dont nous concevons les registres d'informations dans les systèmes distribués. En effet, l'ensemble des fonctions cryptographiques ainsi que la nature distribuée de la technologie font de la blockchain, un des outils les plus sécurisés de nos jours pour maintenir un registre de l'information.

Les premières applications à avoir adopté cette technologie se trouvent dans le domaine des finances, où il est désormais possible de réaliser des transactions directement entre les utilisateurs sans passer par une autorité centrale. Néanmoins, d'autres domaines ont aussi suscité leurs intérêts pour cette technologie, telle que la médecine, pour le partage sécurisé des données médicales ; l'art et la musique, pour le suivi des droits d'auteur et des redevances ; la gouvernance, pour le vote sécurisé, l'IoT, etc.

Or, pour bénéficier d'une telle technologie il est nécessaire de compter avec une haute fiabilité et connectivité, telle que fourni par l'Internet. Dans les réseaux mesh et ad hoc mobile, il est souvent nécessaire de déployer sa propre infrastructure et ses propres services là où l'infrastructure des opérateurs ne sont pas disponibles dus à la géographie du site ou à une situation d'exception comme est le cas de désastre naturels, zone de guerre ou le monitorat des zones protégées pour réaliser des missions déterminées. La dynamiste de ces réseaux rend difficile l'utilisation d'une blockchain pour maintenir un registre d'information. En effet, la mobilité des nœuds peut causer des partitions dans le réseau qui peuvent ou pas être désirées ; des nœuds peuvent apparaître et disparaître, les partitions peuvent se séparer ou se réunir en fonction de la mobilité des nœuds. Cela pose un problème pour une blockchain traditionnelle, car les partitions dans le réseau entraînent des *forks* (des chaînes concurrentes) qui sont souvent résolu en choisissant la chaîne la plus longue et en ignorant les autres chaînes concurrentes. Pour les cas d'utilisation des réseaux mesh et ad hoc mobile que nous cherchons à résoudre, les chaînes concurrentes construites par effet des partitions réseaux peuvent être considérées comme des chaînes légitimes portant des informations relatives à une partition réseau déterminé. Il est donc important d'inclure ces chaînes dans le registre d'information.

Dans ce manuscrit de thèse, nous proposons le **Blockgraph**, une technologie semblable à la blockchain capable de faire face aux partitions réseaux pour les réseaux mesh et ad hoc mobiles. Le Blockgraph, prend la forme d'un graph orienté acyclique en fonction de la mobilité de nœuds et hérite de toutes les propriétés de sécurité de

la blockchain. De plus, nous proposons **C4M**, un algorithme de consensus inspiré en RAFT qui a été adapté au Blockgraph et qui également est tolérant aux partitions du réseau.

Pour valider nos solutions, nous avons d'abord implémenté le Blockgraph et C4M dans le simulateur à événements discrets, NS-3. Nous avons réalisé une première étude des performances de chaque système, puis nous avons implémenté le Blockgraph dans des vrais routeurs mesh à mode de proof-of-concept pour valider l'efficacité de notre solution. Finalement nous avons réalisé une étude de performances et présenté nos conclusions.

Chapter 1

Introduction

Since the surge of Bitcoin's whitepaper in 2008 by the pseudonym Satoshi Nakamoto [1], blockchain technology has gained tremendous notoriety among the scientific community and has inspired the development of new technologies and applications. It first emerged as an alternative to the current economic system where no-trust parties are needed between participants to carry out transactions. This was a revolutionary change of paradigm. Centralized economic institutions are no longer needed to either carry out transactions or verify the solvency of an account. Bitcoin's blockchain was able to remove effectively the *man-in-the-middle* by reaching a consensus on the state of a distributed ledger in a peer-to-peer network. Naturally, the success of Bitcoin has put blockchain technology in the spotlight. Other blockchain projects began to proliferate with the objective to decentralize other services or competing with Bitcoin. A few years after the lunch of the Bitcoin network, Ethereum co-founder Vitalik Buterin introduce the idea of using the key functionality of Bitcoin's blockchain with a Turing-complete language [2], which gave birth to Smart Contracts and decentralized applications (dApps). This blockchain breakthrough gave users the possibility to create and specify all kinds of agreements that are enforced autonomously and automatically by the blockchain system without the need of involving a trusted third party. Today, blockchain technology can be found in almost all areas of the social sphere and it is said to be the cornerstone of the development of what today is known as web 3.0 [3].

One of the main reasons for the success of blockchain technology is its properties that enhance data security and transparency. Today, in a context where cyberattacks are on the rise and where cybersecurity concerns is the 7th risk that worsens the most to companies and industries [4], many are those who have turned towards blockchain as a potential solution to protect data from tampering or as mean to perform traceability [5, 6, 7, 8]. Moreover, as the world increasingly embraces digitalization, the urgency of having a trusted system capable of safeguarding proof-tempered and non-repudiation

information also increases. In this regard, blockchain technology provides a series of guarantees that complies with those expectations.

Today, blockchain solutions are being explored in the health sector [9], supply chain [5], real states [10], energy supply [8], the Internet-of-Things (IoT) [11], citizen and corporate governance [12], financial applications [13], defense [14], and nature monitoring and conservation [15]. However, not all kind of networks can benefit from blockchain technology. Wireless Mesh Networks (WMNs) and Mobile Ad Hoc networks (MANETs) are a kind of network that changes its network connectivity dynamically as nodes in the network move. And since blockchain technology relies on a peer-to-peer fully connected network, the connectivity and reliability requirements are no longer met to use a blockchain in these networks. Indeed, due to topology changes, nodes in the network might split and merge into different independent partitions, disrupting the connectivity of the network, which creates consistency issues in the blockchain distributed ledger by creating different versions of the blockchain. This issue is commonly referred to as the split and merge problem as defined in [16].

1.1 Context and motivation

MANETs are a form of a decentralized network that is composed of wireless nodes that can directly communicate with each other without relying on network infrastructure. They use multi-hop routing to reach other wireless nodes, which are not within their transmission range. WMNs are composed of mobile wireless routers that can directly communicate with each other to form a mobile wireless infrastructure, providing connectivity to clients who are connected to the nearest wireless router. These technologies have the advantage of allowing easy network deployment in areas where network infrastructure, such as access points or base stations are not available or not easy to build. Thus, they are the chosen technologies in case of natural disaster areas, war and conflict zones, wildfire and other environmental monitoring, remote areas, and ephemeral campsites such as refugee camps.

Traditionally, MANETs are associated with very restrained resource devices. However, the significant advances in computing power, storage capacity, and battery autonomy have enhanced the use of these networks by integrating more advanced and sophisticated embedded services and applications [17]. Indeed, some mobile wireless routers are now endowed with important resources that enable cloud computing capabilities at the edge of the network [18]. It is called the Multi-Access Edge Computing (MEC) paradigm, which in our context, allows for storage and process of data close to the end-users, which helps drive significant performance enhancements, including

higher bandwidth, lower latency, and faster response times and decision-making.

It is, therefore, under these new circumstances, where MANETs and WMNs are no longer composed of wireless devices that lack computational and material resources, that we aim to secure legacy and future services and applications by integrating blockchain technology to the benefit of these networks.

However, to integrate blockchain technology with these networks, we first need to deal with the split and merge problem. Indeed, the mobility of the nodes may cause changes in the network topology at the point to split the network into several independent sub-networks or merging several sub-networks into a single network. Thus, the mobility factor poses several challenges that need to be addressed to correctly integrate blockchain technology with MANETs and WMNs. Today, no traditional blockchain is capable of dealing with the split and merge problem since they were conceived to function in a fully connected network. Submitting them to a context of mobility would make their protocols deal with network partitions as if they were undesirable events by triggering security mechanisms that counteract the harmful effects that network partitions entail. In the case of the Bitcoin network, it is the longest chain rule that prevails, in other cases, different mechanisms are implemented to maintain a single chain. These mechanisms are in some cases implemented at the consensus level and are the main reason on way MANETs and WMNs are not able to benefit from blockchain technology.

Thus, it is as important to rethink the way blockchain technology works as the consensus algorithm that it implements. In this regard, we developed a blockchain-like technology that treats network partitions as normal behavior of the network while maintaining the properties and guarantees of blockchain technology. Moreover, we were inspired by a legacy consensus algorithm to develop a new consensus algorithm capable of tolerating network partitions. Thus, we can now provide MANETs and WMNs with a blockchain-like technology capable of securing legacy and future applications in a mobile distributed system.

1.2 Contributions

This dissertation discusses and explains the fundamental basis for a blockchain-like technology to become resilient to intended network partitions in a mobile environment and proposes a new consensus algorithm capable of tolerating frequent network partitions.

The main contribution of our work is the construction of a new type of Distributed Ledger Technology (DLT) with the characteristics of a blockchain, which is capable

of overcoming the challenges that a mobile network entails. To such a solution, we named it **the Blockgraph**. The Blockgraph is a full framework composed of three independent modules that treat three relatively independent sub-problems.

The first module is *the Blockgraph protocol*. This module is in charge of managing the blockchain-like data structure, also referred to as the Blockgraph data structure, which can take the form of a Direct Acyclic Graph (DAG). It administrates the incoming and outcoming blocks of the module to ensure the data consistency across all nodes is maintained. It also executes *the merge synchronization procedure*, which allows for the fusion of different Blockgraph data structures into a single data structure when several network partitions merge into a single network. Finally, it provides the needed communication interfaces to exchange with an external application and the other modules of our framework.

The second module of our framework is *the consensus module*, which could be understood as a sort of proxy between a consensus protocol and the Blockgraph framework. It provides *the Blockgraph protocol* with blocks that have passed through a consensus process and pass to the hosted consensus protocol with updated network topology information. Moreover, we have defined a series of properties and characteristics that a consensus protocol needs to have to be compatible with our Blockgraph solution. To that, we have proposed a new consensus algorithm, **Consensus for Mesh (C4M)**, which can tolerate network partitions by renewing the set of nodes participating in the consensus process as the topology of the network changes with mobility. Our algorithm inspires by the work of Diego Ongaro and John Ousterhout, which introduced the RAFT algorithm [19]. Their work includes a cluster membership change mechanism that updates the set of nodes participating in the replicated state machines consensus. However, this mechanism can only be used in occasional configuration changes, when a node fails, or when changing the degree of the system replication without having to take the entire cluster down. In this regard, we made modifications to introduce our *configuration change procedure* that performs this action in a distributed manner. Additionally, C4M implements a new *operational mode* and a *log realignment process* to make the algorithm more reactive to network partitions.

The third module that integrates our Blockgraph framework is *the group management module*. It provides *the consensus module* and *the Blockgraph protocol* with network topology information, such as the list of nodes in the topology and the nature of a topology change, meaning whether the network has suffered a split or a merge. It rests on a network discovery solution capable of sensing the network topology. In this regard, we implemented our solution based on the proactive nature of the OLSR routing protocol [20] by exploiting the nodes' routing table.

To validate our Blockgraph concept, we first implemented our solution in the discrete-event network simulator, NS-3 [1], and we then evaluated our solution. The results of our evaluation show that our solution was able to replicate 99.72% of the blocks when submitted to a split and a merge in a network composed of 10 nodes. Moreover, the system showed stability during operation in all modules when providing a transaction arrival rate of 5 transactions per second (tps). The concept of our work and the results of the simulations were presented in this conference paper [21].

We then focus our attention on C4M. To validate our algorithm, we implemented C4M in NS-3 [2]. The objective of our simulations was, from one side, to provide a first validation of the efficacity of our algorithm when subjected to intended network partitions and, from the other side, to evaluate the performance of our algorithm and characterize its behavior. Results of our simulations demonstrated the efficiency of our algorithm in changing the set of nodes participating in the consensus process with intended and frequent network partitions. Moreover, we characterized the parameterization of our algorithm based on the number of nodes in the system. The introduction of C4M and the results of our work were presented to the community in a conference paper [22] that we extended into a journal article in process of publication.

As part of a research project, we implemented our Blockgraph solution in a testbed composed of five low-power mesh routers. During the implementation phase, we improved our *merge synchronization procedure*, which increased the efficiency of our system in merging the Blockgraph data structures. Moreover, we successfully tested the modularity of our framework by integrating the mesh routers' consensus algorithm with our *consensus module*. The demonstration of our proof-of-concept consisted in provoking intended network partitions by separating the mesh routers into two clusters. Our Blockgraph solution was able to handle the network partitions by constructing the DAG and replicating it in all five mesh routers. Our SIGCOM'21 Demo paper [23] was awarded the second prize of the ACM Student Research for this work. Further on, we conducted a performance evaluation of our testbed where we measured the transaction throughput of our solution and the resource consumption of our framework in the mesh routers. The results of this evaluation published in [24] demonstrate that the achieved transaction throughput represented at least 85% of the maximum theoretical transaction throughput and that the resource consumption, in terms of Random-Access Memory, was relatively moderated regardless of the transaction arrival rate.

[1] https://gitlab.lip6.fr/cordova/b4mesh
[2] https://gitlab.lip6.fr/cordova/c4m

1.3 Thesis outline

To begin, **Chapter 2** presents the background and related work of this thesis. We first introduce the technical details of blockchain as well as the cryptographic primitives involved. We then cover the related work specific to each of our contributions and we finish this chapter with a summary.

In **Chapter 3** we present the Blockgraph. We begin by introducing the context and boundaries of our solution and its main characteristics. We then present the split and merge problem that our solution solves. Here, we raise several questions about the issues that may result from a peer-to-peer network running a blockchain-like technology and the main consequences on the state of the distributed ledger when facing network dysconnectivity. Furthermore, we explain how this issue (split and merge problem) may affect the different consensus algorithms. We continue by describing the system model and a punctual description of every module forming the Blockgraph framework. We highlight our *merge synchronization procedure*, which allows the fusion of two independent chains into a single one and gives the Blockgraph its graph structure. We then present our two solution implementations that validate our concept with its corresponding performance evaluation, and we close this chapter with a summary.

In **Chapter 4** we delve into the consensus layer and explain the need for a distributed consensus algorithm resilient to constant changes in the network topology in our context. We present our consensus solution, C4M, which is a RAFT-based consensus algorithm with the ability to adapt its set of nodes participating in the consensus process dynamically and to automatically align the indexes used for the consensus process according to the topology changes. To validate our solution, we implemented a prototype version of our C4M algorithm using the discrete-event network simulator, NS-3. Most of our implementation was done and designed to approach as much as possible to a real implementation. Thus, in this chapter, we present an overview of RAFT, a full description of our C4M consensus mechanism, and the NS-3 implementation of our consensus algorithm. Finally, we conclude this chapter by presenting the performance evaluation of our simulations and a summary.

Chapter 5 presents our general conclusions and future perspective of our work. In addition, we provide **Appendix A**, which provides our main algorithms from our contributions.

Chapter 2

Background and State-of-the-Art

2.1 The Bitcoin System

To understand the fundamental basis of blockchain, we first have to understand the motivation behind blockchain technology. Bitcoin's blockchain is the first use case of blockchain and therefore, is it important to understand which problems its solves to correctly implement this technology in other contexts and use cases such as the one we present in this dissertation. Bitcoin's blockchain was conceived to create a peer-to-peer electronic cash system [1] as an alternative to the current economic system. Many works in the field of economics [25, 26, 27, 28], refer to the unsustainability of the current economic system and criticize the current *modus operandi*, which mainly relies on its centralized approach built on baseless trust; where financial institutions unilaterally dictate political-economic decisions that directly impact civil society. In this context, Bitcoin's blockchain allows individuals to transact directly with each other without the need for financial institutions such as banks or other trusted third parties while maintaining pseudo-anonymity, financial control, and small transaction fees.

2.1.1 Basic Functioning

The Bitcoin system works the same way as an accounting ledger book; it keeps a register of all transactions performed in the Bitcoin network. However, unlike a traditional accounting book ledger, bitcoin's transactions are replicated in a distributed network composed of thousands of nodes with no central authority and use a consensus mechanism that allows all nodes to agree on the same state of the distributed ledger and the order in which transactions were executed. Once the nodes have agreed on the same state of the distributed ledger, transactions become immutable and cannot be undone or forged, providing certainty and guarantee on bitcoin's balances.

Transactions are compiled into blocks, which is a data structure where transactions are stored. We can think of blocks, as a page in our accounting ledger book where a set of transactions are listed. Blocks are said to be chained among them by including bits of the previous block in the new block as illustrated in Figure 2.1. Indeed, every block in the blockchain includes the identifier of the previous block in its header, except for the genesis block, which is the first block in the blockchain and has no predecessors; these identifiers are the outcome of a cryptographic function known as the one-way function (a.k.a., hash function) that takes as an input, the header of the block, which also includes transactions' information and return a unique value. They are said to be one-way because the input cannot be deduced from the output of the function and because there is only a unique output for every input. In this regard, tempering with a block will change the outcome of the hash function and will invalidate the block since the block identifier will no longer match with the information provided in the next block.

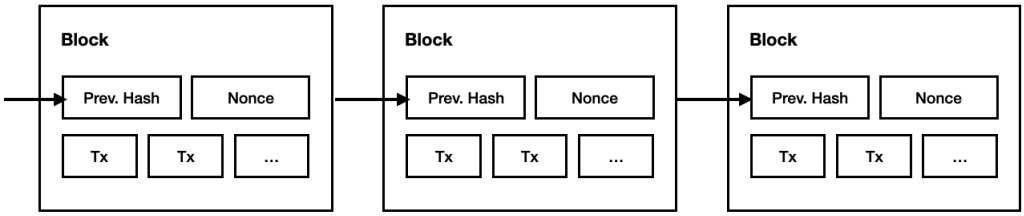


Figure 2.1: Blocks of transactions chained through a hash function.

Blocks are proposed by miners, which are nodes participating in the consensus mechanism; they compete with each other to be the first in proposing the next block to be appended to the blockchain. The first miner node proposing a valid block has the right to propagate the block in the network so other nodes can verify the validity of the block. In Bitcoin, miners nodes have to solve a probabilistic mathematical puzzle that requires high computational resources before every other node solves it. Miners have to find through a hash function the identifier of the block they are proposing, such as the outcome of the hash function begins with a certain number of zeros; to do that, they use a nonce that takes different values until generating the correct hash that meets all the requirements. When the puzzle is solved, the value of the nonce that allows the verification of the puzzle is included in the header of the newly proposed block so other nodes can easily validate the result of the puzzle. This process is known as Proof-of-Work (PoW) since proof of the solution is included in the block, which accounts for the effort spent by a node to solve the puzzle. A block is considered committed after being validated by a majority of nodes. Indeed, every node participating in the

consensus process validates every transaction in the proposed block, ensuring that there are no transactions incoherences in the block and that the nonce included in the header solves the cryptographic puzzle. If a node does not validate a block due to transaction inconsistencies or the nonce does not solve the puzzle, the node will refuse to acknowledge the validity of the block and will ignore the block.

2.1.2 The Double Spending Problem

The Bitcoin system solves a long-time unsolved problem by its predecessors, the double spending problem. The problem stands that a digital coin can be duplicated and spent more than once due to its digital nature. In the real world, when using cash, one cannot pay for two different things using the same money since a physical trade is performed on-site when transacting; and when transacting through a centralized entity such as a bank, it is the bank that makes sure to differentiate both payments. However, in a decentralized system, there is no central authority that controls the unicity of coins in payment. Bitcoin solves this problem with its transaction model that makes heavy use of cryptographic primitives that links transactions with each other through digital signatures and the use of asymmetric keys; ensuring that every transaction in the network is related cryptographically. Figure 2.2 illustrates Bitcoin's transaction model. Moreover, the distributivity and decentralization of the system reinforce the statement that every honest node will not allow a double spending situation since it will cause transaction inconsistencies. In this regard, as long as a majority of nodes are honest, double spending bitcoins is extremely difficult to achieve.

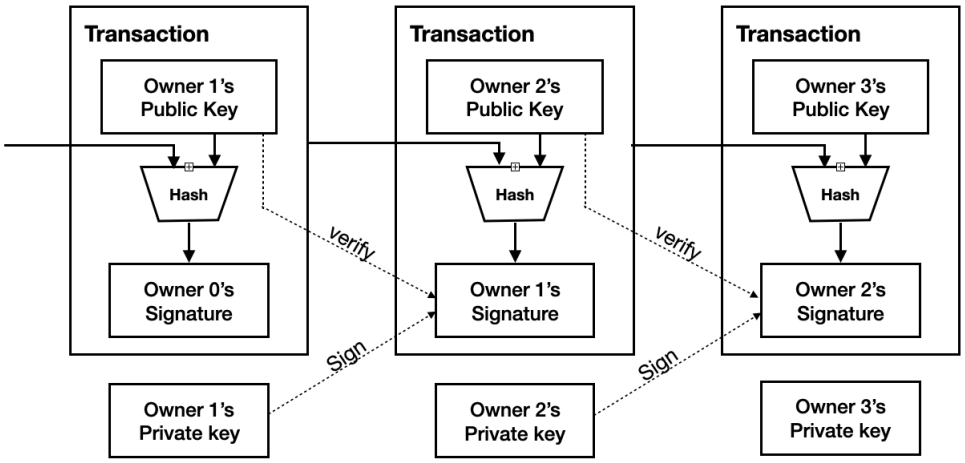


Figure 2.2: Bitcoin's transaction model.

2.1.3 Bitcoin Tokenomics

Bitcoin has a maximum supply of 21 million bitcoins; this number is hard-coded in the core system and can only be changed through a synchronized decision of all participating nodes. This limit in the maximum supply makes bitcoin deflationary and therefore, increases its value over time. This is a major difference between Bitcoin and a government-issued currency whose value is derived from the relationship between supply and demand, and whose value can be regulated by increasing or decreasing the total supply; new bitcoins cannot be created once the threshold is reached, which introduces scarcity and store of value to the digital currency.

On the other hand, the Bitcoin system encourages users to participate in the maintenance of the network by rewarding them with bitcoins for creating new blocks. Indeed, the Bitcoin network rewards miners with new bitcoins to validate transactions, which contributes to the decentralization and distributivity of the network. Furthermore, the reward (bitcoins) obtained to mine new blocks, also works as a way to introduce new bitcoins into the network, which increases the number of bitcoins in circulation since there is no central authority to issue them. By design, the number of bitcoins minted per block is reduced by half every $210,000$ blocks, which is around every four years; this allows a gradual augmentation in the total supply.

2.2 Blockchain Background

The idea of having an immutable ordered, and append-only chain of blocks came from Haber and Stonetta's work [29], who proposed a trusted time-stamping system to certify digital documents. Their idea was to use servers that would sign and certify digital documents for clients by time-stamping them and keeping a record of these certificates. However, a question about trust arises in this scheme since nothing prevents those servers from issuing a false time-stamp. Their solution is to link the entries together by including bits from the previous sequence of clients' requests in the new signed certificates. Once in a while, the most recent hash would be widely published in a public time-proof document, such as a newspaper, proving time-stamped historical evidence of the existence of that digital certificate and all certificates before it. With this action, it is possible to know whether a certain time-stamp occurred after previous sequences, providing time-ordered sequences of events. Moreover, their solution makes heavy use of one-way hash functions to provide privacy and the use of digital signatures to provide certainty. However, no matter how well designed this solution is, it has not become popular due to a lack of incentive to run a trusted time-stamping server. Furthermore, no consensus mechanism was thought to make the system truly distributed, which

centralized the information in one or a few servers making it vulnerable to a single point of failure attacks. Later on, the same authors proposed in [30] an alternative to their linking system. In this new proposal, a verification of time-stamped certificates is conducted employing hash trees or Merkle trees [31], which would reduce the storage and computation required to validate a given certificate. The main idea was to merge a considerably high number of unnoteworthy time-stamping events into a single time-stamping event (the root of the tree). This allows efficient and secure verification of the content of a large Merkle tree.

In 1990, Cynthia Dwork and Moni Naor [32] proposed a method to deter the sending of junk mail. In their proposal, the mail system would require the sender to compute a moderately, but not intractable, mathematical function to send a message. This requirement would mainly affect spammers and not common mail users since this principle is based on the fact that spammers would make heavy usage of the mail system and therefore would have to pay the cost of computational resources. The authors also propose a series of mathematical functions that could be used as a computational cost, that included extracting square roots modulo of a prime number, the Shamir signature scheme, the Ong-Schnorr-Shamir based Scheme, and Hash functions.

In 2002 Adam Back proposed Hashcash [33], a similar idea implemented in the form of an extension or plugin to be used in the email system. It would use partial hash collisions as a cost-function. The main idea is to throttle systematic abuse of un-metered Internet resources such as the email service. Cryptographic functions are designed to be collision-resistant. This means that is very hard to find two output results from a hash function such as two different inputs giving the same output. In other words, $h(x) \equiv h(y)$ where $x \neq y$. This is called a hash full-collusion and for a SHA1 function, it would take around 2^{160} tries of different y values until the same output is obtained for a given x value, which is today computationally infeasible. In the partial hash-collusion function, the k most significant bits match the output of the two functions. It's a partial collision and would require fewer tries than a full collision, simplifying the problem. The degree of difficulty to find a partial hash collision varies in terms of k. In this work [33], the computed solution would be used as a proof-of-work token from the sender to the recipient and would take non-effort from the recipient to verify its validity. This would guarantee the recipient that the received email is legitimate and comes from an honest intention since the proof-of-work attached to the email header was specially made for the receiver and attests to the amount of computational resources spent to send it. Thus, was not sent from a spammer.

In 1982, Leslie Lamport et al. [34] introduced a problem well known in distributed systems in the form of a metaphor. The idea was to expose the problem of distributed

systems to reach a consensus on a given command in the presence of faulty or malicious nodes. The problem is known as the *Byzantine General Problem* and goes as follows: A Byzantine army is divided into different units surrounding a city prepared to coordinate to take a city. A general commands one unit, and generals can only communicate with other generals using a messenger. They can only be successful if they decide whether to attack or retreat. The problem, however, is that any general could be disloyal and act maliciously to prevent a united plan. Nevertheless, if all honest generals agree on the same decision and those generals are the majority, then a united decision is reached and success is guaranteed even with the presence of treacherous generals. The problem remained unsolved until 1999 when Castro and Liskov presented the Practical Byzantine Fault Tolerant (PBFT) algorithm [35], which can solve the Byzantine General Problem in an asynchronous environment like the Internet using a state machine replication protocol.

Blockchain technology uses all these principles to create a peer-to-peer distributed ledger that is cryptographically secure, append-only, immutable, and that can only be updated via a consensus mechanism involving all peers. The time-stamp server solution [29, 30] is used in Blockchain by widely publishing a block in a peer-to-peer distributed network. Each block includes the time-stamp of the previous block, forming a chain, with each additional time-stamp reinforcing the ones before it. However, in a peer-to-peer distributed network is not enough to just widely publish a block. Cryptographic proof of the correctness of the block is necessary so other nodes can acknowledge the veracity of the information. Similar to Adam Back's Hashcash solution [33], the proof-of-work is used in this context by using the previous block hash as the seed value to find the new block's hash such as the hash begins with a certain number of zero bits. This is done by increasing a nonce in the block until a value is found that gives the block's hash the required zero bits. Once the block satisfies the proof-of-work, the block cannot be changed without rebuilding the work. As new blocks are chained after it, the work to change a block would include rebuilding all the blocks after it. The proof-of-work also performs as a stochastic consensus algorithm, in part determined by the computational power of a node that efficiently solves *the Byzantine General Problem* proposed by Lamport [34] and as well as a means to determine representation in a majority decision making. This means that as long as a majority of CPU power is controlled by honest nodes, the chain of blocks created by that majority will remain hegemonic, and therefore, honest. In this regard, the honest chain will represent the greatest proof-of-work effort and will always grow faster and outrun any other competing chains. Nonetheless, in [1] the author points out that the incentive system in Bitcoin's

blockchain helps to encourage nodes to stay honest and mitigate *Sybil Attacks* since, for a greedy node to be able to defraud people by stealing back his payments, it would require the attacker to assemble more CPU power than all honest nodes to rebuild the proof-of-work of the concerned blocks; the same power that could be used to mine new honest blocks faster than anyone else in the network and obtain the rewards. Finally, the ensemble of all these solutions allowed Bitcoin's blockchain to solve the double-spending problem that arises with a digital currency. It all comes to agreement and transparency, agreement on the order of transactions, and transparency on all the history of transactions ever executed on the network. Naturally, pseudo-anonymity is assured by using digital signatures. This way, a user only needs to sign a hash of the previous transactions with its private key and the public key of the next owner to execute a transaction.

While not every blockchain works the same way Bitcoin's blockchain does, we thought it was important to recapitulate the different breakthroughs that led to its achievement. In this regard, we consider Bitcoin's blockchain as the result of many years of research in the fields of cryptography, computer networks, distributed systems, and especially an eagerness for a change of paradigm regarding our current economic system. It is no coincidence that the appearance of Bitcoin coincides with the financial crisis of 2007-2009. It can be found in the first transaction of the first block, a hexadecimal message that goes as follow: *"The Times 03/Jan/2009 Chancellor on brink of second bailout for bank"*. The message refers to The Times' newspaper headline of January the 3th, 2009 as the British government prepared to bail out a bank that had declared itself in bankruptcy due to the economical crisis. Although this dissertation does not pretend to explain the complexity of financial behavior in the current economic system, we can relate that the existence of central authorities within any system represents a potential single point of failure.

2.3 Blockchain Generalities

Blockchain technology is another form of Distributed Ledger Technology (DLT), which is a board term for all shared databases. Although all blockchains are DLTs, all DLTs are not necessarily blockchains. In this regard, blockchain can be considered the most recent member of the DLT family. Since 2009, blockchain technology has proliferated in the digital world, evolving into more complex systems accordingly to the top-level application and the entities participating in the network. However, despite all this new diversity of blockchains, we can find that a large majority share common elements. In this Section, we describe the generalities and the elements that compose the blockchain.

2.3.1 Properties

Bitcoin's blockchain is proving itself to be a solid alternative to the traditional financial system. The main reason for its success comes from its properties that provide guarantees and assurances to the end-user. However, not every blockchain assures the same level of guarantees. Indeed, depending on certain features and types of blockchains (permissionless or permissioned), some properties may or may not be assured. Nevertheless, some properties are common to every blockchain.

2.3.2 Types of Blockchain

We can distinguish two types of blockchains: permissionless and permissioned blockchains. A permissionless or open blockchain is a blockchain that is accessible to any node. Nevertheless, restrictions may apply in some cases, such as the case of the Hybrid blockchain, which is a compromise between permissionless and permissioned blockchains. A permissioned blockchain, on the other hand, is a blockchain where participants of the network are already known and trusted. According to the nature of each blockchain project, the network will not have the same characteristics, guarantees, or performances. While a permissionless blockchain will guarantee the equity of participation and trust at the cost of lower performance, a permissioned blockchain will guarantee better performances at the cost of trust. Indeed, in a permissionless blockchain, any node can join the blockchain network even if it is malicious. Thus, a consensus algorithm capable of dealing with byzantine nodes is essential in these networks. Nevertheless, these algorithms require a higher complexity to reach consensus, which directly impacts the overall performance of the network. As an example, the Bitcoin network can only handle between 7 and 10 transactions per second because every transaction has to be verified and replicated in all nodes. Bitcoin network has a yearly average of $14,604$[1] reachable nodes. Therefore, an average of 10 min is needed to propagate a block to verify the transactions and update the ledger. Figure 2.3 synthesizes the different types of blockchain and their details can be described as follows:

- **Public Blockchain:** it is a permissionless ledger. Every node participating in this network has the same level of rights as any other node in the network. Nodes are allowed to mine and validate blocks. The code of the blockchain protocol is open source, and changes to the core system can only be achieved through collaboration and collective agreement. It is considered to be truly decentralized because there is no central authority that has clear control over the network. Its

[1] https://bitnodes.io/dashboard/ - Consult on July the 26th, 2022

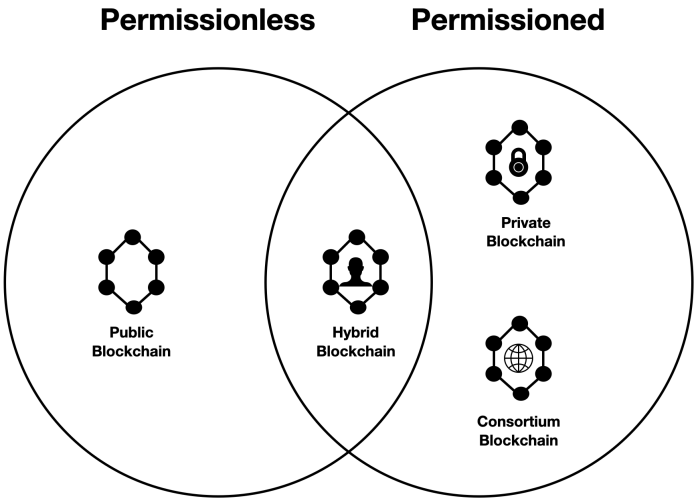


Figure 2.3: Types of blockchain.

consensus algorithm must be resilient to byzantine nodes and scalable to large networks.

- **Private Blockchain:** it is a permissioned ledger. Nodes participating in this network are known and trusted by a central authority. Different rights may be granted to each node, creating different roles or levels of participation in the network. Access to this network is regulated by an access control policy. It cannot be considered a decentralized ledger since the nodes are under the influence of a single group or a single authority. However, this type of blockchain does not necessarily seek decentralization but rather to obtain certain benefits provided by blockchain properties such as traceability, transparency, and immutability, depending on the top-level application. They have the advantage of being able to use non-stochastic consensus protocols that do not require a complex and secure mechanism to neutralize Byzantine nodes. Instead, a simple State Machine Replication (SMR) mechanism and an agreement protocol are sufficient to agree on the state of the distributed ledger, which also provides better performances.

- **Consortium Blockchain:** it is a permissioned ledger. In this type of blockchain, the register is distributed among nodes belonging to different entities or organizations that form the blockchain consortium. Nodes are authorized and known by each entity but not necessarily by the other organizations that form the consortium. In this regard, the distributed ledger is no longer controlled by a unique central authority, and therefore, trust is not presumed. It is considered to be a decentralized ledger but not of open access. Regarding the method to reach consensus on the state of the distributed ledger, a Byzantine fault-tolerant

mechanism needs to be enforced since there is no guarantee by all involved parties that a node is not malicious. A consideration of the size of the network must also be taken into account to better determine a consensus algorithm.

- **Hybrid Blockchain:** it is a permissioned ledger with a permissionless access policy. Miner nodes are usually controlled by a single or a group of entities, while other (public) nodes can join the network as validator nodes in exchange for some sort of compensation. This model allows an entity or a group of entities to be credible in the face of public opinion while maintaining control of the network and high performances.

2.3.3 Base Elements

In this Section, we detail the basic concepts that forms blockchain technology.

2.3.3.1 Miners

The term miner was first introduced to refer to a computer node in the Bitcoin network that would first validate and then compile transactions in a block hoping to be the first to solve the proof-of-work and obtain the incentive (reward) granted by the network. Today, it is used indistinctly to refer to a node participating in the blockchain consensus mechanism with the ability to propose a new block.

2.3.3.2 Validators Nodes

A validator node is a node in the blockchain network that maintains the distributed ledger. Its role is mainly to verify that transactions are valid. In the Bitcoin network, a validator node does not propose new blocks; instead, it only verifies the validity of the transactions within the blocks proposed by the miner nodes. In other blockchain networks, a validator node can also be a miner node.

2.3.3.3 Addresses

An address is a unique identifier in the blockchain. It is usually a public key or derived from it. Blockchain technology makes heavy use of asymmetric cryptography primitives to create unique identifiers in the blockchain. A user will usually own a pair of public/private keys. The public key can be perceived as a bank account number, an email, a username, or any form of unique identifier within the blockchain that can be securely shared. The private key, however, is the element that allows the transfer of a digital asset attached to its correspondent public key to another address in the

blockchain. It also works like a digital signature that allows to claiming responsibility for a transaction. In this regard, a "transfer" could be understood as an update in the distributed ledger. Addresses are the fundamental element of a transaction since they are the subject of the ledger and they serve as the origin and destination of a blockchain transaction.

2.3.3.4 Transactions

A transaction is a new entry in the distributed ledger. Depending on the use of the blockchain, the entry of the transaction can take different forms. It can be a transfer of value between two or more addresses in the blockchain, the registration of an event, GPS coordinates, data files, or simple references to any digital information. It can virtually be anything that needs to be stored in the distributed ledger.

2.3.3.5 Blocks

Blocks are data structures within the blockchain where transactions are permanently stored. Blocks are built by miners who collect some of the most recent transactions not yet validated by the network. In this regard, a block can be seen as a page of a traditional bookkeeping ledge compiling transactions within a time frame. Blocks are appended one after the other forming the blockchain data structure.

2.3.3.6 Mempool

The mempool is a common term to refer to a physical space in the node's memory where unconfirmed transactions are waiting to be included in the next block. When a transaction is created, they are broadcasted in the network for other nodes to validate. The nodes will collect all transactions that were propagated in the network and will store them in the mempool where transactions will await until either a node selects them for inclusion in the next block (the node mines the block) or drop them when another's block containing the same transaction arrives for validation.

2.3.3.7 Forks

Forks in the blockchain ecosystem can be characterized in two different manners. An intentional fork and an unintentional fork. Intentional forks are also known as programming forks. Usually, a programming fork happens when a modification in the source code of the blockchain is introduced into the system. As not every miner node implements the update at the same time, two versions of the blockchain are created until all miner nodes update the source code. If a disagreement on the implementation

of the system occurs, two versions of the blockchain likely exist. On the other hand, an unintentional fork happens when two or more new blocks are proposed to the network almost at the same time. This can only happen in stochastic consensus algorithms where the probability for a miner node to propose a new block is determined by protocol factors. This could be a consequence of a low difficulty creation block, a high block creation rate, or a small block size. Usually, there are mechanisms within the consensus protocol that foresees and mitigates these forks.

2.3.3.8 Digital Assets

A trivial definition of an asset is a useful or valuable thing. A digital asset in the blockchain context can be defined as digital and valuable information that resides in the blockchain. Today, we can mainly difference two types of digital assets in blockchain that we can define as follows:

- **Cryptocurrencies:** it is the native asset of a blockchain conceived to function as a means of exchange. It is considered a currency since it presents all money properties such as fungibility, durability, or portability. Nevertheless, there is a debate on wheatear cryptocurrencies should be considered real currencies. The most popular cryptocurrencies are Bitcoin (BTC) and Ethereum (ETH). In the case of Bitcoin, it is the native asset of Bitcoin's blockchain since its existence is intrinsic to the Blockchain itself. It was defined in [1] as a chain of digital signatures.

- **Tokens:** it is a form of a unit of value that resides in a blockchain. They are often issued using smart contracts, which is a programmable contract in the blockchain that executes automatically upon the fulfillment of the contract, it is invoked via a top-level application. Depending on the top-level application and the token standard used to create it, they can represent different forms of a unit of value. Some of the most popular token standards are those from the Ethereum blockchain, which includes the ERC-20 and ERC-721 standards, which are a set of rules to create fungible and non-fungible tokens respectively. Tokens can represent financial assets like a share in a company, voting power, a government-issued currency like the USD, or even a deed for an artwork.

It is important to point out that, according to the use case, some blockchains may not make use of digital assets; thus, they are known as *tokenless blockchains*. They are often used in cases where there is no need for a transfer of value to take place in the network. Nevertheless, other forms of transactions are viable such as sharing data between users.

2.4 Blockchain Consensus Algorithms

The problem of reaching consensus in blockchain is similar to the problem of reaching consensus in traditional distributed computing systems. However, the consensus target for blockchain is for all nodes to create a unified view of the network's entire history of transactions. In practice, a blockchain system such as Bitcoin or Ethereum, which are permissionless systems, needs a consensus protocol capable of considering several physical challenges, such as network connectivity, average network bandwidth, faulty or byzantine nodes, the size of the network, and its ability to scale in terms of the number of transactions per seconds, or the rate in which transactions are created. Such considerations are not treated in the same manner in permissioned blockchains since several elements are controlled by the consortium or the private entity, such as the bandwidth and the number of nodes. Therefore, it is crucial to take into account the functionality of each blockchain to correctly characterize its consensus algorithm.

2.4.1 The Nakamoto Consensus (Proof-of-Work)

In a permissionless blockchain like Bitcoin or Ethereum, the network model is a peer-to-peer overlay communication network on top of the Internet. Every node runs an instance of the Nakamoto protocol [1] and maintains a local replica of the blockchain. Messages are considered to be exchanged in a partially synchronous fashion, which means that the transmission delay of an eventual message is bounded to guarantee continuity in the consensus protocol. The Nakamoto consensus allows a homogenous network, meaning that there are no coordinating roles or a predefined or deterministic leader node. Each node has a probability ρ of mining the next block and each node is capable of verifying information on its own with no or minimal trust among the nodes.

For the distributed ledger to remain consistent, a common order over the transactions has to be agreed upon among the nodes. This problem is solved by broadcasting a block created by one of the miners participating in the consensus process. A block contains a set of transactions Tb that the node which created the block has chosen to commit. The block is then distributed to all peer nodes in the network. Each node verifies the validity of the transactions and the proof-of-work associated with the block. Once the block has been propagated and validated in a majority of nodes, a consensus on the state of the distributed ledger is reached. To determine which miner will generate the block, the nodes attempt to find a solution to the proof-of-work with a given probability of success. The proof-of-work consists in finding a nonce, which is a string of bytes that combined with the block header, results in a hash Hb such as Hb has a given number of leading zeros. Nodes are capable of verifying any proof-of-work with no effort by

calculating *Hb* with the given nonce that is included in the block's header. The number of leading zeros needed to be found is commonly called *target* or *difficulty*. The target is adjusted by every node every 2,016 blocks and it aims to regulate the throughput of mined blocks in the network such as there is one mined block every 10 minutes. This rule provides, time for newly founded blocks to propagate through the entire network and gives a consistent issuance of newly created bitcoins, which provides stability to the cryptocurrency. Indeed, once a block has been mined, it takes a certain amount of time for the other miners to know about the presence of the new block, and until then, miners are still competing against the new block instead of adding to it. In other words, miners are still trying to find the proof of work from a previous block while the new block is being propagated through the network. During this period, other miners might find a valid block and start disseminating their newly founded block. At this point, a race between the blocks to reach a higher number of nodes is held. Eventually, the block that reached a greater number of replicas will prevail since a greater number of replicas will start adding to that block. According to the longest chain rule defined in [1], nodes will always conserve the longest chain in the network. Therefore, if a node finds out the existence of an alternative chain longer than his, it will update its ledger with the predominant one. In this regard, all blocks that were added to the smaller chain from the moment of the fork are dumped; and with it, the computational effort to find them. Such blocks are called orphan blocks and represent waste in terms of computational resources.

We can notice that there is a relationship between the target or difficulty and the rate of orphan blocks produced in the network. Indeed, the easier is to find a block, the more competing chains will appear in the system, and thus, the more energy will be wasted due to chain reorganization. On the other hand, the harder is to find a block, the slower would become the global transaction throughput in the Bitcoin network and the less accessible would become for miners to participate in the system since more specialized hardware would be needed to mine new blocks. A 10-minute inter-time block seems to be a fair tradeoff of the Nakamoto consensus between a healthy network (reducing the number of forks and reducing the number of orphan blocks) and a low transaction confirmation time, according to [1].

2.4.2 Proof-of-Stake

Another type of consensus protocol for blockchain is the Proof-of-Stake (PoS). It is a consensus proposed by the blockchain community that aims to solve Bitcoin's energivore problem. In this model, a node participating in the consensus protocol must stake a certain amount of cryptocurrency to be chosen as the next validator node proposing

the next block to obtain the network reward. The model works under the assumption that if a person holds a stake in the system, they will never try to sabotage the system. Validator nodes are often chosen by the amount of stake they are putting into the system. In other words, the probability of proposing the next block is directly proportional to the amount of value staked. However, if only the wealthiest participants can propose new blocks, then the outcome will be rather deterministic for a distributed consensus algorithm. In this regard, there are several ways to introduce randomization into the consensus protocol to provide opportunity and fairness such as chain-based PoS [36], delegated PoS [37], and BFT-based PoS [38]. Other different variations of PoS emerge as new blockchain projects adopt this approach while keeping specific features of their system. Nevertheless, the main idea remains the same. Peercoin [39] was one of the first blockchain projects to introduce a consensus mechanism based on the concept of proof-of-stake.

2.4.2.1 Chain-based PoS

In this model, the blockchain implementing the chain-based PoS selects first a set of validator nodes according to stake capabilities and other inner policy criteria specific to each system implementing the chain-based PoS. The agreement process of reaching consensus is discretized in rounds or block generation cycles. For each block generation cycle, every validator invests a stake in the competition that is kept frozen until the end of the round. The protocol then follows the two following steps:

- Validators pick transactions from the memory pool and create a candidate block.
- Validators try to solve a simple proof-of-work associated with the candidate block as in the Nakamoto's consensus but with a personalized target and within a short-predetermined time interval.

The first validator solving the proof-of-work wins the right to propose the next block and takes the reward. The difficulty target of each validator is calculated so that it is directly related to the amount of value staked and the time that a validator has been staking in the system. Peercoin [39] and Jelurida-nxt [40] are chain-based PoS consensus.

2.4.2.2 Delegated PoS

In this model, a group of validators is chosen via a voting process. Users grant their vote by delegating their cryptocurrency to a validator node. Validators with the greater amount of stake raised are selected to participate in the consensus process. Once the

group of validators has been determined for a particular cycle, each validator creates a block in a round-robin fashion during a predefined cycle.

As happens with Nakamoto's consensus, fault tolerance is achieved if all honest nodes own more than half of the total stake value. The probability for a malicious agent to revoke a block drops exponentially as the chain grows. From an economical point of view, a malicious agent should stake more than half of the total staked value in the system to perform a 51% attack, which is economically expensive. Moreover, most of the systems based on PoS imposes sanctions on those who try to defraud the system by forfeiting their stake. Finally, although the Proof of stake model has proven to be secure and durable, there are still some concerns related to the incentive mechanism of the system that particularly impacts public blockchains. In a PoS model, it is more difficult to participate as a validator in the proof of stack model since it is often needed a minimum amount of stake to become a validator. This excludes a big proportion of stakeholders that would like to become validators but do not have enough staking power to become one. As a result, only the wealthiest have the opportunity to participate in the consensus process and therefore have more weight on the network, which leads to an excessive influence on transaction validation and sets the ground for an unfair repartition of the wealth under the motto "the wealthier you are, *"the wealthier you are, the wealthier you get"*. As a regular stakeholder, you still have the option of delegating your stake to a validator but you will depend on the good behavior of the validator.

2.4.3 Proof-of-Authority

The consensus algorithm Proof-of-Authority (PoA) was introduced by a group of developers in 2017 [41] and it claims to be an enhanced version of proof-of-stake. This model was designed to be used in permissioned blockchains using Ethereum Virtual Machine (EVM) and therefore a group of validators can be chosen as the authority of the network. The main idea is to use the validator's identity as a form of stake in the network. To this end, validators are needed to complete a mandatory process of authentication in which government-issued documents are required to ensure that block issuers (the authorities) can be held accountable. PoA relies on a set of N-trusted nodes that are the authorities. Each authority has a unique identifier and a majority of nodes are assumed honest, namely $N/2 + 1$. Each authority orders the transactions that are issued by clients and creates blocks in a round-robin fashion to distribute the responsibility of issuing blocks among all authorities' nodes. Time is divided into rounds in which each authority can issue new blocks. This scheme is considered to form part of the Byzantine fault-tolerant (BFT) consensus algorithm family but uses fewer messages. PoA has several advantages over PoW and PoS since no communication is

needed to reach a consensus between nodes. Blocks are issued in a pre-defined order and at a fixed time interval by the authorized nodes which increases the transaction rate and confirmation time. Moreover, energy consumption is much less important than in PoW. On the downside, the consensus algorithm lacks decentralization. A study in [42] claims that PoA in a permissioned blockchain deployed over the Internet and with the presence of Byzantine nodes does not provide correct consistency assurances for scenarios where data integrity is essential, and therefore, PBFT fits better in such scenarios despite its lower performance.

2.4.4 Proof-of-Elapsed-Time

This model of consensus was introduced by Intel in 2016 [43]. It uses a Trusted Execution Environment (TEE) where the code of the consensus resides. The basic idea is that each node generates a random number to determine how long it has to wait before it is allowed to propose a block. The generation of the random number is based on a mathematical distribution previously specified by the system. The TEE used by Intel is the Intel Software Guard Extensions (SGX) [44] platform and it is primarily used to ensure randomness and safety in the leader election process. When the node is allowed to propose a block, SGX creates proof of the waiting time spent by the node before proposing the block. This proof can easily be verified by other nodes with SGX technology. The node runs a statistical test to determine whether the waited time indeed follows the agreed distribution. This solution assumes that the TEE is indeed secure, however, some studies claim that SGX and other trusted computing technologies are not 100% reliable [45]. This raises concerns about the capability of the consensus mechanism to remain secure even when the TEE is compromised. Proof-of-Elapsed-Time (PoET) has major advantages compared with the Nakamoto consensus, such as its efficiency since there is no need for participants to compute expensive workload before creating a block, and its fairness since this model achieves the objective of *"one CPU equals one Vote"*. On the downside, all nodes have to use SGX technology to participate in the consensus process.

2.5 Blockchain Applied to Mesh and Ad Hoc Networks

Mesh and ad hoc networks are two types of wireless network technology that rely on device-to-device communication to create a network. In this communication model,

the network devices can communicate directly with each other without the need to transit through a router, a telecommunication base station, or even via Internet Service Providers (ISPs). In other words, there is no need for the Internet infrastructure to deliver messages to a nearby station. Mesh networks, on the other hand, are composed of wireless routers that can form a mobile wireless infrastructure, providing connectivity to clients who are connected to the nearest wireless router. Both technologies can access the Internet via a gateway station to access all the services that the Internet provides, as in the case of *Athens Wireless Metropolitan Network*[2]; where a mesh network is deployed in an isolated and geographically complicated area in Athens to provide connectivity to the local inhabitants; or the case of *Occupy Wall Street*[3] movement in 2011, where a mesh network was deployed to ensure Internet connection to protestors after the local government shut down the surrounding infrastructure where the movement was camping to deprive them of access to the Internet. Nevertheless, Internet access is no longer seen as a fundamental feature as it was for some use cases. Indeed, today the advances in data processing and data storage capabilities allow the embedment of proprietary services that are directly integrated into the routing devices. Edge computing has surely enhanced mesh and ad hoc network capabilities and has brought into the spotlight new useful use cases for these technologies. However, the protocolar level has almost not suffered from significant changes, meaning that, in the case of MANETs, we are still restrained from scaling due to long overhead and control messages from the routing algorithm to perform the process of forwarding packets to their destination. It is also worth mentioning that an ad hoc network is essentially a cooperative network, hence the operation of the entire network relies on the willingness of participants to remain active in the network. Such dependency on something as abstract as *"goodwill"* has incapacitated the adoption of these technologies by the general public since there is no incentive for a participant to remain active. On this subject, researchers have turned their attention to blockchain to incorporate this technology in mesh and ad hoc networks. One of the principal use cases for blockchain in mesh and ad hoc networks is the inclusion of an economical model that offers rewards or incentives to run a node in these networks.

In [46], the authors propose a framework model where users could benefit from blockchain capabilities to enable transparency and accountability in a mesh network. Their idea is to create an economical, sustainable and decentralized Internet access using a mesh network where participants can either pay or get paid for the usage of the network infrastructure to recover the cost of network devices and maintenance

[2]http://www.awmn.net/content.php

[3]https://mashable.com/archive/how-occupy-wall-street-is-building-its-own-internet-video

by employing cryptocurrencies and smart contracts. Their work also includes a use case study where an Hyperledger Fabric blockchain and an Ethereum-based Proof of Authority blockchain are used for comparison. They also provide results on CPU load, memory consumption, and transaction latency for both technologies. Their work highlights the high CPU load on the mesh routers as the main cause for bottlenecks since the hardware used for these types of equipment is often low-power/low-cost and struggles to handle a high throughput of blockchain transactions.

Along the same line, the authors in [47] raise the need for mesh and ad hoc networks to find a suitable way to encourage people to participate in a mesh or ad hoc network. They propose an in-network blockchain using a novel consensus algorithm called "Proof-of-Network". In this model, network participants are economically incentivized to remain active in the network. They make use of the routing algorithm information to quantify the degree of connectivity of a node. The more a node is central in the network (has a higher degree of connectivity), the more likely the node is to be selected as the next miner and obtain the network reward. The authors argue that a tokenized blockchain might encourage participants to participate in a mesh or ad hoc network while providing a reliable tool for communitarian network governance. However, the authors have no details about their consensus algorithm nor how they prevent blockchain concerns such as block collisions from their proposed consensus algorithm.

More recently, Machado et al. [48] studied and classified different data forwarding incentives in multi-hop MANETs using a blockchain technology that has been applied in the state-of-the-art. They focus their work on the assumption that MANETs require cooperative sharing of resources to enable data forwarding, but most often, the presence of misbehaving nodes tends to undermine the network reliability by acting selfishly. Here again, the authors argue that blockchain-incentivized services can deal with free-riders nodes in MANETs. As an example, in [49], the authors proposed a decentralized mechanism for low-latency data forwarding, where participants are rewarded to forward data according to different criteria. In this model, a time-locked puzzle mechanism is used to distribute rewards. Every time a participant forwards data successfully to a destination, it gets a solution to a puzzle. Hence, the first to claim the correct answer to a puzzle wins the economical reward. This mechanism is very close to the proof-of-work reward system in Bitcoin, however, the competition is about delivering messages instead of solving a cryptographic puzzle. Most similar [50, 51, 52] also offer incentives for data forwarding.

Another popular use case of blockchain in mesh and ad hoc network is the utilization of blockchain technology as a method for network authentication. The blockchain properties such as data immutability, transparency, and authenticity, can be used to

grant access to a network while preserving pseudo-anonymity. A follow on the client can then be performed to disclaim responsibility in case of misbehavior. Therefore, blockchain technology can provide not only a form of network authentication but also a tool for decentralized network management.

In [53], a blockchain-based authentication protocol for WLAN mesh routers is proposed. The authors argue that to add a WLAN mesh router with authentication capabilities using a traditional authentication method, a great deal of administrator configuration is required. Every time a WLAN mesh AP is added to the network, administrators need to configure secret keys on both the AP and the RADIUS server. In their approach, each mesh AP runs a blockchain where immutable information containing key elements for authentication is stored. The idea is to remove the centralized RADIUS server and to make it distributed using blockchain. The authors implement their solution in a test-bed environment and evaluate the average authentication delay for different numbers of wireless hops. Results show that their blockchain-based authentication method can reduce the authentication delay in a multi-hop environment when compared to a traditional RADIUS authentication method.

In [54], a mesh network for IoT devices using a blockchain is proposed. In their model, there are two types of nodes; the IoT devices that can create transactions and execute smart contracts and the miner nodes, which can mine new blocks. The collected data from the IoT devices are registered as a transaction and are eventually included in the blockchain. In their proof-of-concept, the authors used a Raspberry Pi 3B as the IoT device. The authors argue that their scheme provides better information management for IoT systems while enhancing security.

Other researchers focus their attention on exploiting blockchain's properties to create a trusted management system at a network level. Routing strategies in mesh and ad hoc networks are known to have particular needs due to their dynamic and unstable topology and are far from being considered secure. Next, we present some representative research where blockchain is used to deal with the trust issue in mesh and ad hoc networks at the network level.

In [55], the authors propose a blockchain-based trust management system in MANET using a lightweight consensus algorithm called Delegated Proof-of-Trust (DPoT). In this scheme, policy-based and reputation-based trust management are used to calculate a node trusted value. The authors use the additive increase multiplicative decrease scheme for their proposal. To evaluate whether a node is malicious or trusted, the authors use DCFM [56] over OLSR [20] as a representative detection mechanism for identifying malicious attackers. When a node is identified as malicious, the node is flagged, and its

information is shared across the network using the blockchain. The other trusted nodes in the network will remove their connections with the malicious node. This process efficiently excludes the malicious node from the network and makes the transition of information more secure. The blockchain used is a public blockchain that is maintained by all nodes in the network. Miners are selected based on their trusted value, and as consequence, only trusted nodes can participate in the DPoT consensus algorithm. A node is elected leader by following the bully election strategy [57].

From another perspective, the authors in [58], propose a blockchain-based approach to enable Unnamed Aerial Systems (UAS) such as a swarm of drones, to collect, aggregate and redistribute routing information in a secure and trustworthy manner. Their goal is to find a solution to the Route Disclosing and Pollution Problem [59] that comes from UAS routing strategies, where malicious nodes can analyze the network topology of the entire network and broadcast fake routing tables to interrupt the whole network by exploiting proactive or reactive routing strategies. Their solution includes the construction of a sub-graph of the entire network where each node in the sub-graph can participate in the consensus process. The blockchain is distributed among all nodes and it makes heavy use of smart contracts. Each node will submit a view of its network topology that includes its neighbors and proof to the smart contract. A smart contract verifies the integrity of the view and aggregates the different views to form a unified view of the network. Finally, cluster-head nodes are granted by smart contracts with sufficient routing information to deliver a packet but the entire network topology is not disclosed. Evaluations show that their approach outperforms conventional methods in preserving routing information confidential and secure.

Along the same line, the authors in [60] propose an enhanced version of the AODV routing algorithm using blockchain technology. Their solution does not use a blockchain *per se*; instead, they use Haber and Stornetta's [29] time-stamping trusted system and Merkle trees to establish a chain of nodes from the source to the destination. A smart contract reads the generated chain of nodes and selects a path toward the destination based on routing conditions to provide QoS.

So far, we have reviewed the literature on the usage of blockchain in mesh and ad hoc networks from a general perspective. Nevertheless, there is a domain within the MANET that is gaining "momentum" in recent years: Vehicular Ad hoc Network (VANET), which is another form of mobile network that defines communication between vehicles and roadside units (RSUs) as V2V and V2R. It may assist drivers in the driving experience by picking the shortest route based on traffic optimization, identifying the closest gas station, preventing accidents, and creating an ad hoc network that facili-

tates communication. Many researchers have been interested in integrating blockchain technology with VANET, mainly to overcome the privacy and security limitations by providing a tamper-proof network system but also to accelerate the adoption of smart vehicles that can efficiently minimize the environmental impact in our ecosystem. For the global industry, corporations are looking forward to relying on an intelligent transportation system that ensures transparency, authenticity, reliability, and immutability of the information. Like in mesh and ad hoc networks, VANET also needs to ensure that the vehicles participating in the network are correctly authenticated and potentially identifiable to be held accountable for any misconduct.

In this regard, the authors in [61] propose a blockchain-based decentralized authentication approach for VANET. Their work intends to replace the traditional PKI authentication method in VANET with a distributed ledger such as the blockchain. In their architecture, three main entities participate in the authentication process: Authentication Parties (AP), the RSUs, and the individual vehicles. The blockchain used in this scheme is based on Hyperledger Fabric, which is distributed among the AP and the RSUs. At first, each vehicle gets from an AP a set of public-private key pairs and a pseudo-ID. After the issuance of the network credentials, the AP adds a registration transaction to the blockchain. At this point, every entity in the architecture, including the vehicles can access the public ledger to verify the information. In the case of V2V communication, a vehicle can verify the authenticity of Broadcast Safety Messages from another vehicle. In this scheme, only the Authentication Parties have the right to issue transactions in the blockchain; thus, a misconduct report has to come first from RSUs. A very similar architecture is presented in [62], with the difference that smart contracts are deployed at the moment of authentication and key pairs are loaded on the onboard unit.

In [63], a self-managed and decentralized system for VANET is proposed using the Ethereum blockchain. In this model, RSUs are equipped with a decentralized application interacting with the Ethereum blockchain. Authentication in the network is performed via the user's Ethereum address. Relevant control messages and registration transactions are included in the Ethereum blockchain and therefore, users need to pay small fees for each transaction. While the proposed architecture allows vehicles to easily join the network, there are several downsides to this proposal, such as the energy used to mine new blocks and the uncontrolled cost of transaction fees.

Another similar approach in [64] uses the Ethereum blockchain as a cooperative protocol for authentication in VANET. In this scheme, an Internet of Vehicle (IoV) can register in the Ethereum blockchain to provide forwarding services to another IoV. The usage of the Ethereum blockchain guarantees the integrity and preservation of

the privacy of the IoVs. The authors validate their results through numerical analysis, and their results show that the proposed architecture successfully increases the system throughput and decreases the packet dropping rate.

In [65], the authors propose a decentralized trust management system based on blockchain for rating the social behavior of vehicles in VANET. In their framework, they assume that different RSUs might use different blockchain technology, and therefore, a cross-blockchain technology will have to be used to conciliate all the information. A cross-blockchain is a technology that allows transferring assets from one blockchain to another without altering basic properties. The authors proposed a cross-blockchain based on the Tendermint [38]consensus algorithm as the interoperability between different blockchains that might co-exist in VANET. In their vision, a vehicle can send another vehicle's trust value to an RSU, which will later be sent as a transaction to the blockchain. If a vehicle's trust value needs to be verified by an RSU or another vehicle, they can do so by querying the blockchain. However, if the information is not in the RSU's blockchain, it can be solicited using the cross-blockchain. The authors argue that this framework allows a better form of decentralization while reducing latency and improving transaction speed. Other blockchain authentication solutions for VANET include [66], which uses a Conditional Privacy-Preserving Authentication (CPPA) protocol using a blockchain instead of a PKI-based protocol.

All the solutions proposed in this Section aim to improve the overall experience in mesh and ad hoc networks by efficiently using blockchain properties for the benefit of these networks. At the network layer, different propositions use blockchain technology to enhance the process of routing packets to their destination or as a mechanism to prevent and remove malicious nodes in the network. At the upper layers, the utilization of blockchain was mainly proposed as a way to improve the authentication process of nodes in the network but as a form of managing the nodes in a decentralized fashion. Then, blockchain technology was proposed to support Decentralized Autonomous Organizations (DAOs) in mesh and ad hoc networks in a more commercial approach. We have also covered the utilization of blockchain as a platform to facilitate the distribution of incentives to encourage people to use these types of networks. Finally, we appreciated different blockchain applications for the management and enhancement of UAS, VANETs, IoT, and IoV.

Nevertheless, none of them takes into account connectivity problems, and as consequence, they all assume full connectivity among peers. In other words, the possibility for these networks to split into two or more independent networks or to merge as a single network is not contemplated as can often happen in MANETs. Indeed, until

now, we understand the blockchain as a peer-to-peer distributed system that maintains a shared ledger in a connected network. Of course, the nodes are always capable to connect and disconnect at will from the network, but the blockchain is still maintained by the other connected nodes. In this paradigm, we assume that a split in the network results from an anomaly in the network connectivity that triggers an unwanted event that alters *the status quo* of the system; since a split in the network implies a fork in the blockchain. Today, most blockchain projects solve the fork problem by choosing one branch and ignoring the others. As we have already explained, Bitcoin's blockchains use the longest chain rule on the basis that the longest chain of blocks accumulates more processing power. Other projects use a similar approach with peculiar differences but the objective is the same: choosing one chain among the others. This solution makes perfect sense in a connected network where transactions are interdependent since having two or more versions of the same ledger have consequences on the data integrity. As an example, a user would only have to change between the networks to perform a double-spending attack in a Decentralized Finance (DeFi) application. Moreover, the aggregation of the transactions that took place in the other chain would be extremely complicated to achieve. Therefore, a unified version of events is necessary for certain use cases, even at the cost of losing the effort spent in constructing other chains.

In the case of the solutions reviewed in this Section, many solutions make use of DeFi applications in the context of mesh and ad hoc networks, which is the case of [46, 47, 48, 49, 50, 51, 63, 64] where blockchain is used in-network. The utilization of cryptocurrency for incentivization implies the maintenance of an accountable ledger and therefore, the necessity of a unified common history of events. A split on this network will result in the continuous application of the longest chain rule, which is not optimal for networks that are meant to be dynamic. Likewise, the authentication application in mesh and ad hoc networks using blockchain [53, 54, 64, 62] needs to access a unified and updated version of the ledger at all times. In the case of a split in this scenario, all new nodes that were authenticated during the fork and which authentication register happens to be in the smallest chain will be lost after applying the longest chain rule, leading to trust problems and data inconsistency during and after the fork. It is important to point out, that changes in the network topology already cause losses of information due to its dynamic nature. This behavior is not only common but is intrinsic in mobile ad hoc networks.

In our solution of which this thesis is the object, we propose a solution to the split and merge problem for blockchain technology in mesh and ad hoc networks. Our solution is based on the concept of a graph of blocks and no longer follows the traditional

blockchain structure. In this regard, we focus our attention to address the problem of having legitimate alternative chains and guaranteeing data consistency in each network partition. Therefore, our work is adapted to dynamic networks. Next, we present a state-of-the-art of relevant work that has incorporated the concept of a graph of blocks as another form of blockchain structure.

2.6 Direct Acyclic Graphs and Blockchain

A Direct Acyclic Graph (DAG) is a mathematical concept coming from Graph Theory. It is composed of a set of vertices and a set of edges, such as each edge is directed from one vertex to another in a way that those directions will never form a closed loop. In recent years, and partly because of the popularization of blockchain, DAG-based data structure has been used as another form of DLT in the same way that blockchain is used. In that sense, DAG-based data structures are trying to solve the same problems that blockchain technology solves. While blockchain builds a chain of blocks as a data structure, DAG-based DLT creates a graph of nodes or blocks to form its data structure. During our research, we noticed that the primary motivation to create such graphs was mainly correlated to the need of improving the transaction throughput limited in blockchain, which significantly differs from our motives to create a graph with our Blockgraph. As far as our research is concerned, none of the protocols presented in this Section solves the problem of split and merge in MANETs. We can differentiate two main forms of DAG-based DLT: Block-DAGs and Blockless-DAGs.

2.6.1 Block-DAGs

Block-DAGs are a DAG-based DLT that uses blocks to compile transactions in a distributed network as is done in the blockchain. Figure 2.4 illustrates an example of the structure created by Block-DAGs. One of the main barriers to blockchain scalability is the orphan rate problem. Orphan blocks are blocks that are eventually not included in the main chain or the longest chain due to unavoidable network propagation delay or the inefficiency of a consensus algorithm to terminate on every single block. The main motivation to avoid the creation of orphan blocks or at least, minimize its rate is to maximize the transaction throughput performance, and therefore, the overall performance of the blockchain. Nevertheless, eliminating orphan blocks is not an easy task. In most blockchain applications, transactions are co-dependents, and being able to determine a specific order and validate each transaction to ensure that no rules are broken, is of major importance. An example of this issue would be the double-spending

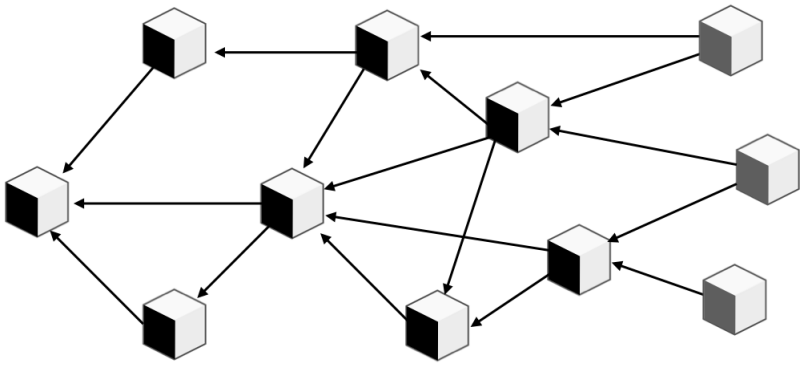


Figure 2.4: Example of a Block-DAG data structure.

problem. Detecting and ensuring these properties requires a fast consensus that can keep up with the utilization of the network (transaction rate). Thus, orphan blocks are eventually created, either because they contain fraudulent transactions, because the correct order of events is not respected, or because the rate at which blocks are created is higher than the capacity of the blockchain system. These blocks that are never included in the blockchain data structure imply costs and a waste of computational resources that should be minimized. The cadence at which orphan blocks are created depends on the block creation rate or its size. This phenomenon is explained because by the time a new block propagates throughout the network, other new blocks which do not reference the block that is being propagated are likely to be created, resulting in spontaneous forks in the blockchain. For that reason, Satoshi's original system implements mechanisms such as the longest chain rule and PoW consensus as we have explained in Section 2.4, which regulates the block creation rate to accommodate the network propagation needs and minimize orphan blocks. However, even if those mechanisms are efficient in securing the blockchain and guaranteeing strong data consistency, its performances are limited by those same mechanisms. At present, blocks are mined in the Bitcoin network every 10 minutes and the average confirmation time for a transaction is between 3 and 7 minutes [4].

Block-DAGs are considered a solution to the orphan rate problem in the blockchain. It modifies the block structure so that a block can refer to multiple predecessors allowing a newly created block to refer to several branches created by forks. However,

[4]https://www.blockchain.com/charts/avg-confirmation-time

by incorporating different branches, some blocks may have conflicting transactions. To avoid this problem, several approaches are proposed in the literature. Zohar et al. from the Hebrew University proposes to use Block-DAG systems, such as GHOST [67] and INCLUSIVE [68], as a solution to increase the transaction throughput in Bitcoin's blockchain while maintaining security. In addition, they have proposed ordering algorithms, such as SPECTRE [69] and PHANTOM [70], to ensure data consistency and prevent double-spending attacks. In these protocols, new blocks are allowed to reference all known childless blocks by a miner node. In other words, a miner can propose a block that refers to all known blocks at the end of a branch. This process creates a DAG-based data structure where blocks are partially ordered. Thus, it is only needed to define a particular order over blocks created in parallel to define an order over the transactions. Once an order is defined, it iterates the DAG and eliminates transactions that are in conflict. Their algorithm first identifies the blocks that were created by honest nodes with a certain probability ρ, that were likely created by misbehaving nodes are separated from the main DAG. In this way, the authors argue that as long as the hashing power remains among honest nodes, including every block in the DAG, it allows for obtaining better performances and increases the block creation rate since their algorithm identifies conflictive transactions as *a posteriori*. Nevertheless, limitations on the confirmation time and the exponential use of the memory space are still a concern for these protocols. Moreover, there is a lack of self-regulation of relevant parameters in the PHANTOM [70] protocol which raises questions about its decentralization and distributivity.

Another work that uses block-DAG is [71]. In this scheme, nodes in the network jointly create a DAG structure and subsequently interpret it as a byzantine tolerant consensus mechanism. Nodes may or may not share the same view of the DAG, and therefore an ordering process is needed. To this end, a consensus mechanism, which is a simplification of the PBFT protocol, elects particular nodes as *"Famous witnesses"*, which are trusted nodes to facilitate ordering and guarantee consensus finality. The network can reach high transaction throughput if we assume a partial synchronization of the nodes via a synchronization protocol and a highly connected network. Decentralization, however, is weakened by the use of *Famous witnesses* since those nodes can eventually collude to perform fraudulent actions. Additionally, the set of trusted nodes might represent a single point of failure and could easily be the victim of a targeted attack putting the whole network operation at risk.

2.6.2 Blockless-DAGs

In the work of Bencic and Zarko [72], a comparison between blockchain and blockless-DAG revealed that the main difference between these two technologies is the choice of structural design. While blockchain gathers transactions to form a block, blockless-DAG forms nodes or units, where a single transaction is stored. Nevertheless, this statement is not completely accurate since the problem of having two conflictive transactions that were committed simultaneously still represents a challenge for data consistency in the DAG-based paradigm. Many works use DAG-based structures, not only as a choice of structure design but also as an alternative to finding more efficient ways of improving scalability in DLTs. But scalability does not come without a trade. In some works, scalability is obtained at the cost of losing strong data consistency or by granting special privileges to certain nodes, and therefore, losing in decentralization, or by considering important computational resources to execute complex ordering algorithms. Figure 2.5 illustrates a Blockless-DAG data structure.

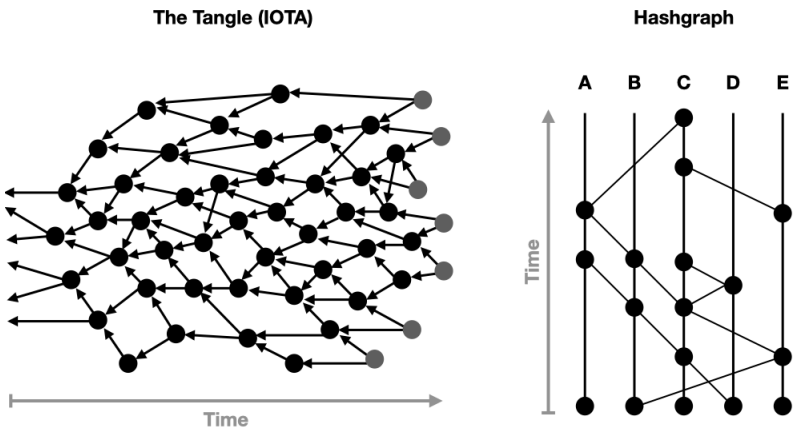


Figure 2.5: Example of a Blockless-DAG data structure.

Authors in [73] use the concept of blockless-DAG to create a DAG-based data structure called *the Hashgraph*. In this data structure, records of gossip about gossip are stored to create correlations about who gossiped to whom and in which order. In that sense, the history of all gossip events can be illustrated by a DAG, where vertex are the events, that might or might not, contain transactions, and downward edges are either a reference to the last known event of the node or a reference connecting to the immediately preceding gossip from another node. The consensus algorithm is asynchronous, nondeterministic, and tolerates byzantine nodes; it is based on the

hashgraph itself. Since all nodes in the network are aware of the chain of gossiping and in which order, no real consensus is required, since everyone already agrees on the chain of events. However, if a conflictive transaction is detected, the consensus algorithm uses the concept of virtual voting. In this scenario, no real votes are cast or transmitted over the network, instead, every node that owns a copy of the hashgraph can determine what would each node have voted if they had been running a traditional Byzantine agreement protocol. Thus, nodes reach a consensus without the need of sending a single message. Their solution assumes that at least 2/3 of the nodes are honest, and strong connectivity among nodes exists. While the hashgraph is one of the first DAG-based data structures for DLT, their solution has problems determining the order of events when two or more parallel events occur.

With most resemblance to Hashgraph, authors in [74] proposed a communication history DAG for distributed systems in asynchronous networks. Their solution uses a DAG-based structure to maintain communication history among peers. They reach consensus by using an Asynchronous Byzantine Fault Tolerant consensus. In their work, they divide the consensus process into rounds and can guarantee data consistency up to a k-round. While this model is decentralized and distributed, it does not scale well in large networks, as is often the case, in financial applications.

Another blockless-DAG DLT is IOTA [75], a cryptocurrency for IoT. The main objective of this project is to maintain a distributed ledger of micropayments for IoT systems. In this work, transactions are added atomically in a DAG structure called *the tangle*. Hence, every transaction represents a vertex in the tangle. To validate a transaction, a node has to validate two previous transactions that are the immediate predecessors of the new transaction. Conflicts among transactions are solved by running multiple times a tip selection algorithm in each node. The algorithm selects the conflicting transactions that are more likely to be indirectly approved. Each transaction is assigned a weight, and a cumulative weight is calculated from past transactions. The higher the weight of a transaction is, the more the transaction is considered secure. This is called confirmation confidence. To avoid the possibility of overthrowing a transaction from the tangle and ensure finality, a trusted centralized entity, called the coordinator, under the control of the IOTA Foundation, issues a milestone transaction every two minutes to fully guarantee the confirmation confidence of previous transactions. This centralized solution that ensures consensus finality is claimed provisory by the IOTA Foundation; however, there is no clear timeline to replace this solution.

Similar to the tangle, the authors in [76] presented Nano, a cryptocurrency that achieves consensus via a balance-weighted vote on conflicting transactions. In this model, each user owns an account and every account has a transaction chain similar

to a blockchain. Nodes participating in this network keep track of account balances rather than transaction amounts, which reduces memory complexity but eliminates transparency and auditability properties. Voting power is gained by proving commitment to the network. While Nano seems to scale in transaction throughput, it is clear that the network is not completely decentralized and therefore vulnerable to greedy conduct.

In a more general use case, Byteball [77] allows a decentralized system for storage and transfer of value of arbitrary data. It uses the blockless-DAG paradigm to store transactions. Every transaction is linked to past transactions. The more a transaction is located deep in the DAG, the more the transaction is confirmed. Byteball achieves consensus by relying on the concept of the main chain, which is a chain of transactions selected by an algorithm that represents the principal chain in the DAG. The authors argue that as long as honest, reputable, and user-trusted *"witnesses"* agree on the main chain, it is possible to define a total order of events and therefore identify conflictive transactions. A conflict is solved by choosing the closest transaction to the main chain. Nevertheless, a conflictive transaction is hardly detected at the time it was issued. It may take some time to detect conflictive transactions, and therefore, it can impact real-time applications.

More similar to our solution, the authors in [78] proposed a partition-tolerant blockchain for IoT that uses a DAG-based structure called *Vegvisir*. In their approach, the DAG-based structure is created due to IoT's lack of connectivity or a high block creation rate. Nevertheless, the number of previous block references in a block is limited to two. To face the problem of short storage in IoT devices, they use a support-chain, which is a regular blockchain in the cloud where devices can offload part of their DAG to save space. However, no detail on the governance of the support-blockchain is available. Very different from our approach, the creator of a block is the originator of all transactions in the block, therefore, there is no need for transactions to be propagated in the network. In this regard, Vegvisir only supports Conflict-free Replicated Data types as transactions, which eliminates the need for a partial or a full ordering of transactions; and thus, the need of employing any sort of consensus algorithm. Moreover, the authors only consider partitions that are limited to short periods and do not consider node mobility.

Other solutions to Bitcoin's scalability problem besides Block-DAG that have gained momentum are the Lightning Network (LN) [79] and other layers 2 solutions such as Polygon/MATIC [80]. There are three desirable properties that a blockchain should have, decentralization, security, and scalability. As the blockchain trilemma states, a

simple blockchain architecture can only achieve two out of three. For permissionless blockchains such as Bitcoin and Ethereum, scalability is sacrificed. Therefore, Layer 2 solutions have positioned themselves as the primary solution to solve scalability issues in blockchain technology. In the case of the LN for the Bitcoin Network, scalability is achieved by limiting the number of transactions that are broadcasted on-chain. Instead of recording every transaction between regular transactors, micropayment channels are created between two entities that commit funds in a 2-of-2 multisignature Bitcoin address. The two entities can then transact off-chain unlimitedly as long as both agree to keep the channel open. In this regard, only the initial and last transactions of a channel are propagated in the Bitcoin blockchain. Any attempt of fraud will end in the 100% redemption of the funds for the affected party. Moreover, there is no need to open a dedicated micropayment channel with every entity to transact. It is possible to send funds to a destination via multiple intermediaries without the risk of theft of funds.

We differentiate between two main types of DAG-based systems: Block-DAG and Blockless-DAG. While Block-DAG compiles transactions in blocks or other structures that ensembles transactions, blockless-DAG links transactions together without the need for a container structure; which eliminates the need to order transactions within a block but still requires performing an ordering among the other transactions. Block-DAG solutions were mainly proposed as a solution to the orphan rate problem in Bitcoin's blockchain, while blockless-DAG came as a simplification of the block structure to facilitate ordering and as an alternative to Nakamoto's consensus algorithm. In any case, as we explained from the work that was presented in this Section, scalability can be achieved but at a cost of losing strong consistency. Moreover, security performances have not been fully validated since DAG-based structures are still considered in their early stages of development.

Despite these interesting works on DAG-based DLTs, they focus mainly on scaling the transaction throughput and do not consider network connectivity, except for [78]. Hence, these solutions are not suitable for mobile ad hoc and Mesh networks. Although the authors in [78] consider IoT devices' connectivity, none of the previous solutions can deal with topology changes. Therefore, the main goal of this thesis is to provide mobile ad hoc and mesh networks with a blockchain-like data structure capable of dealing with mobility and changes in the network topology. As a form of contrast, our DAG-based solution (Blockgraph) does not use its DAG structure to increase the transaction throughput in some blockchain systems, nor as an alternative to achieve consensus on a distributed system. Instead, our solution obtains its DAG data structure to solve the split and merge problem in mobile nodes. To this end, different branches

in the DAG, represent partitions in the network topology that occurred due to node mobility. To our knowledge, no work in the literature presents a DAG-based DLT solution to solve the split and merge problem in mesh and mobile ad hoc networks.

2.7 Consensus for Mobile Networks

Reaching consensus in distributed or decentralized systems is a problem that has widely been studied by researchers. Basically, the consensus problem consists of a situation in which several spatially distributed nodes, agents, or processors must reach a common state on a value, a ledger, an output, or a sequence of commands, but without the intervention of a central coordinator. Most of the consensus algorithms studied in the literature are considered static in the sense that the communication network among processors is reliable. However, the dynamic nature of mobile networks requires tight coordination among nodes. In this Section, we present a state-of-the-art on consensus algorithm for mobile networks. We are especially interested in those algorithms that are partition-tolerant in mobile networks. Nevertheless, few of them highlight this property.

Authors in [81], propose a dynamic consensus for mobile networks. In this work, the authors focus on solving what is known as the distributed average consensus problem, in which, given a graph of nodes, each node has a number associated with it and the goal is to find the mean value of all these numbers across the graph, based only on communication between neighbors on the graph and on local computation. Their solution includes a dynamic consensus based on adjacent matrices from neighbor nodes to form a Laplacian matrix from which a Laplacian consensus dynamic is inferred. They provide proof that a Laplacian dynamic is not sufficient to provide consensus tracking in the case of network splitting and merging and propose modifications to the equations to make the consensus partition-tolerant. Other works solving the average consensus problem in distributed systems are [82, 83, 84]. While these consensus algorithms can reach consensus on a single value in a distributed network, they can not maintain an ordered history of events needed in DLTs. This is because most of the applications using average consensus algorithms such as [85, 86] are implemented for Mobile Wireless Sensor Networks (MWSN), which are often constraints at the hardware level in terms of computational, power, and memory resources, even worse than MANETs.

In [87], the authors propose a stabilizing consensus in mobile networks for biologically-motivated systems. In their approach, the authors argue that a stabilizing consensus does not need that each node commits to a final output at some point. Instead, they consider that each node can change its current output as the execution proceeds. Their

stabilizing consensus is mainly used to converge in a time-sensitive value that could represent a direction, speed, acceleration, or localization in a flock of mobile nodes. Since the output is sensitive to change, this stabilization implies a constant modification of the agreed output which only achieves a weak consistency and leaves no traces of the history of events.

In another work, Sun et al. [88] implement a distributed consensus algorithm for clock synchronization in dynamic networks. Their goal is to synchronize every node in a mobile network in a distributed and decentralized manner using timing messages that are broadcast randomly. A consensus on a single time is never reached, however, each node manages to synchronize with its direct neighbors providing accurate overall time synchronization. In this case, the consensus finality is at a system level and not at a data user level.

DREAM [89] is a data replication technique for real-time ad hoc mobile databases that uses a State Machine Replication technique. The solution considers the mobility of the nodes as a factor in the managing of the data. In their scheme, not every data is replicated in all servers, instead, servers are selected based on an algorithm to determine which data should be replicated. The solution reaches a consensus on the data that is to be replicated but at a cost of centralization. A hierarchy of nodes is necessary for this scheme to succeed since each node plays a role. Moreover, the synchronization process might alter previous entries of the database without keeping a trace of the evolution of the data.

In the same line, [90] proposes a replication extended state machine suitable for MANETs. In their work, they assume one client at a time submitting transactions into the system. The system achieves consensus in 4 phases and uses a hierarchical structure. Nevertheless, there is no consideration of the split and merge problem.

The consensus problem in mobile networks is a well-studied case in the field of distributed systems. Nevertheless, several restrictions limit the solutions that we have presented. On one hand, a solution to the consensus problem is considered in resource-restrained networks such as the case of MWSN. Here, a simplification of the application requires only reaching a consensus on a single value without taking into account conflictive transactions or ordered events. On the other hand, an SMR approach is used where a hierarchy of nodes exists to manage the data replication in the mobile network. These trade-off between node centralization in the SMR model and the single data convergence in the average consensus model limits the potential user applications in mobile networks. In the case of more traditional consensus algorithms such as Paxos [91], PBFT [35], or RAFT [19], there are no solutions applied to mobile networks

using these algorithms. The main reason is that these consensus algorithms assume high connectivity and a small size network; something rather scarce in mobile and dynamic networks. As far as we know, C4M, the solution we present in this dissertation, is the only consensus algorithm designed for blockchain-like DLTs and particularly for our Blockgraph, that works in a mobile environment with the flexibility to support frequent changes in the network topology and that is partition-tolerant.

2.8 Chapter Summary

We have introduced in this chapter the motives and technical backgrounds that are at the origin of blockchain technology and we came to better understand blockchain as a contribution to distributed and decentralized systems. We explained the different types of blockchains, their properties, and how they are related. We highlight the fact that blockchain technology not only is relevant to solving the problem of double spending that arises in financial applications but it also finds its place in all sorts of decentralized applications as we saw in our state-of-the-art. We introduced the main consensus algorithms used in blockchain technology and how are they related to the different types of blockchain. Furthermore, we have contextualized our contributions by providing the state-of-the-art in the different areas of our research. In the next chapters, we further detail our contributions to the field by providing the necessary technical elements, the details of our implementations, and the results.

Chapter 3

Blockgraph

This chapter presents a detailed explanation of our Blockgraph solution; a framework that allows the utilization of a blockchain-like data structure in mesh and mobile ad hoc networks. We present the network characteristics in which our Blockgraph is contextualized, we break down each component of our framework and explain its function and their correlation; we provide details on the construction of the data structure, and explain *the merge synchronization procedure*. Blockgraph inherits important blockchain properties, and therefore, it also has the same guarantees as those of the blockchain, which provides mesh and ad hoc networks with a blockchain-like technology adapted to the needs of the network.

3.1 Introduction and context

Advances in Computer Sciences have enabled researchers to provide better solutions to contemporary problems and innovate with new solutions to improve the quality and life experience of all of us. One of the research areas within computer networks that has constantly been evolving is Mobile and Ad hoc Networks (MANETs). As an example, let us take the different areas within MANETs that have aroused interest. Vehicular Ad hoc Networks (VANETs) are a relatively recent type of MANET that enables effective communication between vehicles or roadside equipment; VANETs are meant to enhance security and provide a better mobility experience to users. Flying Ad hoc Networks (FANETs) are composed of a set of Unmanned Air Vehicles (UAVs) that are used to monitor remote geographical areas and other military and civilian applications. FANETs enable multi-UAV communication, which enhances cooperation and collaboration between UAVs. Other examples of MANETs include Smart Phone Ad hoc Network (SPAN), which creates peer-to-peer network communication without relying on cellular carrier networks; Internet-based Mobile Ad hoc networks (iMANETs),

which support Internet protocols such as TCP/UDP and IP; and Military or Tactical MANETs, which has a particular emphasis on data rate, real-time demand, fast routing during mobility, and security. All these networks consist of a set of mobile nodes connected wirelessly in a self-configured, self-healing network without having a fixed infrastructure and with the possibility of frequent changes in the network topology due to nodes' mobility. In parallel, significant advances in computing processing, volume storage, and radio capabilities have significantly enhanced the use of MANETs by integrating more advanced and sophisticated embedded services and applications [92, 93]. Indeed, Multi-access Edge Computing (MEC) has provided MANETs with an all-in-one set of resources to run an independent network where services and applications are embedded and distributed in the nodes. In this regard, it is possible to conceive a full stack of applications and services, like those of the Internet, that can easily be deployed in the form of a private network. We can certainly foresee several use cases where corporations, civilians, or private entities could make use of these private networks to carry on with their missions while ensuring security and data privacy. MEC technology also brings some advantages, it allows running applications and performing processing tasks closer to end-users, which reduces latency and network congestion, and applications perform better in general. However, in the context of MANETs where nodes are mobile and distributed, edge computing nodes need to remain synchronized to ensure data consistency. To this end, researchers from distributed systems field have also done their part to contribute to the enhancement of these systems. An example of the production of the combination of technologies mentioned above is Green Communications [94], which produces mobile computer nodes with Edge computing capabilities in restrained devices for mesh and mobile ad hoc networks.

In this context of high innovation, resources enhancement, and network dynamicity is that we are interested in providing MANETs, with a blockchain-like technology that allows legacy and new applications to benefit from blockchain properties and guarantees, that until now, have been reserved for connected networks. In section 2.5, we gave a detailed state-of-the-art on blockchain applied to mesh and ad hoc networks, we concluded that section by arguing that blockchain technology is without a doubt useful to MANETs for system services and users' applications. However, every proposed solution using blockchain technology in MANETs, either assumes an external blockchain connected to the MANET via a gateway or assumes a certain number of non-mobile nodes with higher computational resources to maintain the blockchain or does not assume network partitions at all. In this regard, we consider it important that MANETs can handle their own blockchain "in-network", and consider network partitions.

3.2 The Split and Merge Problem

A mobile ad hoc network is a network wherein nodes communicate over a sequence of wireless links that can include one or more intermediate nodes; since nodes are mobile, partitions in the network are likely to happen. Network partitions result from a wireless link disconnection or failures that occur when two previous communicating nodes move such that they are no longer within the transmission range of each other or by the presence of high interference. Likewise, wireless link formations occur when two nodes that were too far from each other to communicate move such that they are within the transmission range of each other. In a mobile environment, those are a common phenomenon and are commonly referred to as *split* and *merge* respectively. Now, let's assume that every node in the network participates in the maintenance of a blockchain; as we have already explained, a blockchain is a particular form of distributed database that should ensure data consistency in all replicas. When a split happens in the mobile network implementing a traditional blockchain conceived for a connected network, the blockchain will treat the split as it treats a fork; and when the networks will merge again, one of the two branches resulting from the fork will be removed (e.g., by applying the longest chain rule) since both branches cannot coexist in the same chain. This behavior seems perfectly normal in a traditional blockchain where the data structure of the blockchain is a unique chain of blocks, and the network infrastructure allows high availability and connectivity. However, in a mobile ad hoc network, where the partition of the network is the result of intended mobility, we can no longer apply the same paradigm since assuming that the mobility of the nodes has a purpose, a network partition implies application and system independence over its distributed system. Of course, it is important to notice that applications in a mobile ad hoc network can vary from connected network applications. Finally, we are left with the question: How do we maintain a blockchain-like data structure and ensure blockchain properties and guarantees in a mobile environment subject to network partitions?

3.3 The Blockgraph Model

The Blockgraph is a full framework that considers three important aspects of the system; the maintenance of a Distributed Ledger Technology (DLT) taking the form of a Direct Acyclic Graph (DAG), a consensus algorithm capable of tolerating network partitions, and a source of topology information to make the system adaptable to changes in the network topology. The Blockgraph data structure guarantee blockchain's properties such as *immutability*, *integrity*, *transparency*, and *authenticity* of the data. In other

words, the Blockgraph model allows the maintenance of blockchain-like technology in the context of mobile nodes capable of tolerating network partitions.

3.3.1 The System Model and Assumptions

We consider a system where a set of n independent nodes communicates over a mobile ad hoc network. We define a node as an electronic device running the Blockgraph framework, which includes *the Blockgraph protocol*, *the consensus module*, and *the group management module*. In our system model, we assume that nodes have a unique identifier like their MAC address or any other kind of unique identifier. Communication links among nodes are bidirectional, and network topology may change frequently. The Blockgraph network is constituted of a peer-to-peer overlay network over the ad hoc network. Nodes create connections with every joinable node in the network to exchange protocol-related communication. We assume that an authentication protocol is needed to join the network and ensure that only trusted nodes can participate in the network; we can suppose an external distributed authentication protocol is employed to make the system as distributed as possible. Thus, we assume that the Blockgraph network is constituted of well-known trusted nodes. [95, 96] are examples of distributed authentication protocols for mobile ad hoc networks. The network synchrony model is assumed to be a partially-synchronous message passing system with bounded transmission delay between two nodes. The data propagation model might vary depending on the message being propagated over the Blockgraph network, but we can assume that a gossip-fashion information propagation model is primarily employed. We also assume the presence of a solution to discover the network topology to provide *the group management module* with a constantly updated view of the joinable nodes in the network. This network information allows the whole Blockgraph framework to better adapt to the topology changes and manage the DAG-based DLT.

3.3.1.1 The Mobility Model

The Blockgraph mobility model considers that any node participating in the Blockgraph framework is capable to move at will. However, we believe that the top-level applications that may use our Blockgraph solution do not entail a completely random model of mobility. Instead, we assume that a certain order and control exist in the mobility of the nodes, as can often be the case of a platoon of vehicles, team squads, or other group movements that cooperates to achieve a common goal. Indeed, as the mobility factor plays an important role in determining the specifications of the Blockgraph and must particular, *the group management module*, the definition of the model of mobility is a

primary subject. While our Blockgraph solution manages to maintain a DAG-based DLT in a group of nodes, our solution does not manage to achieve its goal in extreme mobility conditions. In this regard, our experiments in simulations and testbed only consider similar mobility conditions such as the speed, direction, and density of the network.

3.3.1.2 The Blockchain Model

It is essential to notice that even though our Blockgraph cannot be considered a blockchain due to the shape of the structure it forms, our Blockgraph inherits properties and characteristics from blockchain technology. In this regard, and intending to keep it simple, we call the blockchain model the ensemble of specifications and attributes of our Blockgraph technology.

Our Blockgraph solution was first conceived as a permissioned blockchain in the context of a specific project. In that sense, the Blockgraph network allows trusted nodes that were previously authenticated to participate in the network. Permissioned blockchains have the advantage to reduce the likelihood of a malicious entity performing an attack since nodes need to be known to join the network. Moreover, by employing access control in the network, we can relieve security mechanisms present in permissionless blockchains that otherwise would not be possible to neglect. Thus, a permissioned blockchain allows us to adopt a more efficient consensus algorithm that provides better performances such as consensus delay, message complexity, or block finality; and since nodes are already well-known by the system and depending on the top-level application, there is no need to employ stochastically proof-based consensus algorithms such as proof-of-work (PoW) to achieve consensus since it would become highly expensive in terms of computational resources and would increase the management complexity of the data structure. Instead, we can achieve consensus by using a deterministic consensus algorithm such as state machine replication systems (e.g., Paxos [91], RAFT [19] or PBFT [35]), or recent blockchain consensus such as proof-of-authority (PoA) [41], which is considered the most recent member of the BFT algorithms.

In every blockchain design, the top-level application plays a fundamental role in determining the properties and characteristics of the blockchain. In our case, we did not contemplate a financial application like most traditional blockchains. Instead, we imagined use case scenarios where sensitive information needed to be stored and replicated in a secure database, like disseminating data through the network for user communication or for the exchange of digital information. In that regard, transactions are added to the distributed ledger while preserving the *immutability*, *integrity*, *transparency*, *auditability*, and *authenticity* of data. Finally, all nodes in the Blockgraph network participate in

the consensus process, disseminate transactions, and can create new blocks. Special nodes are not required to be predefined in the network since every node runs the same Blockgraph daemon.

3.3.1.3 The Transaction Model

A transaction in our model is a digital piece of information relevant to the application using the Blockgraph framework. A transaction could be any kind of information such as a message, a digital file, a date, a time, geographical coordinates, data from sensor networks, traces, or system logs. As opposed to most well-known blockchains, in our transaction model, transactions are not correlated with each other to prevent double-spending frauds in financial applications; a transaction in a Blockgraph is rather represented as a piece of information that deserves to be stored securely. This could be useful to disseminate information in the network or as a means to store secure information the way a computer log does.

The main reason that prevents us from using a transaction model like the one used by Bitcoin is the network partitions. Indeed, if we assume that conflicts among transactions are possible, let's say, by tokenizing assets or using cryptocurrencies; an attacker would only have to connect to another network partition to perform fraudulent transactions (e.g., double-spending attack); and since network partitions are independent of one another, we would only be able to detect those conflicts after a merge of the network partitions. Therefore, the Blockgraph transaction model is *tokenless*.

The Blockgraph ledger, on the other side, is a sequence of transactions that is accessible to everyone having access to the Blockgraph data structure. Therefore, it could be possible to perform one-on-one or one-on-many transactions by using cryptographic functions. Indeed, the use of cryptographic primitives can guarantee pseudo-anonymity and data confidentiality in the Blockgraph model. Every entity transacting in the Blockgraph network can have a set of *public/private keys* to execute a transaction. When an entity wishes to disseminate information accessible to every participant in the network, the transaction can be encrypted with the sender's private key; and receivers will only have to use the sender's public key to decrypt the transaction. Likewise, when an entity wishes to disseminate information accessible only to one other entity in the network, the transaction is encrypted with the destination's public key, being this last one, the only one capable of decrypting the transaction with its private key. Multi-addresses allow making a transaction accessible to a group of entities by using a single public key.

As we can see, the use of cryptographic primitives allows us to ensure data *confidentiality* and *non-repudiation* while maintaining *pseud anonymity*. Indeed, asymmetric key cryptography guarantees that only the holders of the corresponding public keys can decrypt the transaction, which provides *confidentiality*. At the same time, once a transaction is effectively executed, the sender and the receiver of the transaction can no longer deny its doing at a later stage since key pairs are unique and the distributed ledger is immutable, which ensures *non-repudiation*. Finally, only an authorized entity knows the relationship between public keys and the real entities' identity, providing *pseudo-anonymity* during operation.

3.3.2 Blockgraph Structure and Components

The Blockgraph data structure is a new type of DLT similar to a traditional blockchain, with the difference, the Blockgraph data structure does not create a unique chain of blocks; instead, it creates a DAG according to network partitions caused by the mobility of the nodes. In other words, the shape of the DAG is defined by network partitions. Each branch of the DAG is composed of a chain of blocks created by a network partition agreed through consensus by the nodes participating in the consensus process within the network partition.

3.3.2.1 Structure of a Transaction

A transaction in Blockgraph is the smallest and most basic data structure of the Blockgraph framework. It is composed of a header and a data container. The transaction header holds needed metadata information to identify and secure the transaction. Figure 3.1 illustrates a generic transaction. We detail the fields of the transaction header as follows:

- **Transaction Hash:** this is the output of the result of a hash function having as an input the content of the data container (payload). It is further used in constructing a data block to calculi the root hash of the Merkle tree.
- **Size:** the size in bytes of the transaction.
- **Timestamp:** the creation time of a transaction.
- **Sender's public key:** this field allows for the identification of the user responsible for the creation of the transaction.

Alternatively, and according to the top-level application, data fields might be included or excluded in the header. It is also important to notice that a generic

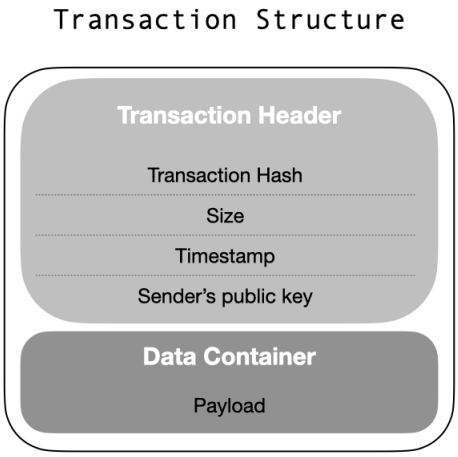


Figure 3.1: Structure of a generic transaction.

transaction model like the one we are proposing here is only the bases for a wider transaction model following the top-level application requirements.

3.3.2.2 Structure of a Block

A block in a Blockgraph is a data structure that includes a header and a data container. Depending on the type of block, the data container might hold a set of transactions or relevant protocol information to perform *the merge synchronization procedure*. In Blockgraph, there are two types of blocks: A *merge block* and a *data block*. A merge block is a special block that is created at the time of merging two or more network partitions. Thus, it does not contain transactions or top-level application information but protocol-related information that facilitates the merging of the network partitions' data structures. A merge block can include in its header the reference of several previous blocks (*HashPrevBlock* field), which are usually the last block of each network partition participating in the merging process. The merge block and *the merge synchronization procedure* are further covered in the Blockgraph framework section 3.4. A data block, on the other hand, can only reference one previous block and can only contain transactions in the data container. Figure 3.2 illustrates the data structure of both blocks; we detail the utility of each field of the block header as follows:

- **Version:** indicates the block's current version. It allows *the Blockgraph protocol* to perform the correct treatment for the block.
- **Block Index:** it refers to the height of the block with respect to the genesis

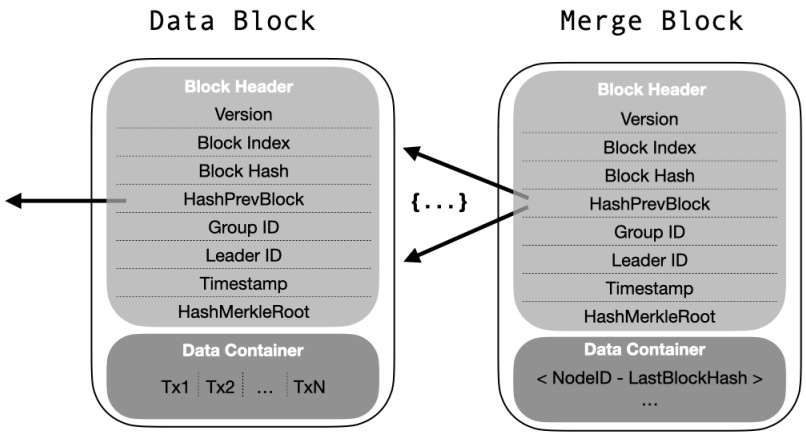


Figure 3.2: Structure of a block.

block. It allows us to find a block and ensure the correct order of the block in the Blockgraph data structure.

- **Block Hash:** it is the outcome of a hash function that takes as input the block's header. It serves as the block identifier and is the value that is included in the *HashPrevBlock* field in the next block's header.

- **HashPrevBlock:** it is the reference to the previous block identifier (*Block Hash*), it allows the interconnection of blocks and gives the chained structure. The field contains an aggregate data type that allows inserting more than one previous block hash in the case of a merge block.

- **Group ID:** it is the outcome of a hash function calculated by *the group management module*, having as an input, the identifiers of all the nodes in the network partition. It allows identifying a group of blocks created by the same network partition (a branch).

- **Leader ID:** it is the identifier of the node that creates the block.

- **Timestamp:** it is the creation time of the block with respect to the miner node's clock.

- **HashMerkleRoot:** it is the hash value resulting from the outcome of the Merkle tree function based on all the transactions in the block. It ensures that potential modifications in the data container entail a modification in the block's header.

3.3.2.3 Blockgraph Data Structure

Like in blockchain, the Blockgraph data structure starts with a genesis block, which is the starting block of the graph and the ancestor of all blocks. Each block is formed of transactions propagated from other nodes. As the system evolves, the Blockgraph adapts to a DAG structure due to splits and merges caused by nodes' mobility. Figure 3.3, illustrate the main elements that compose the Blockgraph data structure. We define each element as follows:

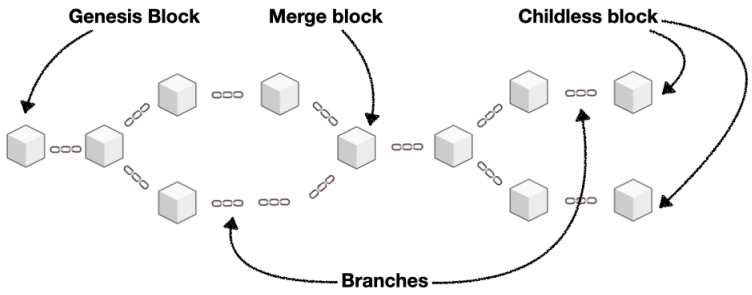


Figure 3.3: Elements of the Blockgraph data structure.

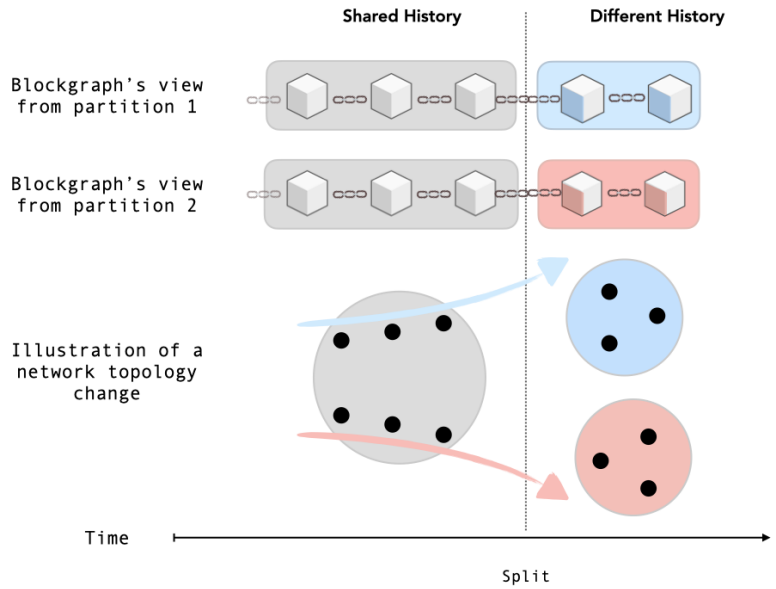
- **Genesis block:** this is the starting block of the DAG-based DLT and the ancestor of all the blocks. It is created when launching the Blockgraph daemon and does not contain transactions. It could serve as a network identifier.
- **Merge block:** this is a special block that is capable of referring to several previous blocks. It is created when merging several network partitions. Merge blocks are the elements of the Blockgraph data structure that allows the fusion of divergent data structure from different network partitions into a single structure.
- **Branches:** they are an ensemble of blocks or a chain of blocks created by a group of nodes in the same network partition. Every block in a branch shares the same *GroupID*.
- **Childless block:** this is a block in the Blockgraph data structure that is not referred to by any other block. In other words, is the last committed block of a given branch in the Blockgraph data structure.

When the network is connected, the shape of the data structure will take the form of a regular blockchain. When a split occurs, the nodes within the same network partition will continue to maintain the Blockgraph data structure by adding new blocks. In this regard, each network partition will create a different block that will have as reference the last common block before the split. At the time of a merge, *the Blockgraph protocol* will merge the different branches held by other network partitions into a single data structure. Indeed, each network partition will synchronize the others' network partitions Blockgraph to conciliate the different branches into the DAG-based DLT. It is important to notice that when a merge is detected, a process of synchronization is triggered at *the Blockgraph protocol* level, which we will fully cover in section 3.4.3. Figure 3.4 illustrates the Blockgraph data structure when subject to a split and a merge. We can notice in Figure 3.4a, that both network partitions share the same history of the DAG-based DLT before the split and become different after the split due to the loss of connectivity. In Figure 3.4b, we can notice that after the merge, all network partitions share the same Blockgraph data structure.

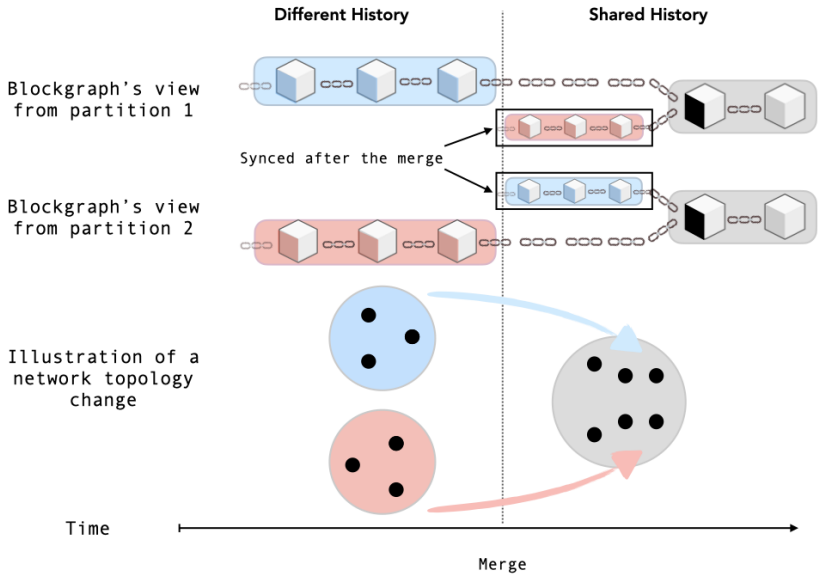
Since the mobility of the nodes plays an important role in the management of the system. Any considerable change in the network topology will trigger the execution of a sequence that will allow the three modules comporting the Blockgraph framework to update their parameters to adapt to the new network topology condition and provide relatively stable conditions to maintain the distributed structure. We define two types of topology changes that trigger updates between modules as follows:

- **Split:** a split occurs when the nodes in the network move such as there is no communication path between two or more nodes due to their transmission range being disrupted by interference or distance. Thus, creating a network partition. Each network partition becomes then disconnected from the other. For our Blockgraph framework, a split in the network topology is detected by every node by their *group management module*, which includes a topology discovery mechanism. The information of the split is then transmitted to *the consensus module* and *the Blockgraph protocol*. At the consensus level, parameters might vary depending on the consensus algorithm being used. For instance, in a vote-based consensus algorithm such as RAFT [19] or PBFT [35], *the majority criteria* are the parameter that defines the minimum number of nodes needed to reach consensus. If the number of nodes changes with the new topology, it is, therefore, a parameter that needs to be changed when a split arises. At *the Blockgraph protocol* level, an updated list of reachable nodes is needed for protocol communication.

- **Merge:** a merge occurs when several independent network partitions become



(a) Split Scenario



(b) Merge Scenario

Figure 3.4: Representation of the effects of a split and a merge in the Blockgraph data structure.

close enough to form a unique connected network. Like in a split, *the Blockgraph protocol* and *the consensus module* parameters are needed to be updated. Moreover, the merge of several network partitions should end in the convergence of the different versions of the Blockgraph involved in the merging. For this purpose, a *merge synchronization procedure* is invoked by *the Blockgraph protocol* to merge all divergent branches. This procedure is further detailed in section 3.4.3.5.

Our Blockgraph, as we have explained, is a distributed system, and as such, is submitted to the CAP theorem [97] proposed by Eric Brewer. The CAP theorem states that any distributed system with data replication cannot guarantee all three desirable properties (*Consistency*, *Availability*, and *Partition tolerance*) at the same time; and therefore, can only strongly support two of such properties. We define such properties as follows:

- **Consistency:** in our context, we argue that data consistency can only be guaranteed at a network partition level. Indeed, when the network is connected, it forms a single network partition in the system, allowing all nodes of the system to agree on the same state of the distributed ledger. When the network is partitioned into several independent network partitions, it is not possible to agree on the same state of the distributed ledger across different network partitions since connectivity between those network partitions is impossible, ergo, the network partitions. Therefore, data consistency only becomes relevant within every network partition. We differentiate two forms of data consistency in the Blockgraph system, strong and weak consistency; strong consistency means that every node in the Blockgraph network has the same state of the distributed ledger. A strong consistency can only be achieved when all the nodes of the Blockgraph network are connected in a single network. A weak consistency means that every node in a network partition has the same copies of the DAG-based DLT within the network partition.

- **Availability:** availability is the capability of the system to respond to clients' requests in a reasonable amount of time without the guarantee that it contains the most recent view of the Blockgraph.

- **Partition tolerance:** this is the capability of the system to continue to operate despite network partitions.

In the case of the Blockgraph context, partition tolerance is a native property of the Blockgraph system since its data structure was designed to adapt to network partitions. Thus, one has to choose between consistency and availability. As for consistency, the Blockgraph system cannot guarantee strong consistency during network partitions since there is no possible way for two or more network partitions without an Internet gateway to maintain the same data due to that each network partition is disconnected from the others. In this regard, it is important to point out that top-level applications should be

consistent with this restriction. Even though strong consistency cannot be achieved during network partitions, weak consistency is a guarantee in each network partition.

3.4 The Blockgraph Framework

The Blockgraph framework is composed of three independent modules that manage three different aspects of our solution. *The Blockgraph protocol* to manage the Blockgraph data structure, *the consensus module* to ensure that all the nodes participating in the Blockgraph network agree on the same state of the distributed ledger, and *the group management module* to provide a stable view of the network topology to the system. We choose to design the architecture of our solution as a modular framework where each module can execute tasks as an independent process. This allows from one side the possibility to easily adapt each aspect of our solution according to the needs and requirements of the top-level applications, and from the other side, to contain potential failures and malfunctions in a module. This type of architecture allows application flexibility and reinforces the security aspect of the system. On the other hand, it is well-known that top-level applications have an important impact on the design and the conception of different aspects of the blockchain, such as the choice of the consensus algorithm or the blockchain transaction model. With our modular framework, changes can easily be implemented to allow other application use cases.

Communication among modules is performed via defined interfaces that we identified for our system. Modules need to communicate selected information from their operation to feed other modules and allow the system to better adapt to changes in the network topology. Figure 3.5 illustrates the architecture of the system along with the three modules that compose the Blockgraph framework with their respective interfaces; we describe the role of each interface of the Blockgraph framework as follows:

- **Network-Io** it feeds *the group management module* with a current view of the network topology. It also provides the module with a list of joinable nodes in the network at regular intervals of time.
- **GMtoConsensus-Io:** it provides *the consensus module* with an updated list of nodes in the network topology at each network partition.
- **GMtoBlockgraph-Io:** : it provides *the Blockgraph protocol* with group membership information, such as an updated list of nodes in the network topology and information on the nature of a network partition (e.g., split and merge).

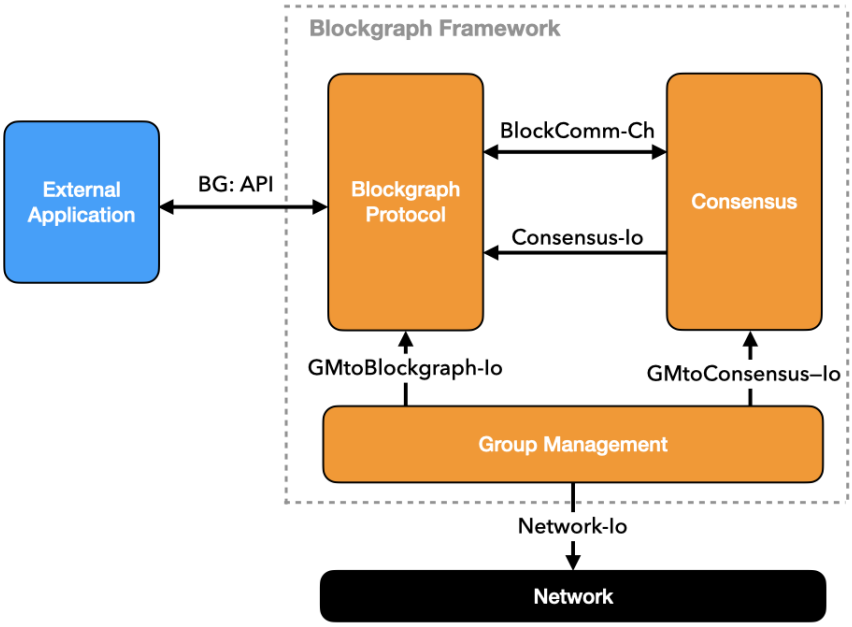


Figure 3.5: Architecture of the system.

- **Consensus-Io:** it allows communication between *the consensus module* and *the Blockgraph protocol*. It particularly provides *the Blockgraph protocol* with information on the current leader node.
- **BlockComm-Ch:** this is the channel where blocks transit between *the consensus module* and *the Blockgraph protocol* for consensus proposes and block treatment.
- **Blockgraph API:** this is the communication channel between *the Blockgraph protocol* and external top-level applications. It allows the top-level application to query the Blockgraph data structure.

3.4.1 The Group Management Module

The group management module is the process of the Blockgraph framework whose main purpose is to constantly be aware of the network topology and to react to drastic changes in the network topology caused by nodes' mobility to adapt *the consensus module* and *the Blockgraph protocol* to a new network configuration. It is also responsible for establishing groups resulting from topology changes. Groups are a list of the reachable nodes in the network partition that provides momentary stability to the Blockgraph framework in the context of dynamic networks. Once a group is established, it remains

so regardless of ephemeral changes in the network topology.

In principle, any solution capable of discovering the network topology could provide the necessary information for our *group management module* to process and generate an understandable output for our framework. To establish a group, each node in the network partition needs to be aware of the presence of other nodes constituting its network partition. To address this problem, several approaches to discovering the network topology might be considered. If we contemplate a network partition as a connected graph, we could conceive a solution that applies a graph traversal algorithm such as Depth-first search (DFS) [98] and Breadth-first search (BFD) [99] to explore the network; and since our framework is distributed, every node will agree on the same group within the same network partition. These types of solutions could be the baseline to adopt a probabilistic consensus algorithm in a dynamic network since we assure that every node in a network partition has the same view of the network topology. Another approach for discovering the network topology is to exploit the information generated from the network layer under the condition of using a proactive routing algorithm. Indeed, a proactive routing algorithm such as OLSR [100] and DSDV [101] maintains a routing table with routes to all destinations within the network partition. We could use this information to obtain a list of reachable nodes to create a group. Nevertheless, regardless of the method employed to discover the network topology, our *group management module* will take as an input a list of joinable nodes in the network topology.

From the list provided by the network discovery solution, our *group management module* will keep a copy of the current network topology that we call, *the current group list*. Such a list will become the comparison element to further changes in the network topology and the input element to calculi the group identifier. The Group Identifier or *GroupID* is the result of a hash function having as an input the list of nodes currently participating in the consensus process, it is the key element that allows *the merge synchronization procedure* to identify the blocks created in the same network partition and therefore belonging to the same branch. The *GroupID* is transmitted to *the Blockgraph protocol* for inclusion in the header of the newly created block. When a new change in the network topology arises, the *GroupID* changes too, and parameters from *the consensus module* and *the Blockgraph protocol* needs to be updated. To identify the nature of the network topology change, meaning whether the change corresponds to a split or a merge, we compare *the current group list* with a *candidate list* containing a new list of joinable nodes provided by the network discovery solution. Since both changes lead to different treatments in the upper modules, differentiating a split from a

merge will allow us to efficiently react to changes in the network topology. We defer a split from a merge by calculating the difference between *the current group list* and *the candidate list*. Indeed, we detect a split by checking that all nodes in the new *candidate list* are also present in *the current group list* and that the size, in terms of the number of nodes in *the candidate list*, is strictly smaller than *the current group list*. Likewise, we detect a merge by checking that all nodes in *the current group list* are also present in *the candidate list* and that the size, in terms of the number of nodes, is strictly greater than *the current group list*. Once the nature of the topology change is identified, it is transmitted along with the *GroupID* to the upper modules to implement their operations. Figure 3.6 illustrates the main elements of *the group management module*.

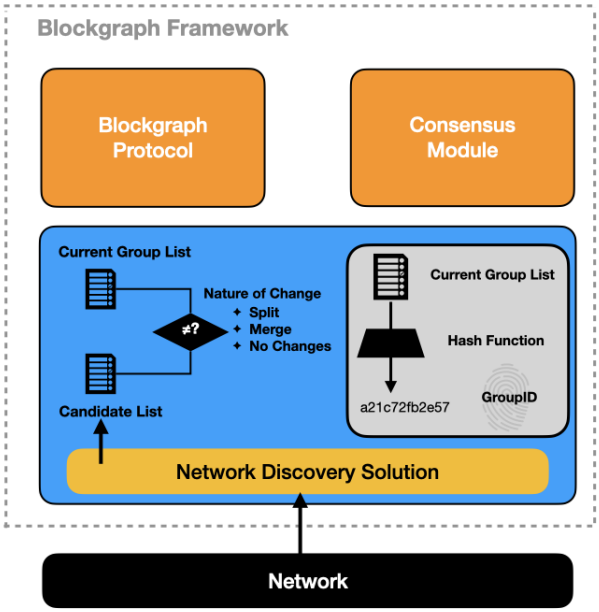


Figure 3.6: Main elements of *the group management module*.

3.4.1.1 Our Network Discovery Solution

For our Blockgraph framework implementations, we chose to make use of the routing tables of OLSR to discover the network topology. This decision was primarily driven by the fact that the mobile routers that we were working on were already using a modified version of OLSR to route packages to their destination. Moreover, as far as we know, there is no precedent in the literature for the exploitation of the routing information to map a view of the current network topology in a distributed system.

The base idea is to extract the routing table of the node and process the information to create a trusted list of joinable nodes in the network topology to create a group.

However, since the network is mobile, some routes tend to appear and disappear with node mobility even without experiencing a network partition. In this case, we would like our algorithm to be capable of reacting only to permanent topology changes and not ephemeral changes. To address this problem, we implement a series of mechanisms that allows *the group management module* to better react to changes in the routing table.

Algorithm A.1 in Appendix A details the mechanisms that we have implemented to fine-tune the level of reactivity to network topology changes. We start by defining two variables, *time_of_change* and *topology_tolerance_time*. The *time_of_change* variable holds the time in which the last effective topology change happened, and the *topology_tolerance_time* is a variable parameter that defines the time that has to elapse without changes for a new routing table before considering a topology change effective. If another change in the routing table appears before the *topology_tolerance_time*, a new *candidate list* is set, no topology changes are triggered, and the process starts again. Once a change is considered effective, and a new list of joinable nodes is selected, the algorithm continues as described above.

It is important to notice that proactive routing algorithms have limitations in converging on the same view of the network topology for every node. This is mainly because the OLSR algorithm selects neighboring multipoint relays (MPRs) nodes to create paths to destinations that only reach a certain number of nodes in the network topology. In a small to medium size topology, this network discovery solution may be used without further problems but loses efficiency for large networks. Finally, in a Blockgraph system implementing a vote-based leader election consensus algorithm, it is the leader node's view that will prevail, and therefore, there is no need for all nodes in the network partition to converge on the same view of the network.

3.4.2 The Consensus Module

The consensus module is the process of the Blockgraph framework that contains a consensus algorithm whose objective is to agree on the same state of the distributed ledger in all nodes participating in the consensus process. It is also responsible for providing the consensus algorithm with updated network topology information and provides *the Blockgraph protocol* with a leader node. Furthermore, it implements a block communication channel where blocks can transit from *the consensus module* to *the Blockgraph protocol* and vice-versa.

Our *consensus module* was designed to contain a consensus algorithm based on leader elections. Indeed, most of the consensus algorithms used in a permissionless

blockchain environment, such as PoW [1], Proof of Stake (PoS) [39], and Proof of Elapsed Time (PoET) [43], assume a stable connected network and implement security mechanisms to tolerate a certain number of dishonest nodes. In the case of our Blockgraph context, we assumed our blockchain model to be permissioned and with a network access control politic. If we take into consideration our assumptions, we realize that using a stochastically proof-based consensus algorithm to reach consensus is not only useless and lowers system performances but could also lead to a waste of resources that are not tolerable within our context. On the other hand, by assuming a trusted permissioned blockchain, we can bypass security mechanisms that are usually intrinsic in stochastically consensus algorithms for permissionless blockchains and adopt a more traditional method such as a legacy state-machine vote-based consensus algorithm to increase the performance of the system. In this regard, our *consensus module* considers a leader election-based consensus algorithm to provide *the Blockgraph protocol* with the leader node capable of creating new blocks.

To respect the modularity of our framework, we have defined a certain number of properties that the consensus algorithm must follow to be compliant with the Blockgraph context. We define those properties as follows:

- **Network partition-tolerant:** the consensus algorithm must contemplate a method that updates membership accordingly to the changes in the network topology; for our Blockgraph system, it allows *the Blockgraph protocol* to only manage blocks agreed from a coherent group of nodes.
- **Crash fault-tolerant:** this property ensures that the consensus algorithm continues to operate in the event of node failure. When a node was in the leader mode during a crash, a new leader must be elected to ensure *continuity*; if the node was a non-leader node, it should be able to reintegrate operation.
- **State termination:** eventually, every honest node decides on the state of the block.
- **State Integrity:** if all honest nodes propose the same state of a block S_b, then any honest node must decide S_b.
- **State agreement:** every honest node must agree on the same state of a block.

Likewise, we define the properties that a leader election algorithm must have to be compliant with our framework.

- **Leader Election Termination:** the leader election algorithm should finish within a finite time.
- **Election Safety:** at most, one leader node can be elected in a network partition.
- **Leader Agreement:** all other non-leader nodes know who the leader is.

The consensus module gets its network topology information from *the group management module*, which provides the module with updated network topology. This information allows the consensus algorithm to update relevant consensus parameters such as *the majority criteria*. In a vote-based consensus algorithm, *the majority criteria* serve as a threshold that determines in the leader election process, the number of needed votes to become a leader; likewise, it determines the minimum number of nodes that a block needs to be replicated and accepted before considered committed. In a dynamic network, where nodes are mobile, the size of the network partition may vary with the mobility; this is the reason why updating this parameter is so important in vote-based consensus algorithms.

Finally, our *consensus module* performs the communication between *the Blockgraph protocol* and *the consensus module*. Indeed, when a new block is created by the leader node at *the Blockgraph protocol* level, it is transmitted to *the consensus module* for agreement. *The consensus module* will then check if the construction of the block is compliant with *the Blockgraph protocol* version, whether the transactions are valid, and if there is no inconsistency at the application level. Once the block is validated by *the consensus module* of the leader node, the block is then propagated to all nodes in the network partition. Other nodes, perform the same verifications and acknowledge the validity of the block to the leader node. Once the block has reached *the majority criteria*, the leader node orders non-leader nodes to commit the block. The commitment process consists in delegating the block to *the Blockgraph protocol* for treatment. If the block does not reach an agreement, the block is dumped, and correct transactions return to the mempool in *the Blockgraph protocol*.

3.4.3 The Blockgraph Protocol

The Blockgraph protocol is the module of our framework that manages the Blockgraph data structure and performs *the merge synchronization procedure*. It gives all the directives for the blockchain characteristics, including the creation and processing of blocks, ensuring the correct order of blocks, and managing the processing of transactions;

it also implements the cryptographic primitives that characterize the blockchain system, such as one-way functions, signatures, and asymmetric keys.

3.4.3.1 Transactions in the Blockgraph Protocol

Transactions are created by external applications that can communicate with *the Blockgraph protocol* via an API. The external application should be capable of generating new transactions following the specifications and characteristics of our system to allow *the Blockgraph protocol* to treat compliant transactions. When a new transaction is created, it is broadcasted to all neighboring nodes in the network partition. Therefore, transaction arrives into *the Blockgraph protocol* in two different ways; either by the Blockgraph API, which allows communication with an external application, or through *the Blockgraph protocol* itself which communicates with other nodes participating in the network partition. In any case, the same treatment is applied to every transaction.

The Transaction Treatment function (Algorithm A.2 in Annex A) details the treatment of a transaction in every node. It starts by checking that the transaction is correctly constructed and that the values in the transaction header are coherent. For instance, it verifies that the time shown in *the timestamp field* is previous to the node's current clock time or that the output of the hash function of the transaction header coincides with the value provided in *the hash field* of the transaction header. It then checks whether the transaction is already present in the node, either in the mempool or into a block in the Blockgraph. If the transaction is not in the mempool nor the Blockgraph, it is stored in the node's mempool until found in a future block or the case that the node is a miner, until included in a block. A transaction is systematically dumped in the case that such transaction is already present in the node or if the mempool has reached its maximum storage capacity.

In the situation of a node receiving a data block, transactions are checked as part of the Block Treatment process; when the block is added to the distributed ledger, the mempool is updated by erasing all the transactions that were present in the block.

In case the node is a miner wanting to create a new block. The transactions are collected from the mempool and added to the block data container. By default, transactions are selected according to the time spent in the mempool; older transactions are selected first.

3.4.3.2 Blocks in the Blockgraph Protocol

Blocks are the main element of the Blockgraph data structure. They are requiring special management due to the mobile conditions in which our solution is contextualized.

Our Blockgraph solution currently supports consensus algorithms based on a leader election protocol. *The Blockgraph protocol* obtains from the consensus algorithm the identity of the leader node; in Blockgraph, the leader node is the only allowed node in a network partition to create new blocks. Blocks are created upon the affirmation of two conditions: The size of transactions in the mempool and the inter-block time must be superior to a certain threshold. Indeed, to obtain an acceptable efficacity ratio between the application data and the protocol overhead, the block must contain a minimum size of transactions to ensure that the application data is larger during blocks transmissions; on the other hand, the inter-block time allows the block to propagate through the network and gives time to the consensus algorithm to agree on the state of the block before a new block arrives. The inter-block time is, therefore, the time that has passed between the creation of two consecutive blocks. Once the block is created, it is delegated to *the consensus module* for agreement and finally transmitted back to *the Blockgraph protocol* for treatment. Figure 3.7 illustrates the different block's communication levels in the Blockgraph framework.

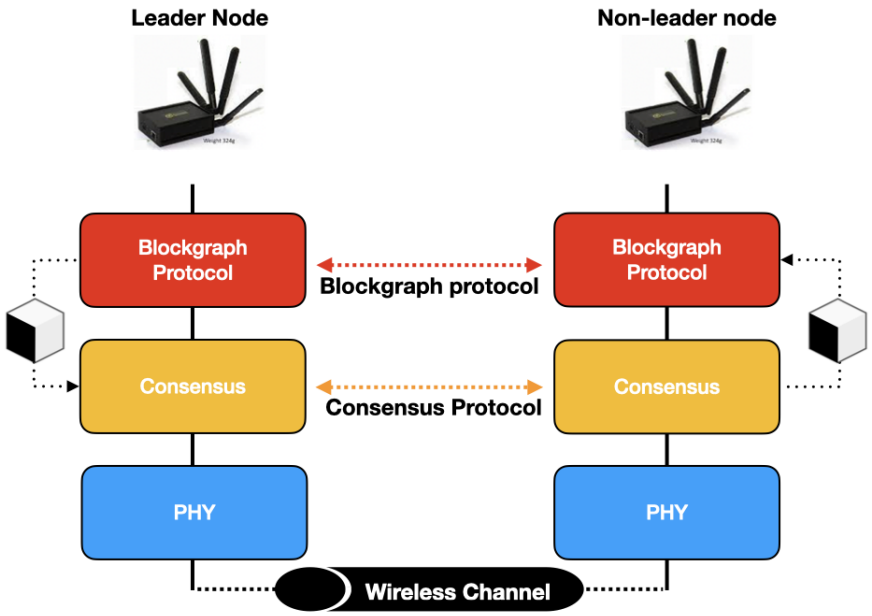


Figure 3.7: Block communication channels in the Blockgraph Framework.

3.4.3.3 Block Mining

Algorithm A.3 in Annex A details the creation of a data block. In the simplest case scenario, where no changes in the network topology are detected nor an ongoing *merge*

synchronization procedure, the leader node ensembles transactions from its mempool into the block data container, obtains the previous block hash and sets the values of the block header. The chosen transactions are removed from the mempool and placed into a temporary memory space until *the consensus module* has managed to replicate the block in a majority of nodes. This temporary copy of the transactions is helpful when a block gets dumped due to the inability to reach consensus within the network partition or due to block or data inconsistencies; therefore, transactions that were in a dumped block can safely be returned to the mempool.

The previous block hash that goes into the block's header that aims to be created is chosen by selecting the blocks in the Blockgraph data structure that are not referred to by any other block in the Blockgraph, such blocks are referred to as *childless blocks*. In stable network conditions, there is usually only one *childless block* per network partition since all nodes in the network partition manage to agree on the same membership configuration. During a merge, it is *the merge synchronization procedure* that deals with multiple *childless blocks* coming from other network partitions to create the merge block. Nevertheless, *the merge synchronization procedure* constantly adds new blocks into the Blockgraph data structure, which can be misinterpreted by the algorithm thinking that the last synchronized block is *childless*, and therefore, a predecessor of a newly mined block. To cope with this problem, at the time of triggering *the merge synchronization procedure*, *the Blockgraph protocol* transits to a synchronization state, where new blocks can refer only to blocks with a higher index than the merge block. *The merge synchronization procedure* is further detailed in section 3.4.3.5.

Once a block is correctly created by the leader node, it is delegated to *the consensus module*. *The consensus module* will take care of broadcasting the block to all neighboring nodes in the network partition to ensure that consensus requirements are meant before treating the block.

3.4.3.4 Block Treatment

The Block Treatment function (Algorithm A.4 in Annex A) of *the Blockgraph protocol* is the main function that manages the Blockgraph data structure. During this process, the protocol decides what to do with the incoming block. It is important to mention that before the treatment of an incoming block, the block has already gone through a period of validations by *the consensus module*. Therefore, every incoming block has already been validated by a group of nodes and is already present in a majority of nodes in a network partition. When *the consensus module* reaches an agreement with a

majority of nodes, the block is automatically transmitted to *the Blockgraph protocol* for treatment. During the treatment, the algorithm examines the header of the block to know if the block is well constituted, if the block is already present in the Blockgraph data structure, and depending on the type of block, sub-processes are invoked. It is also worth mentioning that due to the wireless communication context, blocks might not reach all the nodes in the network partition during the consensus process, therefore, a mechanism to recover missing blocks is also implemented in *the Blockgraph protocol*. Indeed, when a node receives an incoming block, it can realize that it has missing blocks by failing to find its predecessor block announced in the block header. At this point, *the Blockgraph protocol* caches the block in a temporal data structure until recovering the missing block. A sub-process of *the Blockgraph protocol* will then be in charge of recovering all missing blocks.

The process of re-transmitting blocks happens at two different levels. First, at *the consensus module* level, where the leader node re-transmits a block that was not correctly received by other nodes in the network partition until reaching *the majority criteria* of the consensus algorithm; once *the majority criteria* are reached, the block is considered to be sufficiently replicated and is ready for treatment in *the Blockgraph protocol*. Nodes that were not able to obtain the block through the consensus mechanism will try to obtain the missing block through *the Blockgraph protocol*.

3.4.3.5 The Merge Synchronization Procedure

The merge synchronization procedure is a process in the Blockgraph framework that is first triggered by *the group management module* upon the detection of a merge. It invokes a function in *the Blockgraph protocol* that will trigger a series of events that culminate in the fusion of all data structures involved in the merge.

Upon detection of a merge at *the consensus module* level, the current leader node renounces its leadership to invoke new elections. At *the Blockgraph protocol* level, the module waits for the new leader node to start *the merge synchronization procedure*. Once a leader node is elected by the consensus algorithm, it will execute the following steps to perform *the merge synchronization procedure*:

1. **Childless Block Request:** the leader node sends a request to all the nodes that were not in its previous network partition, asking for the hash of their *childless blocks* A.5. Non-leader nodes will only have to look for the last block in their local Blockgraph and send the hash of *the childless blocks* to the leader node A.6.

2. **Creation of a Mapping Table:** upon the reception of the hashes of *the*

childless blocks, the leader node treats the collected information in *the childless block treatment function* detailed in the algorithm A.7. The algorithm primarily searches for the hash of *the childless block* in its Blockgraph data structure and the temporary memory spaces in *the Blockgraph protocol*. Failing to find the hash of *the childless block* sent by another node in its Blockgraph, it will consider that *the childless block* is indeed a *childless block* from another network partition. At the end of the treatment of all the hashes of all *the childless blocks*, the leader node will have generated a mapping table containing a list of tuples <NodeID, childless_block_hash>, where chidless_block_hash is the hash identifying *the childless block*, and the NodeID is the identifier of the node holding *the childless block*, for every new node in the network partition.

3. **Merge Block Creation:** the leader node will create a merge block that will refer to all *the childless blocks* known during the previous phase A.8. Is important to notice that at the time of the creation of the merge block, not every *childless block* is present in the leader's Blockgraph. In fact, only *the childless block* from its previous network partition is present. In the data container of the block, the leader node will include the mapping table instead of regular transactions. Finally, once the merge block is constructed, it is transmitted to *the consensus module* for agreement.

4. **Merge Block Treatment:** once the merge block is committed by a follower node, it is treated by the *Block Treatment* function. The function will identify the block as a merge block and will treat the block in a special way. Transactions, which hold tuples pair <NodeId, childless_block_hash> are extracted from the block and sent for treatment to *the childless block treatment function* A.7. This treatment will allow each follower node to corroborate the leader's node mapping table. At present, each node owns a mapping table that indicates which node has which branch. Nodes will only have to choose among the list, a node that has the branch that they are missing. The merge block is immediately added to the Blockgraph data structure to allow the production of new blocks.

5. **Send Branch Request:** each node will send a *Branch Request* for each missing branch to one node that owns the expected missing branch A.9. The node will choose the node randomly to avoid saturation. If the demand is not satisfied within a certain period, the node can choose another random node to make its request.

6. **Branch Request Treatment:** upon the reception of a branch request, the node

takes the hash of *the childless block* included in the request and searches for a match in its local Blockgraph. When founded, the node will extract from the block's header the information relative to the block's *GroupID*. Let us remember that the *GroupID* is the identifier of all nodes participating in the consensus process in a network partition and that it is included in every block mined by the group. Therefore, the *GroupID* allows identifying all blocks in a Blockgraph's branch. Once the *GroupID* is identified, the node sends all blocks with the same *GroupID* to the requester A.10.

While steps 1, 2, and 3 are processes executed only by the leader node; steps 4, 5, and 6 are executed in a distributed fashion by every other node in the new network partition. We guarantee that new data blocks are not created during steps 1-3 by modifying momentary parameters allowing the creation of a block (block difficulty), naming, the size of transactions in the mempool, and the inter-block time, which are set at higher values to prevent a new block to be created during these stages. Indeed, before *the merge synchronization procedure*, the leader node of each network partition will empty its mempool by creating a last block before renouncing its leadership to give more time to the newly elected leader node to create and commit the merge block. Once the merge block is effectively included in the leader's Blockgraph, the block difficulty parameters are set back to normal values, allowing the production of a new block even during the execution of steps 4, 5, and 6. Resuming a normal production of blocks as fast as possible is a major priority to ensure *continuity* in the system.

There is another challenge in creating new data blocks during *the merge synchronization procedure*. At the moment of creating a new block, *the Blockgraph protocol* will look into the local Blockgraph data structure to obtain the hash of *the childless block* in the Blockgraph to create the link between the new block and the previous block. However, during *the merge synchronization procedure*, blocks from other network partitions are being added as well to the local Blockgraph, causing the momentary presence of more than one *childless block* as illustrated in Figure 3.8. In this case, *the Blockgraph protocol* must be able to only include the hash of the last created block after the merge into the new block header and not the reference of an ancient block from another branch of the Blockgraph. To avoid this problem during the synchronization phase, new blocks can only refer to blocks created after the last merge block.

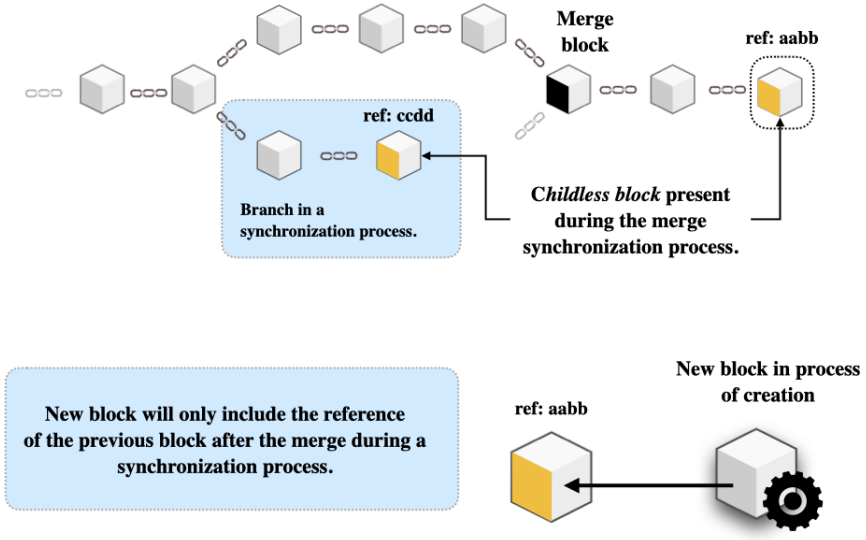


Figure 3.8: Problematic of multiple childless blocks during *the merge synchronization procedure* and the creation of new blocks.

3.5 Blockgraph Implementations and Evaluations

To validate the concept of Blockgraph, we have implemented our solutions in a two-phase approach. First, we have implemented the Blockgraph framework into the discrete-event network simulator for Internet systems, NS-3. Then, we implemented our solution into a testbed composed of five low-power mesh routers from Green Communications as proof-of-concept.

3.5.1 NS-3 Implementation

To implement the Blockgraph framework, we use legacy NS-3 libraries to simulate the lower layers of the protocolar communication model and create new NS-3 code to simulate the application layer. To create the Blockgraph data structure we implement the classes *transaction*, *block*, and *Blockgraph*. Each class contains the needed functions for the creation, manipulation, and destruction of the corresponding instantiated object. We create a special application class for *the Blockgraph protocol*, which contains all the functions related to the management of the Blockgraph data structure. A customized *ApplicationPacket* class was also created, which handles all protocol-related messages exchange. For *the consensus module*, we create *the oracle class*, which is able of simulating a consensus algorithm by adjusting its parameters in function of the required

performances. Finally, we created *the b4mesh-mobility class* to define a customized mobility model adapted to our needs and implement our *group management module* in charge of discovering the network topology.

To simulate the mobile communication network, we used the NS-3 *YansWi-FiPhyHelper* to define the physical conditions for the nodes' communication. We implemented an IEEE 802.11g Wi-Fi standard configured in the ad hoc mode for all nodes. The radio frequency locates in the range of 2.4 GHz with a constant data rate of 11 Mbps. The propagation loss model used for our simulations is the NS-3 *RangePropagationLossModel* with a defined maximum range (MaxRange) of 100 meters. This propagation loss model allows us to control the maximum distance between a transmitter and a receiver to communicate properly. Any receiver beyond the *MaxRange* receives a transmission at a -1000 dBm, which is effectively zero. This configuration allows us to have better control of the splits and merges of the network when defining our mobility model. Additionally, we have installed a full Internet protocol stack that aggregates IP/TCP/UDP functionalities and chose OLSR as the routing protocol for all nodes.

For our mobility model, we have created a customized class application called *b4mesh-mobility* that constantly updates the position of the nodes through time. Each node has installed the NS-3 *ConstantPositionMobilityModel* that allows us to set for any node a new position and velocity at any given time. Thus, our customized application is tasked with updating the positions of the nodes accordingly to a predefined mobility scenario that we defined.

To implement our *group management module*, we made use of the *RoutingTableChange* trace source of the *OLSRRoutingProtocol* module of the NS-3 library that allows us to know and trigger a defined function when a changeset in the OLSR routing table has occurred. Thus, we have implemented our network discovery solution described in Section 3.4.1.1 to discover the network topology.

3.5.1.1 Methodology, Simulations, and Results

The first objective of this implementation was to validate the concept of Blockgraph and obtain the first performance of our system through simulations. To that, we measure the level of replication of the distributed ledger and the impact a network partition can have on the system. We kept track of the number of transactions in the leader's mempool to monitor the creation of new blocks and evidence the impact of the network partitions on the block creation process.

We identified two case scenarios that allow us to compare and test the performance of our solution. In the first scenario, all nodes in the system move in the same direction maintaining a connected network throughout the simulation. Therefore, no network

partitions are inflicted in this scenario. The second scenario involves a partition (split) and a regrouping (merge) of the network to validate the concept of Blockgraph. Table 3.1 and 3.2 summarize the description of nodes' mobility according to the simulation time for each scenario respectively.

We ran each scenario 10 times. Thus, the results correspond to the average of 10 experiments. Each simulation corresponds to a relative duration of 15 min (900 s). The number of nodes for both scenarios is 10 nodes.

Simulation time	**Description of mobility**
$< 900s$	All nodes move towards the same direction in a single connected network.

Table 3.1: Mobility description for Scenario 1

Simulation time	**Description of mobility**
$< 300s$	All nodes move towards the same direction in a single connected network.
$300s - 600s$	A partition in the network occurs. Nodes 0 to 4 and 5 to 9 form an independent network partition respectively.
$> 600s$	All nodes regroup themselves into one connected network. They all keep moving in the same direction as a single connected network.

Table 3.2: Mobility description for Scenario 2

The creation of a block occurs when the leader's mempool reaches 130 transactions. The size of each transaction is generated by a random variable that follows a uniform law that ranges between 300 and 600 bytes. On average, each block has a size of 58.5 KB, which does not exceed the theoretical limit of a UDP packet which is defined at 65.6 KB. Each node generates transactions that are propagated to all nodes in the same network partition. The generation of transactions in each node is modeled as follows: each node generates a random variable that follows an exponential law that represents the time between two generations of transactions (time between two arrivals). We can then model the arrival of transactions as a Poisson process, where lambda (λ) represents the transaction arrival rate per node and is defined as the inverse of the time between two arrivals. For our simulations, we choose to set a relatively low λ to ensure that we do not saturate the system to provide the right conditions to test our concept. Thus, each scenario uses a transaction arrival rate of $\lambda = 0.5$ transactions per second (tps). In other words, a node generates a transaction every 2 seconds.

Figure 3.9 shows the number of unconfirmed transactions in the leader's mempool. Every time the mempool reaches 130 transactions, a block is created and transactions are removed from the mempool. The curve shows the expected behavior; with a $\lambda = 0.5$ *tps* and a network composed of 10 nodes, we have a global transaction rate of 5 tps. We should be able to form a block every 26 seconds and have a total of 34 blocks at the end of the simulation. Table 3.3 summarizes relevant results from the simulation. We can appreciate that both the total number of blocks created during the simulation and the number of transactions generated correspond to the theoretical results. The block replication percentage, which is the percentage of nodes that were able to replicate the full ledger, is 89.34%, which is a large majority of the nodes.

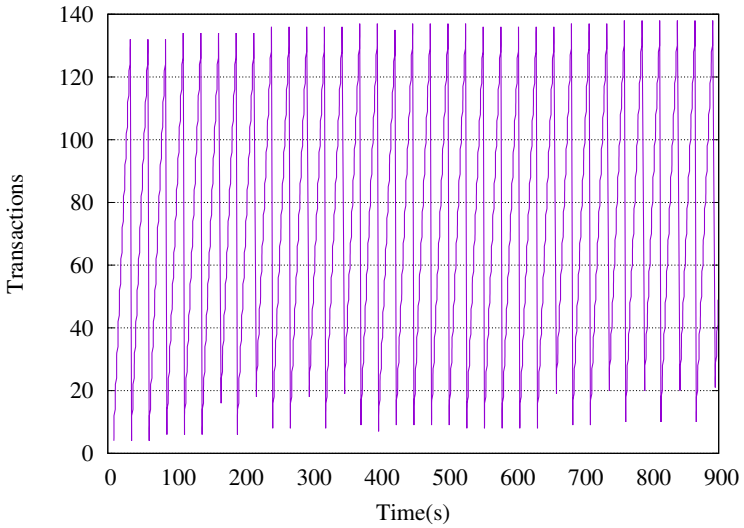


Figure 3.9: Progression of the number of transactions in the leader's mempool through time for the *No Split* scenario with $\lambda = 0.5$ tps.

Number of blocks	35
Number of transactions in the Blockgraph	4030
Block replication percentage	89.34%

Table 3.3: Scenario 1 summary

In scenario 2, we can appreciate in Figure 3.10 that there is a deceleration in the production of blocks between 300 and 600 seconds. This corresponds to the moment there was a split and a merge of the network partitions respectively. This deceleration can be explained by the fact that since the network has been partitioned into two separate network partitions, each network partition is now composed of 5 nodes. With this network configuration, it takes around 52 seconds for every network partition to produce a block. Hence, during the partition period, each network partition could only

have produced 5.77 blocks approximately. Table 3.4 summarizes some relevant results from this simulation. We can appreciate that the results in this scenario correspond also to the theoretical results. The percentage of block replication is close to 99.72%. The difference between the results of the two scenarios might be caused to a decrease in the load on the network in scenario 2 during the partition period. By reducing the number of nodes, the exchange of messages needed to reach consensus is also reduced. Moreover, the arrival transaction rate is divided by two during this time period. The sum of these factors may be the cause of the difference in the two scenarios' performances.

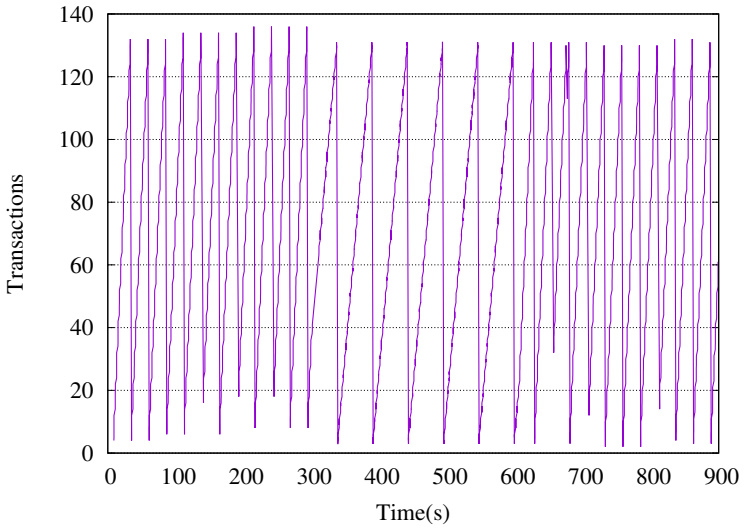


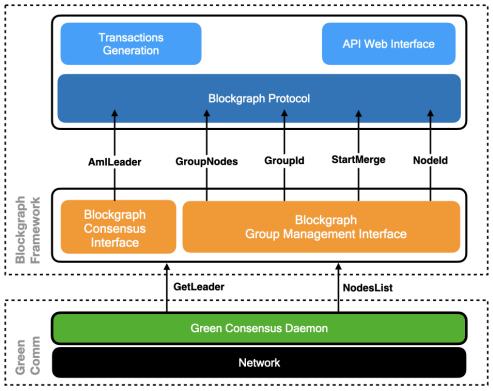
Figure 3.10: Progression of the number of transactions in the leader's mempool through time for the *Split followed by a merge* scenario with $\lambda = 0.5$ tps.

Number of blocks	36
Number of transactions in the Blockgraph	4680
Block replication percentage	99.72%

Table 3.4: Scenario 2 summary

3.5.1.2 NS-3 Implementation Conclusion

The implementation of our Blockgraph solution in NS-3 has allowed us to validate our concept through simulations. The results show that in a 10 nodes network the DAG-based distributed ledger is replicated in both scenarios in more than 89% of the network. Moreover, we validate the efficiency of our *merge synchronization procedure* and provide the first results on the block creation process. More experiments using simulation shall be done to study other Blockgraph performances.



(a) Final architecture of the Blockgraph framework with Green's communication core system

(b) Illustration of a mesh router node with its network interfaces.

Figure 3.11: Architecture of the Blockgraph system in integration with Green Communication's mesh routers.

3.5.2 Testbed Implementation

After having validated the concept of Blockgraph through simulation, we opted to implement our solution in a testbed environment. Our testbed is composed of 5 low-power mesh routers from Green Communications. The main objective of this implementation is two-fold: (i) create a proof-of-concept of our solution in real equipment and (ii) evaluate the performance of our system on a real platform.

The implementation of the testbed was realized in collaboration with multiple parties and was primarily inspired by the NS-3 implementation. In this regard, we choose to implement our solution in *C++* language programming, which is the same language used for the NS-3 implementation. To integrate our solution with the Green Communications Core System, we created *the b4mesh application*, which aggregates the Blockgraph framework and made it compatible with existing functionalities. Figure 3.11a illustrates the final architecture of our system in integration with Green's Core System and Figure 3.11b illustrates the mesh router used for our testbed.

We used the preexisting consensus algorithm and the integrated network discovery topology in Green's Core System to feed our Blockgraph's *consensus module* with a leader node and a list of reachable nodes participating in the consensus process. It is important to notice that these functionalities were already present in the mesh routers since it implements a distributed system that requires consensus for its functioning. Thus, *the group management module* and *the consensus module* were adapted in this implementation to work as a proxy between Green's Core System and *the Blockgraph protocol*.

Our implementation is constituted of 8 classes that regroup the required elements to implement the Blockgraph framework. Figure 3.12 illustrates a class diagram of

our implementation. *The session class* manages the connection of the application among the distributed nodes, *the client class* handles the Blockgraph's endpoints for clients utility, and *the consensus and node classes*, implement *the consensus module* and *the group management module* proxies respectively. Additionally, we meet again *the transaction*, *block*, and *Blockgraph classes* that form the Blockgraph data structure, the *ApplicationPacket class* that handles the Blockgraph's protocol messages, and *the b4mesh class*, which implements *the Blockgraph protocol*.

The mesh routers are equipped with three network interfaces: an Ethernet interface and two Wi-Fi interfaces. The Ethernet interface is meant to provide a gateway to another reachable network, while the first Wi-Fi interface is used to provide access to clients and the second Wi-Fi interface is used to create a backhaul network that connects every other router node together.

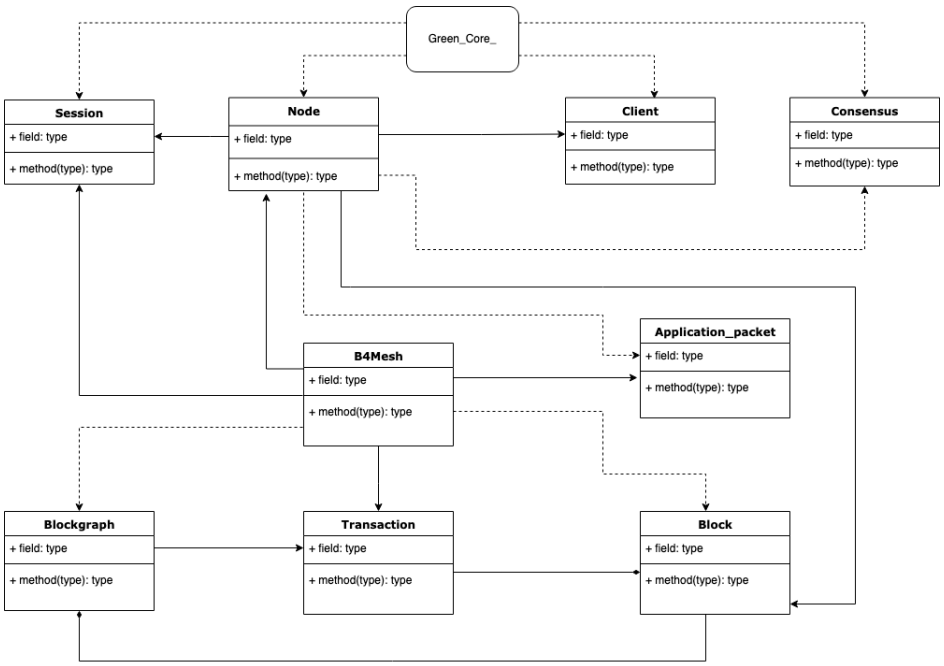


Figure 3.12: Class diagram of the testbed implementation.

3.5.2.1 Methodology, Simulations, and Results

The implementation of the Blockgraph in a testbed has as objective to demonstrate the proof-of-concept of our solution and provide the first performance of our system in a real mobile network. During our experimentations, we took special care to measure the transaction throughput of the system, the resource usage of our solution in the mesh routers, the transaction latency, and the mempool usage. To this, we retook our two scenarios where we contrast the impact of maintaining a distributed ledger in a fully connected network with a mobile network subject to a network partition.

The experiments in both scenarios were held at our facilities at the Computer Science Laboratory of Paris (LIP6). Every experiment made has an average duration of 500 seconds. Thus, we define our two scenarios as follows.

- **Static Network:** in this scenario, the five mesh routers are fully interconnected and are not subjected to mobility. Routers are equidistantly spaced.
- **Split and Merge:** in this scenario, the five mesh routers start as a fully interconnected network. After 120 seconds from the beginning of the experiment, we create a network partition by taking two mesh routers far away in the corridor until the network is completely split into two clusters. One network partition is composed of three mesh routers and the other with two routers. Both network partitions are separated long enough so new blocks are generated in each network partition. At 240 seconds from the begging of the experiment, we create a merge of the two network partitions by bringing the two mobile routers back to their initial position.

For the measurement of the performance of our proof-of-concept, we executed each scenario 10 times. Results are, therefore, the mean of the results of executions. For each scenario, we vary the rate at which transactions are generated. Every node generates transactions at the same constant rate. The average size of a transaction is 450 bytes and it is defined by a random variable following a uniform law. The generation of transactions is modeled as a Poisson process of parameter λ following an exponential law independent from the other mesh routers. Thus, for our experimentation, we choose a transaction generation rate per node (λ) varying between 0.5 and 3 tps. Table 3.5 summarizes the transaction generation rate values and the main fixed parameters of our experiments.

Transaction generation rate values

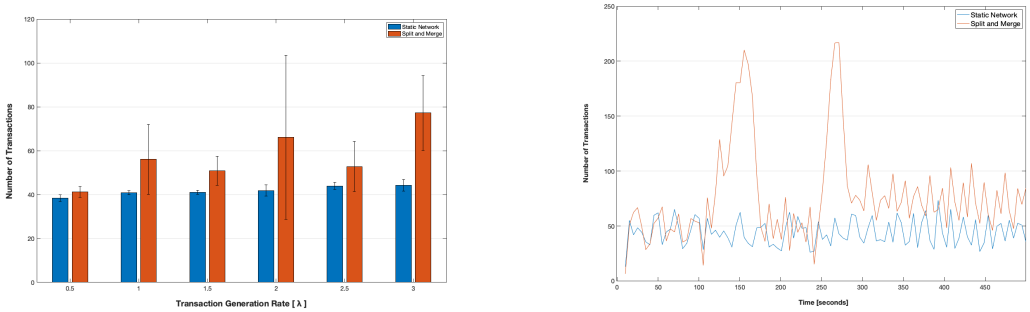
Per node in tps	*Global in tps*
0.5	2.5
1	5
1.5	7.5
2	10
2.5	12.5
3	15

Execution parameters

Parameter	*Values*
Number of nodes	5
Execution time	500 sec
Mempool Size	500 KB
Avg. Transaction Size	450 bytes
Min. Inter-Block Time	5 sec

Table 3.5: Variable and fixed parameters of the executions.

Figure 3.13a compares the average number of transactions in a router's mempool for different transaction generation rates for both scenarios. This performance allows



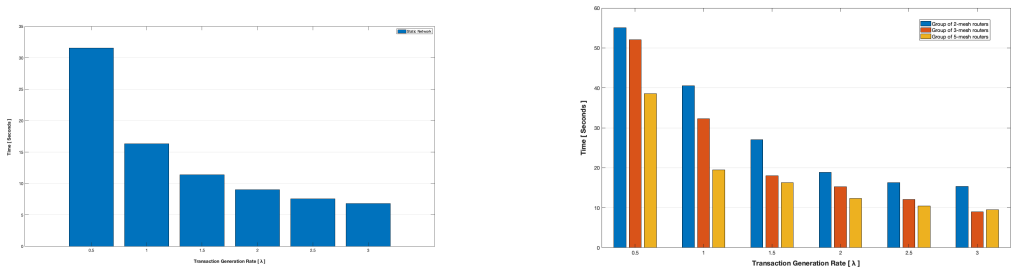
(a) Number of transactions in the Mempool.

(b) Evolution of the number of transactions in the Mempool.

Figure 3.13: Average Mempool usage.

us to monitor the node's mempool utilization space for all λ values. We can notice that in the static network scenario, the difference between the number of transactions in the router's mempool is very short among the different values of λ. On average a mesh router has 38.4 transactions in its mempool when $\lambda = 0.5$ *tps*, while for $\lambda = 3$ *tps* it only increases to 44.2 transactions. That is a difference of 5.8 transactions for a 3x transaction income rate. This behavior indicates that the protocol manages to correctly adjust the rate at which blocks are created. On the other hand, in the split and merge scenario, we can notice an upward trend in the average number of transactions in a node's mempool. This is partially explained by *the merge synchronization procedure*, which maintains transactions in a node's mempool until recovering the missing blocks. However, we can notice in Figure 3.13b, which illustrates the average evolution of a node's mempool through time for a global transaction generation rate of 15 tps for both scenarios, that the split and merge scenario presents two spikes at the split and merge respectively. Thus, concluding that the topology change has an impact on the mesh router's mempool but recovers its normal behavior relatively fast.

Figure 3.14 show the evolution of the average inter-block time for every λ in each scenario respectively. The inter-block time refers to the time that has passed between the creation of two consecutive blocks and models the cadence at which blocks are created. We can notice that as λ grows in both scenarios, the inter-block time decreases forming an exponential decay curve. This is the expected behavior since as transactions arrive at a higher rate, conditions for generating a new block are satisfied faster. In Figure 3.14a, when $\lambda = 3$ *tps*, we reach an inter-block time of 6.8 seconds which is still relatively far from the 5 seconds we defined in Table 3.5 to allow a block to propagate in the network before a new block arrives. Thus, there is still room for an increase in the transaction throughput. Figure 3.14b shows the same curve behavior for the scenario



(a) Scenario Static Network.

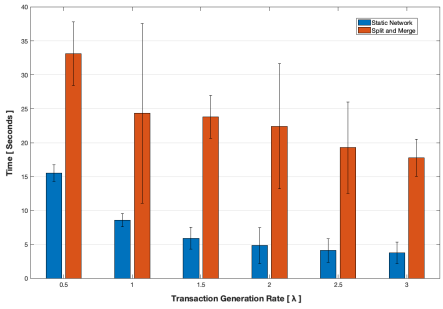
(b) Scenario Split and Merge.

Figure 3.14: Average Inter-Block Time.

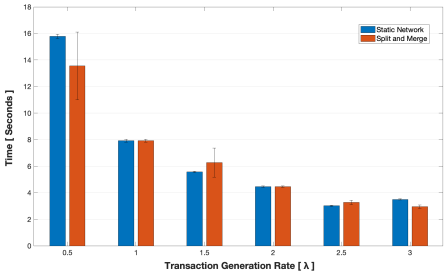
split and merge. However, due to topology changes, notably the split, the inter-block time is higher since it takes more time for network partitions to satisfy the conditions to generate a new block.

Figure 3.15 illustrates the average time for a transaction to be included in a block with respect to λ for both scenarios. To that time, we call it transaction latency. It is measured from the moment the transaction is created to the moment the transaction is included in a block. Figure 3.15a illustrates the average transaction latency for all transactions in the Blockgraph. We can notice in the split and merge scenario, that transaction latency is between 2.2 and 4.8 times slower than in the static network scenario. This difference is due to *the merge synchronization procedure*. Indeed, our calculations include transactions created in a network partition and added to another network partition when recovering the blocks. Nevertheless, the values in the split and merge scenario might correspond to the average time for a transaction to be fully replicated in the distributed system for this specific case scenario. Figure 3.15b, on the other hand, only considers the transactions created in each network partition without considering *the merge synchronization procedure*. The results show that the transaction latency has on average the same performance for both scenarios. Thus, the mobility of the nodes has little impact on the transaction latency performance and goes as low as 3 seconds for $\lambda = 3$ *tps*.

Figure 3.16 shows the evolution in time of the transaction throughput for each λ. The transaction throughput is the number of transactions per second being added to the Blockgraph data structure. We can notice in Figure 3.16a, which corresponds to the static network scenario, that our system remains in a transient state during the 150 seconds of the execution. During this state, we can appreciate that the number of transactions per second is increasing with time until reaching a practical maximum. For the rest of the execution, the system reaches a steady state near the theoretical

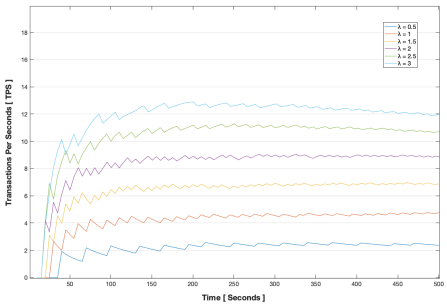


(a) Transaction Latency considering the merge synchronization procedure.

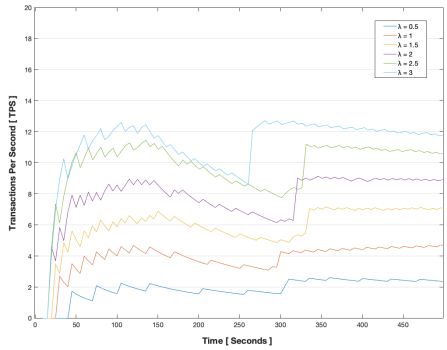


(b) Transaction Latency without considering the merge synchronization procedure.

Figure 3.15: Transaction Latency.



(a) Scenario Static Network.



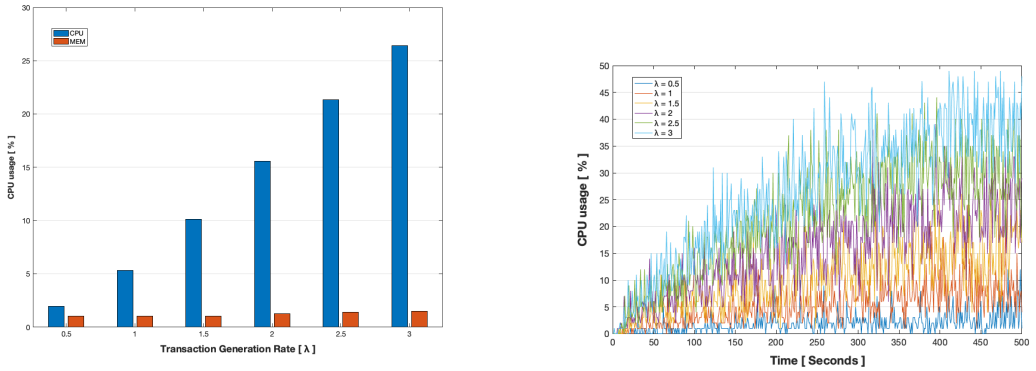
(b) Scenario Split and Merge.

Figure 3.16: Transaction Throughput.

maximum of transactions per second, which corresponds to the global transaction generation rate defined in Table 3.5. On the split and merge scenario illustrated in Figure 3.16b, the curve shows a more dynamic evolution of the performance. At 150 seconds, as the network is being partitioned, a decrease in terms of the number of transactions is observed. This is the expected behavior since the fewer nodes are in a network partition the longer it takes for a network partition to create new blocks, which impacts the transaction throughput. Around 300 seconds, the two network partitions are merged again and a marked rise of the transactions per second being added to the data structure is witnessed. This is caused by *the Blockgraph protocol* merging both data structures to create a single branch. Finally, at 350 seconds a steady state is reached again.

Figure 3.17a illustrates the average CPU and Random-Access Memory utilization for each λ value. We can notice that the CPU usage of the node increases with the transaction generation rate in a linear form, reaching a maximum of 27% of the CPU usage for $\lambda = 3$ *tps*. It is important to notice that the Blockgraph daemon is not

the only service running in the mesh router. Other services such as Nginx and native services of the system are also using the node's resources. Thus, we can not attribute the full amplitude of the given percentages to our Blockgraph resource utilization. Memory usage, on the other hand, increases very little for different values of λ. We can therefore conclude that the Blockgraph framework has very little impact on the Random-Access Memory of the mesh node. Figure 3.17b shows the evolution of CPU usage through time for all λ values. We can observe a transit state for each λ value. As time passes, more aggressive variations in the usage of the CPU are witnessed for higher λ values. Nevertheless, we can observe a clear stabilization of the CPU usage as time reaches the end of the execution. Thus, even though there is a clear correlation between the transaction generation rate and the CPU usage, our testbed manages to use less than 50% of the CPU resources for all executions.



(a) Resource usage.

(b) CPU usage through time.

Figure 3.17: Average Resource Usage.

Finally, Figure 3.18 illustrates an example of the Blockgraph data structure of our testbed implementation. Figure 3.18a shows the final Blockgraph of one of the mesh routers for the split and merge scenario with $\lambda = 0.5$ *tps*. Figure 3.18b shows the inclusion of every block through time. We can notice the effect of *the merge synchronization procedure* by remarking that the merge block (3505486) is added into the mesh router's Blockgraph before blocks 15297894 and 9918612, which are blocks created by the other network partition in which the corresponding mesh router did not participate. It is important to mention that our Blockgraph prototype managed to replicate the 100% of the distributed ledger for all executions.

(a) Final Blockgraph data structure. (b) Addition of Blocks through Time.

Figure 3.18: Example of the final Blockgraph data structure after an execution.

3.5.2.2 Testbed Implementation Conclusion

We implemented a prototype of Blockgraph using mesh routers as a proof-of-concept of our solution. The results demonstrate from one side, the possibility of maintaining a blockchain-like data structure in MANETs in the presence of topology changes leading to network partitions. On the other side, we provide the first performance evaluation of our solution that allow us to characterize our Blockgraph. Thanks to this results, we can better dimension the required space allocated to the usage of a node's mempool in function of the transaction generation rate and the number of nodes in the network. For our prototype, we only used less than 10% (45 KB) of the allocated memory space (500 KB). We demonstrate the possibility of increasing the transaction generation rate without crating forks in the Blockgraph data structure by calculating the average inter-block time. We also demonstrate that the transaction latency remains constant regardless of the mobility and that solution reached a maximum throughput of 13 tps, which is only 2 tps less than the theoretical maximum. Moreover, we have implemented a live visualizer that allow us to see in real time the evolution of our Blockgraph and provide statistics of the Blockgraph. In the next steps, the testbed will be used to measure main performance metrics such as block and transaction processing time. These measures will be then integrated into our simulator to evaluate the performance of Blockgraph in more complex mobility scenarios and large scales.

3.6 Chapter Summary

In this chapter, we have broken down our Blockgraph solution, a full framework capable of maintaining a blockchain-like DAG-based DLT data structure that is capable to deal with the mobility of the nodes. We have started by introducing the context in which our Blockgraph solution is susceptible to function. We explained the split and merge problem, which is the fundamental difference and the main reason why a traditional blockchain solution cannot deal with frequent network partitions. We then proceed to explain our system model and the assumptions we made to develop our solution. We noticed that our Blockgraph solution was not meant to deal with random and chaotic mobility since it responds to specific application needs such as platooning, squad commuting, and team-oriented mobility. We highlighted the flexibility of our modular

architecture that allows us to adapt functionalities of our Blockgraph framework to better fit with the top-level application. This includes the transaction model, the network discovery solution, and the consensus algorithm. We have also detailed the structure of the main elements that composes the Blockgraph data structure such as the transactions, data blocks, and merge blocks. We have provided a detailed explanation of our *merge synchronization procedure*, which is the method we use to merge divergent branches to form a unique data structure after a merge. Likewise, we have explained the solution we have adopted to perform the discovery of the network.

In the next chapter, we present a consensus algorithm capable to tolerate network partitions that we have called C4M (Consensus for Mesh Networks). This consensus was developed in the context of our Blockgraph solution and was meant to function as the *de facto* consensus algorithm working in *the consensus module* of our Blockgraph framework.

Chapter 4

Consensus for Mesh Networks

This chapter presents our consensus algorithm solution for our Blockgraph: A partition-tolerant algorithm for mesh and mobile ad hoc networks. Consensus for Mesh (C4M) implements a novel solution to constantly renew its configuration (nodes participating in the consensus process) and synchronize unphased entries from different network partitions. We also present an overview of the RAFT consensus algorithm, which is the base of our work. Moreover, we provide details of our implementation in the discrete network simulator, NS-3, and present a performance evaluation of our solution. Finally, we present our conclusions and final thoughts.

4.1 Introduction

Reaching consensus has always been at the center of any social concern. It allows us to agree on practical issues and make collective decisions that go forward in the development of society. Nevertheless, we must follow established frames to make those decisions in a regulated and ordered manner; for instance, in Occident, most societies have opted for a parliamentary system in which representatives from the civil society gather to discuss and agree on a certain number of measures and resolutions.

In the context of distributed systems, the problem of reaching consensus is approached differently. A distributed system is composed of computer nodes that strive to achieve a common goal despite being separated geographically; the common goal would be to reach an agreement on certain data values. To that, a value or set of values needs to be proposed for consensus consideration by a node.

There are two main consensus models that worth to be considering at the time of designing a distributed system: a deterministic or a stochastic model. In a deterministic model, the set of events needed to reach consensus is completely determined by previously existing causes, for instance, the appointment of a leader node proposing new data

values for agreement. Paxos [91], Practical Byzantine Fault Tolerance (PBFT) [35], and RAFT [19] are representative vote-based consensus algorithms employing a deterministic approach since they use a voting procedure to elect a leader among a set of nodes, which is the only node allowed to propose data values. On the other hand, in a stochastic model, the set of events needed to reach consensus is described by a random probability distribution, where every node participating in the consensus process has a certain probability P to propose new data values. Proof-of-Work (PoW) [1], Proof-of-Stake (PoS) [39], and Proof-of-Elapsed-Time (PoET) [43] are representative consensus algorithms using pseudo-randomization to select a leader. Naturally, the adoption of each model will mainly depend on the nature of the distributed system itself, namely, a permissioned or permissionless system. Indeed, when considering a permissioned distributed system, we are implying the presence of a centralized entity with a certain degree of control over the distributed system. In this case, a deterministic consensus model might be adequate since nodes are usually known and trusted by the central entity and provides strong control over the distributed system; in addition to better performance. On the other hand, when considering a permissionless distributed system, we are implying that there is no control over the participants of the network, and therefore, the network is trustless. In this case scenario, a stochastic consensus model is preferred since providing the opportunity to any node to propose a value or a set of values for agreement is preferable to having a selected group of non-trusted nodes dictating new values.

It is also important to keep in perspective that the performance of the consensus algorithm will depend on the type of model chosen for the application. While a deterministic model will provide high transaction throughput, most of these solutions usually suffer from scalability issues since they tend to be intensive in terms of message transmissions. In opposition, a stochastic model has better scalability and reduces the number of messages used by the protocol to achieve consensus but at the cost of lower transaction throughput and the possibility of introducing collisions in the leader selection process. Therefore, choosing a consensus algorithm is one of the most critical decisions in the designing processes of a distributed system.

In chapter 3, we have detailed our Blockgraph solution and the context in which it unfolds. We can think of our solution as a permissioned distributed system composed of a set of mobile nodes that maintains the Blockgraph data structure. As in blockchain, our Blockgraph solution needs a consensus algorithm to reach an agreement on the state of the distributed ledger. Nevertheless, and very different from traditional blockchains, the nodes participating in the consensus process are mobile; thus, partitions in the

network topology are likely to happen. Therefore, the consensus algorithm for our Blockgraph solution should be *partition-tolerant*.

In general, most of the consensus algorithms designed for blockchain consider a stable and fully connected network that can tolerate faulty or malicious nodes. Hence, these protocols can deal with topology changes caused by node crashes, node arrival or departure, and node misbehavior. However, in the context of mesh and mobile ad hoc networks, most network topology changes occur due to nodes' mobility. In this context, the network might split into different partitions or merge into a single one. Thus, the consensus algorithm should not consider these topology changes as faulty events but as network partition events (split and merge), while continuing to consider faulty or byzantine nodes. In this regard, the consensus algorithm should adapt to network topology changes as they are happening, allowing nodes in each network partition to continue with the consensus process.

Next, we present Consensus for Mesh (C4M): A partition-tolerant algorithm for mesh and mobile ad hoc networks based on RAFT [19]. It was designed as part of the Blockgraph framework, nonetheless, it might find utility in general mobile distributed systems. We opted for a deterministic consensus algorithm due to the permissioned nature of the Blockgraph system. And we have chosen RAFT as our based algorithm due to its simpleness, task division (leader election, log replication, and safety), and profile needed features for our solution. For this purpose, we have mainly modified the "membership change" procedure of RAFT to enable the algorithm to handle frequent changes in the network topology and added a realignment indexes procedure when merging different network partitions. Our modifications remain compatible with other events such as node crash, node insertion, and node removal that RAFT was designed to handle.

4.2 RAFT Overview

RAFT is the result of the work of Diego Ongaro and John Ousterhout (Standford University) in their quest for better understandability of how consensus can be achieved in a distributed system. Priory to RAFT, the Paxos algorithm [91] was the primary reference on how to achieve consensus in distributed systems; however, Paxos is very difficult to understand and implement, making it unpractical and less attractive for building new systems. RAFT produces similar results and performance as Paxos. It enhances understandability by separating key elements of the consensus process, such as *leader election*, *log replication*, and *safety*. Moreover, it introduced novel features

that made RAFT more robust and adaptable than other consensus algorithms. Such features are (i) *Strong Leader*, which states that log entries can only flow from the leader node to other nodes; (ii) *Leader Election Mechanism*, where RAFT introduced randomized timers to elect a leader, which resolves conflict simply and rapidly; and (iii) *Membership Changes*, which allows the distributed system to continue operating normally during configuration changes (replacing the set of nodes participating in the consensus process). This last feature was a key piece in the decision of employing RAFT as a based algorithm for our solution in which our modifications mainly, but not only, were concentrated on solving the split and merge problem in the consensus process.

The RAFT algorithm was designed to manage a replicated log containing state machine commands issued from clients. In other words, it implements a Replicated State Machine (RSM), where a collection of servers computes identical copies of the same state and can continue operating even if some server crashes. We said that the system is Crash Fault Tolerant (CFT) since the system remains available. In this model, the replicated log is a series of application commands where each server stores a copy of the log, which its state machine executes in a particular order. Since state machines are deterministic, each server will compute the same state and generate the same sequence of outputs for a given sequence of inputs. Managing to keep the replicated log consistent among the servers is the consensus algorithm finality.

4.2.1 Basic Functioning

The RAFT algorithm based its philosophy as follows: if a series of decisions must be taken, it is simpler and faster to elect a leader node and then have the leader coordinate the replication of the entries. As such, the RAFT algorithm starts by appointing a leader node and then endowing it with the responsibility of dictating to the other nodes the updates of the replicated log. Once a leader is established, it accepts the entries coming from clients, replicates them in the distributed system, and tells the nodes when is safe to apply (commit) log entries to their state machine.

There are three states that every node can take at any given time: *leader*, *follower*, and *candidate*. Figure 4.1, illustrate the transitions a node can take between each state. In normal conditions, only one leader node is active in the system. Nodes in the follower state do not issue new entries or requests on their own, instead, they respond to leader and candidate requests. A node turns into the candidate state when the presence of a leader is not detected.

Time is divided into *terms* of arbitrary length, which is numbered and increases monolithically. Each term is determined by the election of a leader and ends with a new

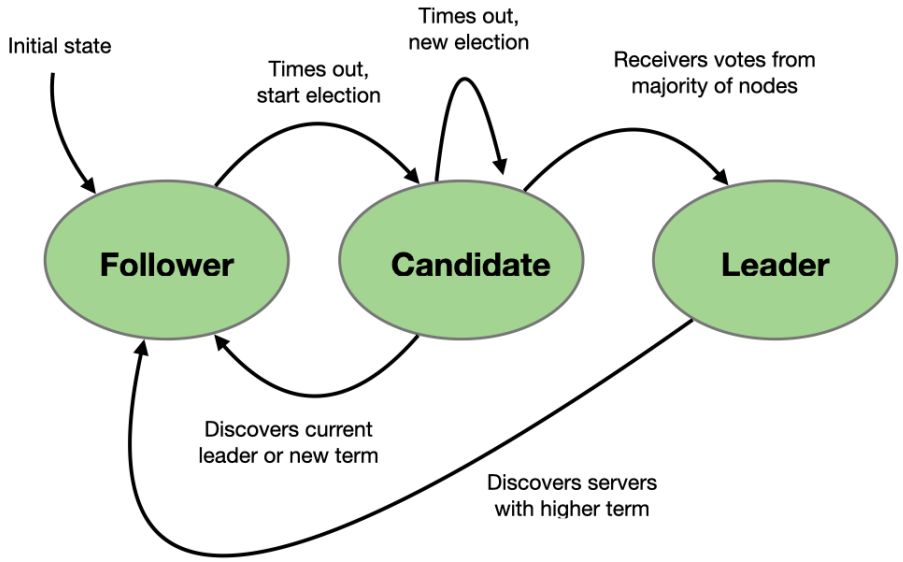


Figure 4.1: Nodes states of the RAFT algorithm.

election. Terms work as a time reference for every node, which allows for detecting stale information, such as unchanged candidates or obsolete leaders. Indeed, as described by the authors, some nodes may not observe an election or entire terms due to unavoidable delays in the network or node failures. Therefore, every node keeps a record of the current term, which is exchanged whenever nodes communicate. Thus, when a candidate node or a leader node receives information from another node holding a higher term, it immediately reverts to the follower state.

The original RAFT algorithm uses Remote Procedure Calls (RPCs) to communicate among nodes. Only two types of RPCs are needed to reach consensus: a *RequestVote RPCs*, which are used by candidates during elections, and *Append-Entries RPCs*, which are used by the leader node to replicate log entries and as a form of *heartbeat* to maintain its ruling.

The RAFT algorithm decomposes the consensus problem into three relatively independent subproblems, namely, *the leader election*, *log replication*, and *safety*.

4.2.1.1 Leader Election

At the start, every node is in the follower state. When a follower does not receive communication from a leader node over a determined period, called *the election timeout*, it assumes that the distributed system has no active leader node. It then increases its current term value and transitions to the candidate state to call for new elections. It

issues RequestVotes RPCs to every other node in the distributed system and grants a vote to itself. It remains a candidate until one of the following events occurs first. (i) Winning the election, (ii) receiving an Append-Entry RPC from a leader node, or (iii) a time period passes without a winner.

In the first case scenario, a candidate node wins the election by gathering a majority of votes, which is defined by *the majority criterion*, which is the minimum value needed to either become a leader or to replicate an entry before considering it committed. The majority criterion (m) is calculated in function of the number of nodes in the distributed system, $m = \frac{N}{2} + 1$, where N represents the total number of nodes in the distributed system. A follower node can only grant one single vote per term. Once the candidate node has won the election, it transits to the leader state and issues Append-Entries RPCs to every other node to maintain its ruling and prevent new elections.

In the second case scenario, the candidate node receives an Append-Entry RPC from a leader node. The candidate node will then compare the leader's term included in the RPC to take a decision. If the leader's term is at least as high as the candidate's current term, then the candidate transits to the follower state, otherwise, it continues in the candidate state.

In the last case scenario, more than one candidate may request votes, splitting the election so that no candidate manages to reach the majority criterion to win the election. In this case, each candidate's election timeout will exhaust, which triggers a new election round. To avoid falling into the same repetition pattern, randomization is employed in the election timeout timers to resolve this issue.

4.2.1.2 Log Replication

Once a leader node has been elected, it might start taking requests. In the RSM context, the leader appends the client's command to its log as a new entry and issues Append-Entries RPCs to every other node to replicate the entry. Entries are not immediately applied (commit) by a follower node upon the reception of an Append-Entry RPC; instead, followers wait for the leader's command to commit the entries. A leader node commits a client's command when the entry has been safely replicated in a majority of nodes according to the majority criterion.

The replicated log is composed of a succession of entries holding a command ordered sequentially. Each entry is identified by an index, *the logIndex*, which defines the position of the entry in the log. Moreover, each entry contains the current term in which the client's command was issued, which allows to better identify inconsistencies in the replicated log.

The leader node maintains a reference of the replication level for each of its followers.

To that, it stores two variables for each follower: *the nextIndex* and *matchIndex*. The nextIndex defines the logIndex of the next entry to be sent to that follower and the matchIndex defines the logIndex of the last entry known to be replicated by that follower. Indeed, each node evolves at its own pace due to multiple factors. Therefore, the leader node needs to keep track of the replication advancement of each of its followers to ensure consistency. Likewise, all nodes keep track of their own level of log advancement by storing two variables: *the commitIndex* and *lastApplied* index. The commitIndex stores the logIndex of the last entry known to be committed by the node and the lastApplied index stores the logIndex of the last entry added to the log by the node. The commitIndex is sent at every Append-Entries RPCs (including heartbeats) by the leader node so that every other node becomes aware of when to apply a given command. Committed entries are those that have been safely replicated in a majority of nodes and can never be reverted in any case. Figure 4.2 illustrates the RAFT replicated log along with the indexes.

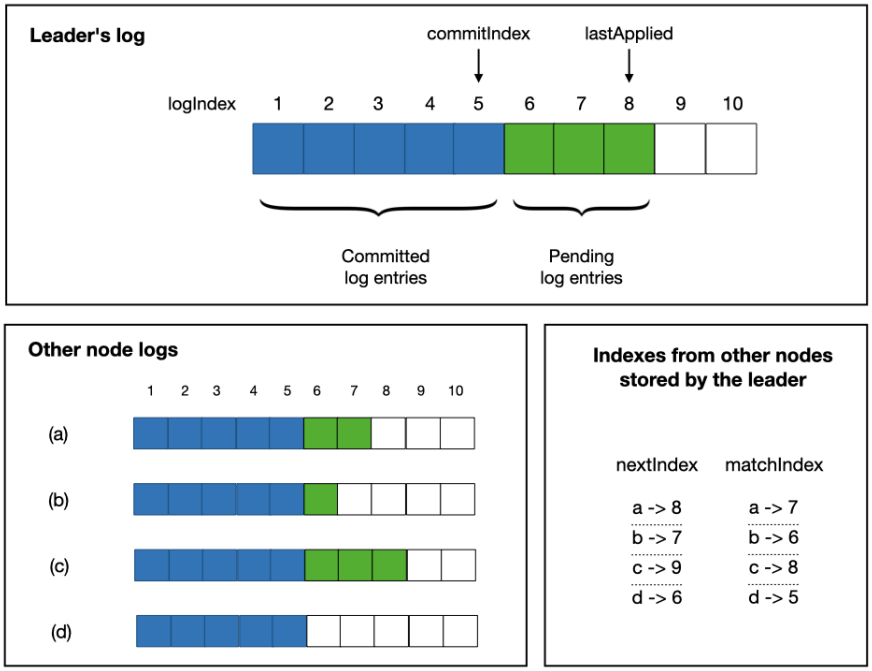


Figure 4.2: RAFT's replicated log.

A leader node can crash, leaving the replicated log inconsistent in the distributed system. Multiple case scenarios may exist in this situation; a follower may have extra entries that are not present on the leader, or it could be missing entries or both. In this case, the leader deals with those inconsistencies by forcing the follower nodes to duplicate its own log. To approach this issue, the leader node needs to find the last log entry where the two logs agree and overwrite the follower's log entries from this point

with the leader's entries.

4.2.1.3 Safety

The safety subproblem of RAFT consists in defining a series of rules that ensures that the same commands are applied in the same order in all state machines. Such rules can be summarized as follows:

- **Election Safety:** only one leader can be elected in a given term.
- **Leader Append-Only:** a leader can never overwrite or delete entries in its own log; it can only append new entries.
- **Log Matching:** if two different logs contain a given entry with the same log index and the same term, then the logs are identical in all entries up through that given index.
- **Leader Completeness:** if a log entry is committed in a given term, then that entry continues to be committed in the log of future leaders for all higher-numbered terms.
- **State Machine Safety:** if a node has applied a given entry at a given log index to its state machine, no other node will apply a different log entry for the same log index.

Those are the safety rules proposed in [19] that were thought for the RSM context. In the next Section, we will see that compromises on certain safety rules can be made in the Blockgraph context.

4.2.2 Membership Change

The RAFT algorithm was designed to function in a connected network using a fixed set of nodes participating in the consensus process. However, it has a special procedure to change the configuration. In [19], the authors point out that the reasons to make a configuration change in the distributed system are mainly occasional; this could be the case when replacing faulty nodes or updating the degree of replication.

Taking the distributed system down to perform a configuration change leaves the system unavailable during the process, which could be unacceptable for some use cases. Thus, to achieve a configuration change during operation, the authors proposed a two-phase approach, where there is no possibility during the transition of having two elected leaders for the same term. The approach goes as follows: it first switches to a

transitional configuration state called *the joint consensus*; this allows individual nodes to transit between configurations at different times. Once the joint consensus configuration has been committed, the systems switch to the new configuration; nodes that are not included in the new configuration can safely be removed. The joint consensus configuration includes nodes from both the old and new configurations. Therefore, during the joint consensus phase, log entries are replicated to all nodes in both configurations.

In the Blockgraph context, the membership change procedure proposed by RAFT cannot be employed since it assumes control over the physical location and assurances of network connectivity for all nodes involved during the membership change transition process. This is primarily because membership changes in Blockgraph cannot be foreseen or scheduled. Therefore, RAFT's join consensus solution, which needs the nodes from the old and new configuration to be available, cannot be assured in the Blockgraph context.

4.3 C4M: A Consensus Algorithm for Mesh and Mobile Ad Hoc Networks

In Blockgraph, the problem of reaching consensus remains the same but the entities and elements of the process change. For instance, a client no longer submits a series of commands needed for execution in a set of servers; instead, clients create application transactions to be included in a block in the Blockgraph. Thus, it is the blocks that need to be replicated in a majority of nodes to agree on the next block to be appended in the Blockgraph data structure. Therefore, the role of the replicated log changes completely from being the structure where the application data resides, in the RSM context, to being the structure where the control information necessary to achieve Blockgraph consistency resides. This new approach places two levels of replication information: the replicated log at the consensus level and the Blockgraph distributed ledger at the application level. In this regard, it is important to point out that due to frequent membership changes caused by mobility and the nature of the linear structure of the C4M log, in the Blockgraph context, the replicated log no longer guarantees *the Log Matching* property of RAFT's safety process, which makes the replicated log inconsistent at a given point. Indeed, at the time of a merge, where two different replicated logs encounter each other, both replicated logs have a different history of replication and cannot, therefore, be consistent with one another. However, the consistency of two separate replicated logs is not the objective at the consensus level, that is the

Blockgraph data structure objectives. Thus, in the case of a merge, it is only important to generate the initial conditions for the new configuration to begin agreeing on the new entries (replicating blocks in a majority of nodes) regardless of the previous log entries before the merge.

In Blockgraph, the distributed ledger is an append-only data structure composed of blocks containing application transactions. When a block is created through *the Blockgraph protocol* by a leader node, it sends the block to *the consensus module*. The block is then added as a log entry, which the leader node will try to replicate by sending the block to its followers. The followers are all nodes present in the current configuration. Once the block is replicated in a majority of nodes, the leader node instructs its followers that a given block in a given entry can safely be committed. The follower nodes become aware of the commitment of a block when receiving the Append-Entries messages, where protocol-related information is included. When a node commits a block, it transmits the block to *the Blockgraph protocol* for treatment.

The contributions of our solution focused primarily on the ability of the algorithm to perform frequent membership changes that we call configuration changes and to synchronize all nodes indexes during a merge. To that, a new operating mode is defined, which modifies the behavior of a node to go forward in the direction of a smooth configuration change. Next, we present the new operating mode in which our contributions are implemented.

4.3.1 Operating Modes

For the C4M consensus algorithm to adapt to frequent changes in the configuration, we define two operating modes that a node can take: a *stable topology mode* and a *topology change mode*. Figure 4.3 illustrates the two operating modes. In the stable topology mode, the consensus algorithm operates in normal conditions, meaning, that the fundamental processes of electing a leader and replicating logs are executed as the RAFT basic functioning. On the other hand, when a node shifts to the topology change mode, the node will prioritize the upcoming configuration change before the leader election and the log replication processes. Naturally, the actions that a node will take in this mode are dependent on the state they are in.

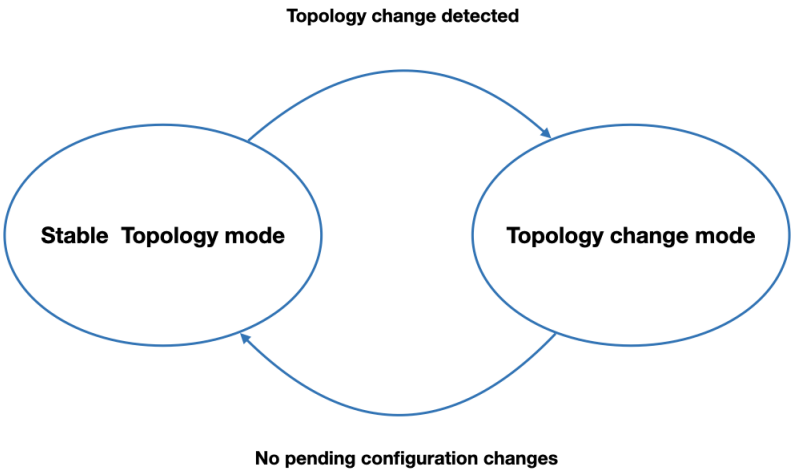


Figure 4.3: C4M operating modes.

4.3.2 Configuration Change

The shifting to the topology change mode starts with a topology change notification from *the group management module* 3.4.1 of the Blockgraph framework and ends with the commitment of a configuration change entry in the replicated log. It is the module that generates a new list of nodes that are available to participate in the consensus process upon the detection of a topology change in the network partition; it also provides the nature of the topology change, meaning, a split or a merge. Each node in the network partition holds two new control variables as part of the consensus process: *the currentConf* and *the pendingConf* variables. The currentConf variable stores the current list of nodes participating in the consensus process and the pendingConf stores the most updated list of nodes in the new network partition.

A configuration change only occurs when the node is in the topology change mode. This operating mode will provide the conditions for the leader node to enforce a change of configuration as fast as possible. In the Blockgraph context, a configuration change is defined by the addition or removal of nodes from the currentConf variable, either because new nodes are introduced in the overall distributed system or either because the mobility of nodes entails topology changes, meaning, nodes arriving or leaving the network partition. For the new configuration to take effect, a special entry called, *the configuration change entry*, is added by the leader node in its log. The configuration change entry holds the list of nodes, viewed from the leader's perspective, of the new set of nodes that will be participating in the consensus process upon the commitment

of the special entry. We will further see that the configuration change entry is added differently to the leader's log depending on the nature of the topology change.

In a connected network context, the addition or removal of new nodes is controlled by the administrators; in the mobile and dynamic network context, such control is not possible. Therefore, a joint consensus approach as proposed in RAFT [19] is not viable for our case scenario since we must infer the possibility of removing the leaving nodes at the time of committing the new configuration. However, the node's autonomy prevents us from ensuring that such timing is possible. Thus, a new approach, said: "on-the-fly" is proposed.

When a node is in the leader state during the topology change mode, it first has to figure out two things before taking any actions: the role that it's going to take during the configuration change phase and the nature of the topology change, which is given by *the group management module*. To know whether the leader node remains the leader during the configuration change phase, we define two new concepts: *the new majority* and *the new minority*. A leader knows if it belongs to the new majority or the new minority by comparing its currentConf with the new pendingConf.

If more than half of the nodes, including itself, from the currentConf, are also present in the pendingConf, it means that the leader node is part of the new majority in the new configuration and therefore, remains as a leader. In the case scenario where the leader node is part of the new minority, meaning, that less than half of the nodes in its currentConf are present in the latest pendingConf, it transits to the follower state. In other words, if the leader node is the node leaving the network partition, it transits back to the follower state, and if the leader node stays in the network partition, it remains the leader to deal with the configuration change.

If the leader crashes or a candidate node takes over the leader's position, the new leader can carry out the configuration change process without further actions.

Depending on the topology change nature, the leader node will proceed to the configuration change differently.

If the topology change refers to a split, the leader node will stop adding new block entries to its log, adds a configuration change entry at the next available entry, and prioritizes the commitment of the remaining entries in its log. Once the configuration change entry is committed in all nodes from the pendingConf, it transits back to the follower state to elect a new leader. In the case of the new network partition resulting from the split has no leader node, a leader is elected and performs the same procedure.

If the topology change is a merge and the leader node is part of the new majority, it will follow the same procedure as in the split but will add the configuration change entry based on the new minority logs. Indeed, for the new configuration to take effect in a merge case scenario, the indexes of all nodes in the pendingConf must be aligned to restart the consensus process. The log realignment process is further explained in Section 4.3.3.

In either case, whether is a split or a merge, the leader node has to commit the remaining entries in its log. To that, the leader has to choose which majority criterion to adopt. As we have already explained, the majority criterion is the minimum degree of replication needed to commit an entry, and it is based on the number of nodes in a configuration. However, during the configuration change phase, two configurations are available the currentConf and the pendingConf.

The leader chooses the majority criterion for the configuration change phase based on the nature of the topology change. Indeed, a different majority criterion is applied for a split that is for a merge. In the case of a split, the majority criterion is based on the pendingConf, since nodes from the currentConf are no longer available. On the other hand, in the case of a merge, the majority criterion is based on the currentConf, since block entries are meant to be committed in the current configuration.

While block entries are committed based on the majority criterion, a configuration change entry needs unanimity to be committed. In other words, the entry has to be replicated in all nodes before the leader node can advance its commitIndex at the logIndex position value where the configuration change entry is placed. Once the configuration change entry is committed, the pendingConf will become the new currentConf, and all nodes in the currentConf will switch to the stable topology state.

Follower nodes in the topology change mode will act differently whether they are included in the new majority or new minority. If a follower node is in the new minority, it will be allowed to respond to AppendEntries messages from a leader node not included in its currentConf but present in its pendingConf. However, it will only update its state if the AppendEntry message includes a configuration change entry or stuffing entries.

If a follower node in the new minority transits to the candidate state, it will ask for votes only to nodes in their currentConf since the pendingConf is not yet applied. In this regard, if the candidate manages to become a leader, it will transit back to the follower state since the node is still part of the new minority. On the other hand, if the follower node is in the new majority, its behavior does not change from the basic RAFT functioning.

4.3.3 Log Realignment Process

The log realignment process is a special procedure that occurs during the merge of diverse network partitions. This procedure is necessary to align the indexes used by the C4M protocol, inherited by RAFT, to achieve consensus among a set of nodes. Indeed, by aligning the indexes of all the nodes forming the new configuration, the ground is set to start a new consensus process. This desynchronization of the indexes occurs because the rate at which the Blockgraph branches grows is different for each network partition; it depends on the number of nodes participating in the consensus process and the cadence in which transactions are arriving the system. Therefore, when a merge arises in the network topology, the indexes used to maintain the replicated log as the content of the replicated logs are different in each network partition.

In regards to the difference in the content of the two replicated logs, we consider that up to the merge, each network partition has managed to maintain a consistently replicated log. Thus, the new network partition should start a new replicated log regardless of the previous content. This consideration is possible because the replicated log is used as an underlying layer that provides blocks agreed by a majority of nodes to *the Blockgraph protocol* to be appended in the Blockgraph data structure. Once an entry in the replicated log has been committed, it becomes merely historical record keeping. It is the Blockgraph data structure that needs to maintain the full consistency of the distributed ledger by only appending blocks that were committed at the consensus level.

On the other hand, realigning the different indexes used by C4M is capital to synchronize the new set of nodes (pendingConf) participating in the consensus process and will provide the proper conditions to begin a new consensus process to ensure agreement on the future blocks.

It is also worth mentioning that in the case of a split, the configuration change in both network partitions equals the removal of nodes in the distributed system. Therefore, there is no need to realign the indexes upon this configuration change.

In C4M as in RAFT, each node keeps track of its log's advancement by using the commitIndex and lastApplied index. Both indexes are updated upon the reception of an AppendEntry message sent by the leader node. The leader node, on the other side, holds two more variables for each of its followers: the nextIndex and the matchIndex. These indexes are updated upon an AppendEntry reply message from a follower node.

The realignment process of the indexes is performed by the standing leader node in a change topology operation mode during the configuration change process of a merge. It is important to remember that a leader node only remains a leader if he belongs to the new majority.

Upon the reception of the new configuration by *the group management module*, the leader node starts sending heartbeat messages to the new arriving node. The new arriving nodes form the new minority, and therefore, no leader nodes can be among them. The follower nodes are only allowed to respond to AppendEntries messages from a leader node, if this last is not present in the follower's currentConf and if the node is in the topology change mode.

Upon the reception of an AppendEntry message (heartbeat) of the leader node by the follower nodes forming the new minority, the follower nodes will reset their election timeout timer and respond with an AppendEntry reply containing their *current term*, *commitIndex*, and *lastApplied* index values.

This process allows the leader node to maintain its ruling during the configuration change process and to establish at which logIndex to place the configuration change entry. To that, it will compare its own lastApplied index with the highest commitIndex of the nodes from the new minority. If the commitIndex has a higher logIndex than the leader's lastApplied index, then the leader node will place the configuration change entry at the logIndex = commitIndex + 1. If the highest commitIndex from the new minority is inferior to the leader's lastApplied index, then it will place the configuration change entry at the logIndex = lastApplied + 1. At this point, the configuration change entry is added to the leader's log, in a way that, the leader node can either add stuffing entries to its log and applied them to its new majority, or force the new minority to add stuffing entries after their commitIndex until the leader's lastApplied index. This process allows realigning the indexes of all nodes in the pendingConf such as the configuration change entry is in the same logIndex for all nodes in the pendingConf. Figure 4.4 illustrates this process for the two case scenarios.

4.4 C4M Implementation and Evaluation

To assess the efficacy and to evaluate our solution's performance, we have opted for a simulation approach since it allows us to evaluate the potential effects of our solution as in a real implementation. To that end, we have chosen to employ the discrete-event network simulator for Internet systems, *NS-3*.

To simulate a mobile network, we used the NS-3 modules that allow us to implement in every node, an IEEE 802.11g technology configured in the ad hoc mode for the communication protocol. The radio frequency in which this technology work is in the range of 2.4 GHz with a data rate of 11 Mbps. The propagation loss model used for our simulations is the *RangePropagationLossModel* with a maximum range (MaxRange)

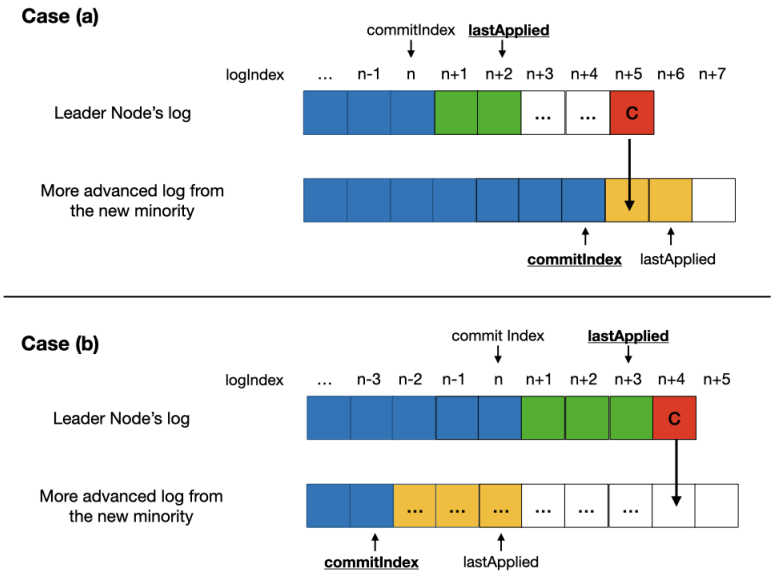


Figure 4.4: Logs realignment process.

of 250 meters. This propagation model is convenient for our use case scenario since it allows us to effectively schedule splits and merges topology changes during our simulations. Moreover, we have installed a full Internet protocol stack in every node, which aggregates IP/TCP/UDP functionalities to the mobile nodes. In this regard, we installed the IPv4 protocol for addressing, OLSR as the routing protocol, and UDP as the transport protocol.

For the C4M algorithm, we created an application for NS-3 implementing our C4M solution. To that, we first implement the RAFT basic functioning, meaning, the leader election and the log replication processes; then, we implemented our modification as described in Section 4.3.1 to create the C4M algorithm. The replicated log is represented by a linear vector containing pair data structures, where each entry contains the term when the entry was first added to the log and a random sequence of characters representing the hash of a block. For simplicity, new entries can only be generated by a leader node. The C4M algorithm uses a costumed *ApplicationPacket* class that handles protocol-related messages. It allows us to create our application packet as a form to substitute RAFT's RPCs.

To simulate the mobility of the nodes, we created an NS-3 application that updates the position of the nodes through time. To that, we installed in all nodes the *ConstantPositionMobilityModel*, which is an NS-3 model that allows us to set the position and the velocity of a node; these parameters do not change until explicitly set with a new

value. Thus, our mobility application performs the task of setting new position values according to the direction we want the nodes to take at a given simulation time. In Section 4.4.1, we define the mobility scenarios use for our simulations. Additionally, we implemented our *group management module* in this application. In that regard, we used the same approach described in Section 3.4.1.1 to discover the network topology.

Traces to measure our metrics are collected by using control variables that are updated constantly during the simulations. Those variables are printed in a log file for a post-treatment.

4.4.1 Methodology, Simulations, and Results

The objective of our simulations is to validate the efficiency, describe the characteristics, and define the limits of our solution. To that, we placed special care on the measure of metrics related to the leader election and the configuration change processes, such as the number of messages to elect a leader, the number of messages to commit a configuration change, the configuration change delay, the leader election efficiency, and the configuration change efficiency. We tested our C4M algorithm under mobility conditions and introduced topology changes during the simulation to assess the behavior of the configuration change procedure and the realignment log process. Thus, we have defined two mobility scenarios in which our algorithm is tested to contrast results obtained from both scenarios. In each scenario, the nodes starts with the same network topology configuration, which is a fully connected network. The description of our mobility scenarios goes as follows:

- **Scenario 1 - Split and Merge:** in this scenario, the nodes move in the same direction at a constant speed. Two groups of nodes are defined: even and odds nodes. A split is scheduled after 100 seconds of simulation by changing the direction in opposite ways in which each group moves; two different network partitions are then formed for a period of 100 seconds of simulation time. A merge is then scheduled by changing the direction of both groups towards each other; the network will then form a connected network for a period of 100 seconds. This pattern of split and merge repeats every 100 seconds until the end of the simulation.
- **Scenario 2 – Connected Network:** in this scenario, the nodes move in the same direction at a constant speed. All nodes are close enough to form a single network partition. No topology changes occur during this scenario. Thus, all nodes form a single connected network throughout the simulation.

The simulation time for both scenarios is set to a fixed 600 seconds. Scenario 1 aims to test our C4M algorithm in the context of topology changes, while scenario 2 is used as a reference scenario to contrast the performance evaluation of scenario 1 in a stable connected network. For each scenario, we vary the number of nodes in the system to evaluate the scalability of our solution. The values defining the number of nodes used for each scenario are 6, 10, 16, 20, 26, and 30 nodes.

We identified two parameters that have a direct impact on the behavior of the leader election and the configuration change processes: the AppendEntries messages interval (or heartbeat interval) and the election timeout window. The heartbeat interval is the time between two AppendEntries messages sent by a leader node to its followers, and the election timeout window is the range of time in which a node randomly chooses a value to set its timer. As a manner of reference, we included the original values proposed by RAFT to compare the performances of both algorithms, namely, a 50 ms heartbeat interval and an election timeout window of [300-450] ms. The rate at which entries are added in the replicated log is configured to happen 1 entry every 5 seconds, with a 50% probability of success of adding the entry. This, allow us to desynchronize the log indexes between the distributed log of different network partitions and help us create the conditions to test our log realignment procedure. Table 4.1 summarizes the C4M parameters used for the simulations.

C4M Simulation Parameters

Parameters	*Values*
Entry log rate	0.2 entry/s
Heartbeat interval (HB)	50 ; 500 ; 1,000 ms
Election timeout window (Elto)	[300-450] ; [600-750] ; [1,600-1,750] ; [4,000-4,150] ms
Number of nodes	6 ; 10 ; 16 ; 20 ; 26 ; 30
Propagation loss model	RangePropagationLossModel (MaxRange 250 m)
Routing protocol	OLSR
Transport protocol	UDP

Table 4.1: Values of the C4M parameters for the simulations.

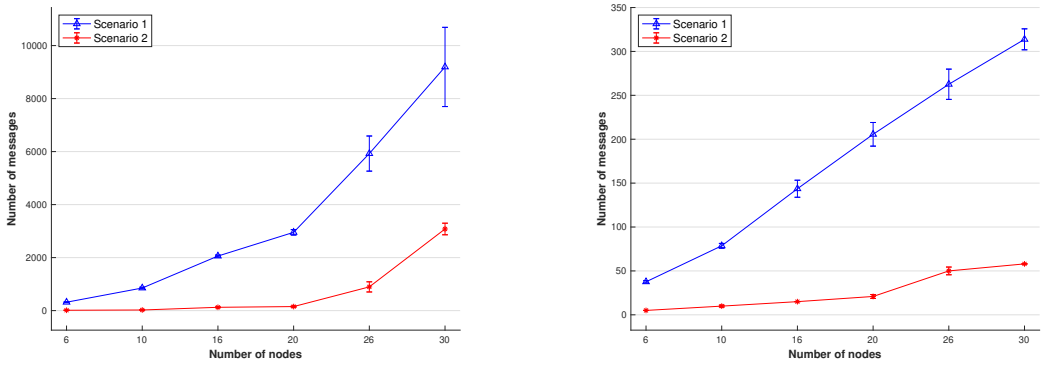
To obtain better and more accurate results, we run each configuration of each scenario 10 times. Thus, the results are average values with a confidence interval of 95%. It is also important to notice that the heartbeat interval must remain shorter than the election timeout window. Thus, for a 500 ms heartbeat only [600-750], [1,600-1,750], and [4,000-4,150] election timeout windows are considered for simulations. Likewise, for a 1,000 ms heartbeat only [1,600-1,750] and [4,000-4,150] are considered. Finally, and for the sake of simplicity, we choose to display only results for 50 and 1,000 ms

heartbeat intervals since 500 ms results were must the time redundant with 1,000 ms results. In this regard, 500 ms results have a similar performance that 1,000 ms results unless said otherwise.

4.4.1.1 Leader Election Results

First, we analyze the number of leader election messages of C4M. The leader election procedure uses two different messages: *VoteRequest* and *AckVote* messages. When the election timeout timer reaches *zero*, the node switches to the candidate state and sends a VoteRequest message to all nodes in the same currentConf. Upon the reception of a VoteRequest message, the node responds with an AckVote message by either granting or denying its vote. It is important to point out that not every VoteRequest culminates in the election of a leader. Figure 4.5 compares the number of leader election messages generated for each scenario and different network sizes. Figure 4.5a uses a heartbeat interval value of 50 ms and an election timeout window of [300-450] ms, while Figure 4.5b uses a heartbeat value of 1,000 ms and an election timeout window of [4,000- 4,150] ms. As expected, in scenario 1, the size of the network has a significant impact on the number of leader election messages. Interestingly enough, the heartbeat rate and the election timeout window play an important role in the number of election messages. We can notice that in Figure 4.5a the curves tend to grow exponentially, while in Figure 4.5b the curves present a linear growth. These observations are valid for both scenarios. This difference is mainly due to network congestion since reducing the heartbeat interval, and the election timeout window implies an increase in network traffic. Thus, as network congestion escalates, packet loss increases, causing the efficacy of the election procedure to shrink. These problems might lead not only to an augmentation in the number of VoteRequests to consolidate an election but also to an increase in the total number of elections. Moreover, the short election timeout interval increases the chances to trigger new leader elections. In contrast, Figure 4.5b shows a more linear progression according to the size of the network and a much lower number of leader election messages. Thus, a higher election timeout window allows the leader node to maintain its leadership for longer periods.

Figure 4.6 shows the number of VoteRequest and AckVote messages for scenario 1 using extreme heartbeat interval and election timeout values. In an ideal network condition, the number of VoteRequest would be equal to the number of AckVote messages. However, we can notice in Figure 4.6a that the gap between the number of VoteRequest and AckVote messages tends to increase exponentially with the size of the network. We attribute this difference to the loss of some of the AckVote messages. As we have seen in the previous result, a 50 ms heartbeat interval can cause the saturation

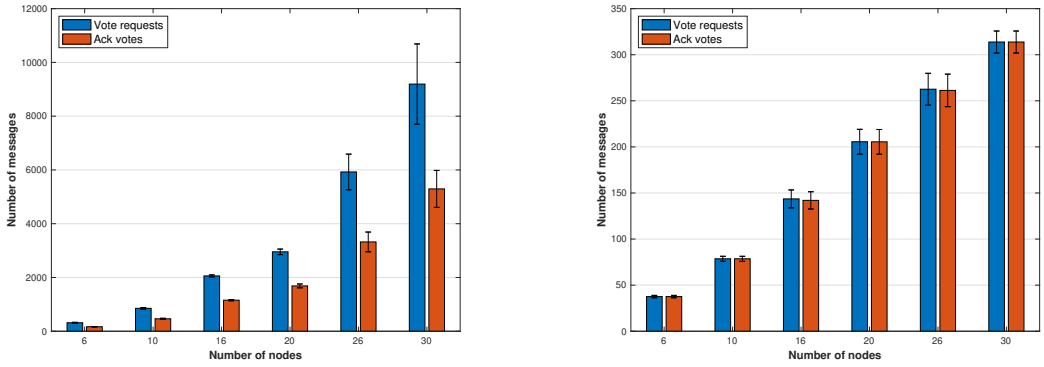


(a) HB 50 ms - ETo [300-450] ms.

(b) HB 1000 ms - ETo [4000-4150] ms.

Figure 4.5: Number of leader election messages.

of the network for large networks, which causes high packet losses and longer delay. These losses combined with a short election timeout window ([300-450] ms) can severely impact our algorithm by triggering several leader elections. It might also cause the algorithm to stay in a constant leader election process when using shorter intervals. In contrast, Figure 4.6b shows 10 to 30 times fewer leader election messages with almost no loss of AckVote messages. Therefore, these results confirm the significant impact of the heartbeat interval on the leader election performance and show that a less frequent heartbeat interval and a longer election timeout window allow for better conditions in mobile and topology changing networks.



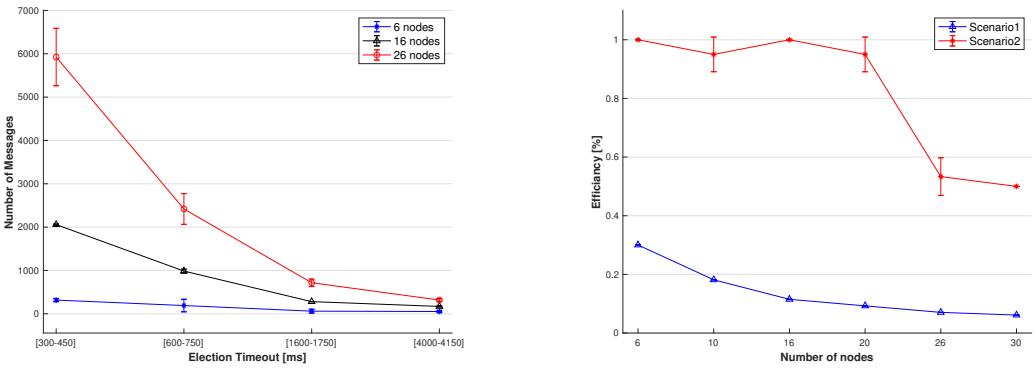
(a) HB 50 ms - ETo [300-450] ms.

(b) HB 1000 ms - ETo [4000-4150] ms.

Figure 4.6: Number of *Vote Request* and *Ack Vote* for scenario 2.

Figure 4.7a shows the number of leader election messages generated for a network size of 6, 16, and 26 nodes according to different election timeout window values, with a fixed heartbeat of 50 ms. We can notice that the difference in the number of messages is more important for a shorter election timeout window. Indeed, part of this phenomenon is explained by the fact that larger networks require more messages to elect a leader node.

Moreover, we can also notice that the number of leader election messages decreases with the higher values of election timeout. This is explained because longer election timeout timers give more time for heartbeat messages to reach the follower nodes in conditions of a congested network that entails packet losses and long delays. This conclusion is in line with Figure 4.7a, which shows that different network sizes tend to converge with an election timeout window of 4,000 ms. Figure 4.7b depicts the efficiency of the leader election process. We calculate the efficiency by dividing the number of successful leader elections by the total number of leader elections triggered. It is important to highlight that neither the heartbeat nor the election timeout parameters impact the leader's election efficiency. Only mobility and the size of the network have a significant impact on this metric. Indeed, one could think that the election timeout parameter may impact the leader's election efficiency due to its natural correlation. However, results show that regardless of the election timeout parameter, the proportion of successful leader elections is always around the same values. We notice that the leader election efficiency is much lower when nodes split and merge, as happens in scenario 1, than in scenario 2, when nodes stay together in a single connected network.



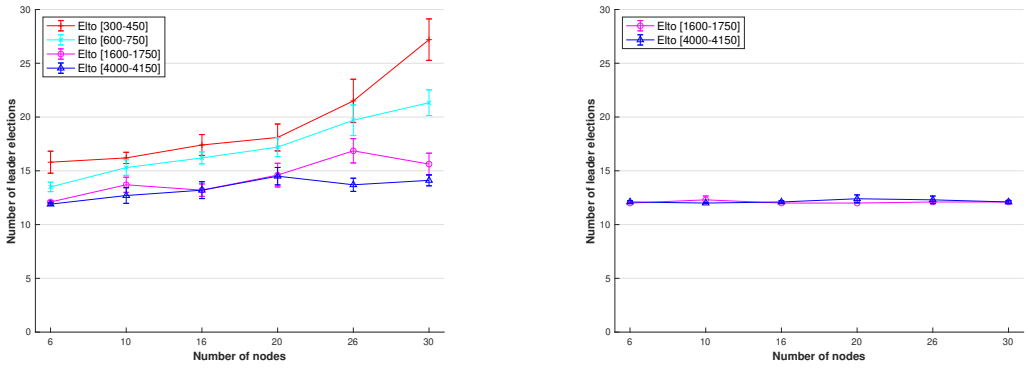
(a) Number of leader election messages with respect of the size of the network.

(b) Leader election efficiency

Figure 4.7: Leader election performance.

Figure 4.8 shows the number of completed leader elections for scenario 1. In this scenario, the network alternates between split and merge every 100 seconds for 10 minutes, meaning that at least 10 leader elections should occur if no leader is detected after each configuration change. In Figure 4.8a we can notice a gradual growth in the completed number of leader elections when we increase the number of nodes in the network for a heartbeat interval of 50 ms. We can also notice that simulation scenarios with a shorter election timeout window tend to acute this phenomenon. Indeed, here once again we can notice how a short timeout window prevents the leader from keeping its leadership. Figure 4.8b, on the other hand, shows a constant number of completed

leader elections. There are two main reasons for this result. First, a 1,000 ms heartbeat interval imposes a significantly lower network load, leaving less room for an eventual network saturation, which allows communications to take place correctly. Second, using higher election timeout window values gives the leader more time to keep its leadership, since nodes wait longer to call for new elections. We obtained similar results for a heartbeat interval of 500 ms.



(a) HB 50 ms

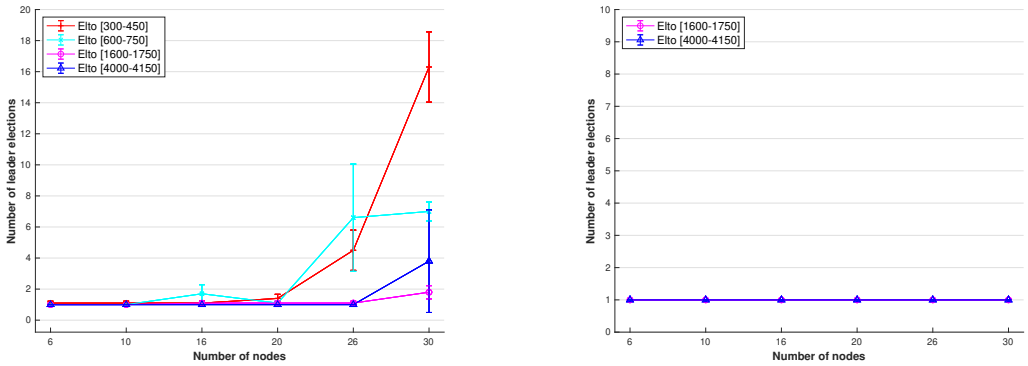
(b) HB 1000 ms

Figure 4.8: Number of completed leader elections for scenario 1.

Figure 4.9 shows the number of completed leader elections for scenario 2. In this scenario, despite mobility, nodes are connected in a single network partition. We can notice in Figure 4.9a that the number of completed leader elections is 1 for all election timeout window values, up to 20 nodes. This result is expected for this scenario since it only takes one leader election in a single network partition without any splits or merges. However, as the size of the network increases, the network condition degrades, causing packet loss and longer delays. This problem provokes new unnecessary elections, especially for shorter election timeout window values. Figure 4.9b shows that for a 1,000 ms heartbeat interval, all election timeout values are equal to 1. We observe a similar result for a 500 ms heartbeat interval. The longer election timeout window values combined with less frequent heartbeat intervals maintain the network stable and allow the leader to maintain its leadership.

4.4.1.2 Configuration Change Results

The group management module 3.4.1 is responsible for detecting changes in the network topology and identifying new network partitions to form the new group of nodes allowed to participate in the consensus process. *The consensus module* receives this information and stores it in the pendingConf variable waiting for the leader node to commit this new configuration according to the conditions explained in Section 4.3.1. For our simulations,



(a) HB 50 ms

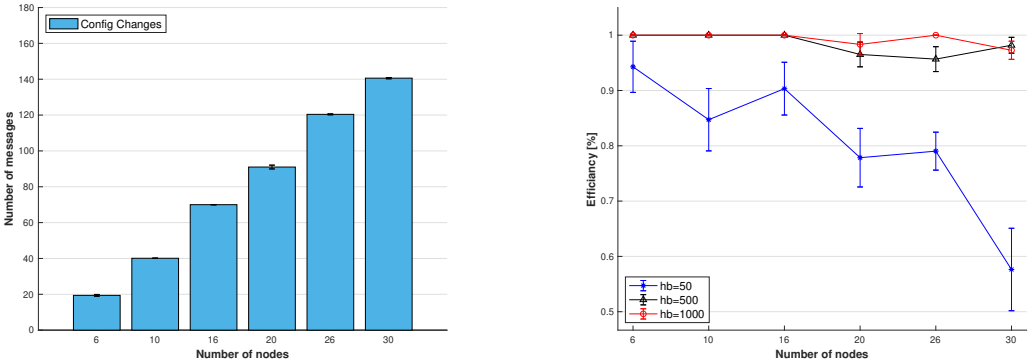
(b) HB 1000 ms

Figure 4.9: Number of completed leader elections for scenario 2.

we marked the messages related to the configuration change process to better treat our results. Hence, AppendEntries messages with a configuration change entry are marked as *AppendConfig* messages for the simplicity of the results treatment.

Figure 4.10 depicts the performance of the configuration change process in scenario 1 with an election timeout window of [4000-4150] ms. In Figure 4.10a, we show the number of AppendConfig messages in scenario 1 with a heartbeat of 1,000 ms. We can notice that the number of configuration changes messages increases with the size of the network. This effectively means that the more nodes in the network, the more AppendConfig messages are issued. This is explained by the inability of *the group management module* to effectively provided the consensus mechanism with an updated list of nodes present in the network partition. Thus, triggering new AppendConfig messages. Nevertheless, the increase of AppendConfig messages between network sizes has a linear progression. Figure 4.10b shows the configuration change efficiency for different heartbeat values. We can notice that the heartbeat interval has a direct impact on the configuration change efficiency. A 50 ms heartbeat tends to lose efficiency as the network grows, while a 500 and 1,000 ms heartbeat manages to maintain a high efficiency above 95%, which indicates that most of the configuration changes are correctly committed. Hence, even though our *group management module* struggles to provide a correct view of the network partition, the C4M algorithm using a 500 or 1,000 ms heartbeat and a high election timeout window manages to correctly commit the frequent configuration changes.

The configuration change delay is the time from the moment *the group management module* notifies the C4M algorithm of a topology change until the new configuration is committed. Figure 4.11 shows the results of the average number of completed configuration changes during simulation for different values of the election timeout

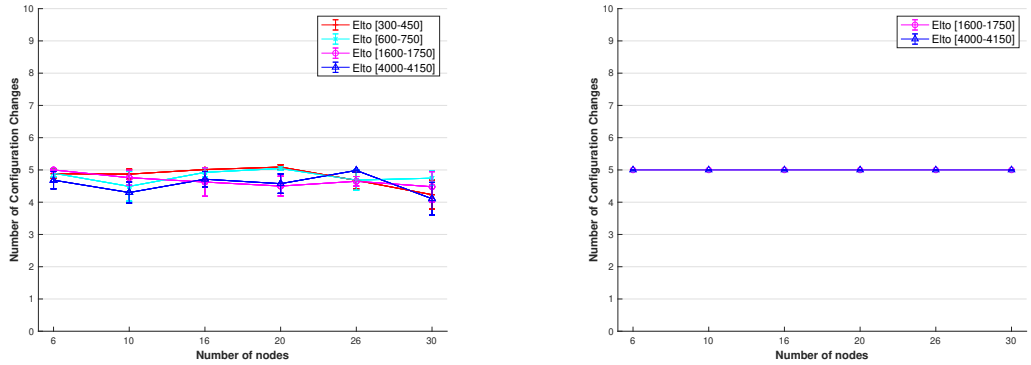


(a) Number of Configuration change messages

(b) Configuration Change efficiency

Figure 4.10: Configuration change performance in scenario 1.

window using a particular heartbeat interval. We can notice that in Figure 4.11b all different election timeout window values converge to the exact number of configuration changes expected. This result is also valid for a heartbeat interval of 500 ms. However, Figure 4.11a shows a slight variation in the number of configuration changes executed. Nevertheless, the variance between election timeout values is barely perceived. These results confirm that the election timeout parameter has no impact on the correctness of the execution of our configuration change procedure, as already mentioned. In any case, the variance experienced among the different values of the election timeout window in Figure 4.11a can be attributed to the negative effect of a short heartbeat interval.



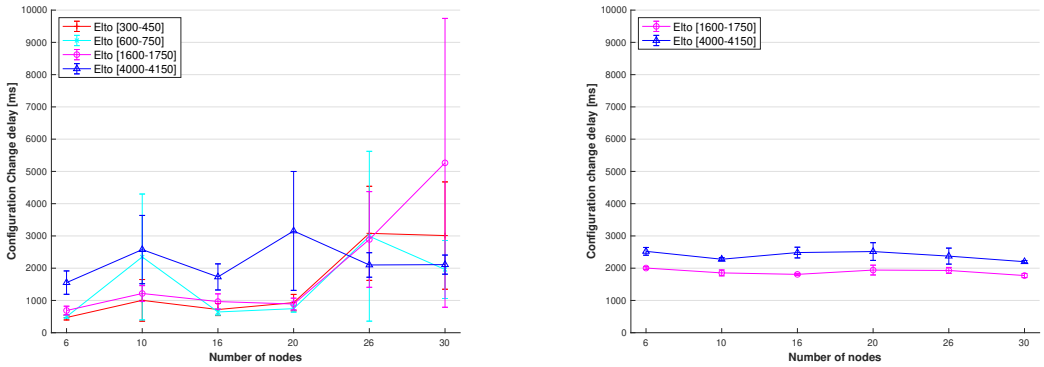
(a) HB 50 ms

(b) HB 1000 ms

Figure 4.11: Number of completed Configuration Changes for scenario 1.

Figure 4.12 illustrates the results of configuration change delays for different election timeout values and different numbers of nodes in the network using a particular heartbeat interval. We can observe that varying the heartbeat interval and the election timeout window have little impact on the delay of the configuration change process. This difference is due to the increase in the heartbeat interval since configuration changes are

propagated through heartbeat messages. In addition, Figure 4.12a reveals that smaller values of the election timeout window still perform better than higher values. Indeed, smaller values imply shorter election delays. Thus, given that configuration changes are committed by the leader, shorter election delays allow leaders to react faster to configuration changes.



(a) HB 50 ms

(b) HB 1000 ms

Figure 4.12: Configuration Change delay for scenario 1.

Finally, Table 4.2 presents the message overhead of C4M in terms of the number of messages and the traffic volume considering all types of C4M messages. We can observe that although heartbeats and ack_entries correspond to 93% of the total messages, their amount of traffic represents only 15%. Likewise, block messages only represent 3% of the total messages but they correspond to 85% of the traffic volume. In all cases, configuration change entries and election-related messages correspond only to a small fraction of the total number of messages and data volume.

Proportion of messages in number

Type of message	*Percentage*
Vote Request	2%
Ack Vote	1%
Data Blocks	3%
Config Change Entries	$< 1\%$
Heartbeats	45%
Ack Entry	48%

Proportion of traffic overhead in bytes

Type of message	*Percentage*
Vote Request	$< 1\%$
Ack Vote	$< 1\%$
Data Blocks	85%
Config Change Entries	$< 1\%$
Heartbeats	8%
Ack Entry	7%

Table 4.2: Message overhead of C4M in number and volume.

4.4.2 C4M Implementation Conclusion

Recent solutions for IoT applications, vehicular networks, and mesh networks rely on the mobile ad hoc networks' paradigm. However, maintaining a consistent distributed

database in MANETs is not a trivial task. In this regard, we have described all the challenges of using non-partition tolerant algorithms in a mobile network context. Therefore, we have proposed a new consensus algorithm for Blockgraph that is robust to network partitions. C4M is a consensus algorithm based on RAFT that achieves this goal by introducing a new operation mode that prioritizes configuration changes to follow up with nodes' mobility. Our simulation results demonstrate that our algorithm responds to a correct characterization, behaves well, and fulfills its purpose.

We conclude that the original values proposed in [19] do not perform well in our context. Indeed, the added complexity and the node mobility require a less frequent heartbeat interval and a longer election timeout window. Our results show that our algorithm performs best for a 500 ms heartbeat interval and an election timeout window of [1600-1750] ms for all the parameters we have chosen. However, these values may vary accordingly to the size of the network and other factors, such as the nodes' speed. We also evaluate the efficiency of our leader election and configuration change processes in the presence of network topology changes. Results show that certain configurations of our parameters do not impact the efficiency of these processes. Nevertheless, it demonstrates the importance of correctly sizing the network. Additionally, we present the message overhead of C4M, considering both the volume of data sent, as well as, the number of messages. Our results demonstrate that even though control messages of the protocol (messages not containing entries) have a large proportion of the overall number of messages sent during the simulations, it only represents a small proportion of the total data volume, which gives a good efficiency to the protocol.

4.5 Chapter Summary

In this chapter, we presented the importance of consensus algorithms in a distributed system. We first emphasized the relevance of choosing between a deterministic and a probabilistic consensus algorithm, according to the characteristics and needs of the top-level application and the network. We highlighted the need for mobile ad hoc and mesh networks to count on a consensus mechanism capable of keeping up with the mobility of the nodes. In this regard, we present C4M, a Blockgraph consensus algorithm tolerant to network partitions based on RAFT. Our C4M algorithm differentiates from RAFT by adding a temporary state: the partition mode state. This new state shifts the prioritizing task of the algorithm from the log replication process to the configuration change process. Moreover, it changes how a membership change is executed. Indeed, C4M performs its configuration change while the nodes are mobile, on-the-fly, while RAFT performs its membership change by controlling all nodes involved in the configuration change process.

To that, we have modified the configuration change process and introduced a index realignment procedure. To test our solution, we implemented C4M in the discrete-event network simulator, NS-3. Finally, we present the result of the performance of our algorithm by performing extensive simulations which aim to characterize and validate the feasibility of our solution.

Chapter 5

Conclusions

In this thesis entitled “Blockchain application in Mesh and Ad hoc Networks”, we presented a full blockchain-like solution for Mesh and Mobile Ad Hoc Networks. We introduced the need for these networks to rely on a blockchain-like technology to secure new and legacy applications in the context of mobility and network partitions. Blockgraph is the result of this challenge for which we validate our concept through simulations and the implementation of a prototype. We conclude this thesis by giving a summary of our contributions and proposing several perspectives for each of them.

5.1 Summary of Contributions

Each of our contributions unfolds in the context of maintaining a blockchain-like data structure in a Mobile Ad hoc Network where the changes in the network topology can lead to multiple network partitions. In this regard, we believe that the Blockgraph concept along with the Blockgraph framework is our first contribution. Blockgraph is a new concept where the use of a DAG-based blockchain-like data structure is employed to respond to the split and merge problem without neglecting the intrinsic properties that blockchain technology provides. On the other hand, the Blockgraph framework allows the separation of the three fundamental sub-problems linked to the maintenance of the Blockgraph, namely, a network discovery solution, the consensus algorithm, and the Blockgraph data structure management. By designing our solution in a modular fashion, we also provide the opportunity to make changes and adaptations easily without impacting the whole system. A future system designer might want to use a proprietary solution to discover the network topology other than the one we provide or might want to use a different consensus algorithm that better adapts its needs. The Blockgraph framework facilitates all this. We presented this contribution in a conference paper [21] where we validate our Blockgraph concept through simulations and results.

We then directed our attention to implementing a testbed version of our Blockgraph solution in a real distributed system composed of five low-power mesh routers. Although the validation of our concept through simulation provided the first results of the Blockgraph performance, the implementation of the testbed brought improvements in the algorithmics of our system, which results in a new and more efficient way to perform *the merge synchronization procedure*. Thus, the results of our testbed not only reaffirmed our Blockgraph concept but also improved the performance of the system compared with the simulation results and become a data collection source to feed our simulator. We presented the implementation of our proof-of-concept to the community [23], which was awarded second place in the ACM Student Research Competition in 2021.

Finally, we decided to work on a consensus mechanism capable of tolerating network partitions to integrate with our Blockgraph solution. To achieve this goal, we were inspired by RAFT's approaches to reaching consensus, to create a tolerant-partition consensus algorithm that we called, C4M. The problem of this contribution was mainly focused on being able to update the list of participant nodes in the consensus process during execution and synchronize the different distributed logs upon network partitions. By using our network discovery solution, we were able to provide C4M with updated topology information and by using a novel index realignment procedure, we managed to synchronize all distributed logs after a merge. This work resulted in the publication of a conference paper [22], which we extended into a journal article in process of publication.

5.2 Perspectives

This section concludes our dissertation by presenting the future line of work that we identified for our contributions.

5.2.1 Blockgraph

The main objective of Blockgraph is to provide MANETs with a blockchain-like technology to secure in a distributed fashion new and legacy applications. Nevertheless, at the current state of our solution, Blockgraph only supports permissioned access to the system. Thus, all nodes participating in the distributed ledger are well-known by the system, which entails a form of centralization that limits potential applications. We, therefore, believe that exploring the possibility of decentralizing the Blockgraph in future work, could provide MANETs with the benefits of decentralized applications (dApps). To do that, it is important to review different aspects of our solution. As a fundamental layer, we shall use a new form of consensus mechanism that is scalable,

allows agreement in a trustless manner, and is partition-tolerant. However, existing consensus algorithms do not hold all these characteristics at once. We believe that some lines of work to resolve this issue can go forward in the direction of a form of economic incentive that privileges nodes by creating new connections with other network partitions. The work of Ghiro et al. [47] approaches this problem and could be an interesting starting point to making the Blockgraph decentralized.

We also believe that Blockgraph should support a form of tokenization that allows both inter-related transactions and independent transactions. Nevertheless, the Split and Merge problem still poses challenges to a tokenized form of transaction, since the lack of communication between several network partitions creates security issues that can easily be exploited by malicious actors.

Concerning the performance optimization of our Blockgraph solution, we believe that an important improvement that can be made is on the memory consumption usage. As we know, the Blockgraph is an ever-growing distributed ledger that increases its needed storage space as time passes. Thus, we believe that an integration of the sharding solution with our Blockgraph could reduce the memory requirements of mesh routers, which will also benefit the effective transaction throughput of the system. Indeed, integrating sharding in the Blockgraph not only optimize the resource consumption in term of memory space and CPU usage but will also be consistent with the resource restrictions of mobile mesh routers embedded system.

5.2.2 C4M

The C4M consensus algorithm responds to the need of relying on a robust consensus capable of tolerating network partitions. It was designed in the context of Blockgraph and it was meant to integrate the Blockgraph framework. In this regard, we believe to have identified the main characteristics that a consensus algorithm working in a mobile and dynamic network needs to have to overcome frequent changes in the set of nodes participating in the consensus process. Indeed, our Blockgraph framework uses a network discovery solution that provides updated network topology information to the C4M consensus algorithm. Nevertheless, it would be interesting to integrate this functionality as part of the consensus process without relying on external information. This will allow, from one side, to export this solution in a different context outside the blockchain scope and from the other side, to provide the algorithm with full independence to achieve consensus in a dynamic network. Another interesting perspective would be exploring a different form of distributed log data structure. C4M uses a linear data structure inherited from RAFT consensus, which was designed to function in a connected network. A DAG-based structure could potentially allow faster configuration changes in both

split and merge cases, which could positively impact the overall performance of the system.

Appendices

Appendix A

The Blockgraph Algorithms

A.1 The Group Management Module

Algorithm A.1: Network Topology Change Detection

```
1  Function TopologyChanges()
2  TOPOLOGY_TOLERANCE_TIME = 5 sec;
3  RouringTable RTable;
4  v_candidateList ← CreatCandidateList(RTable);
5  if currentGroupList != candidateList then
       /* A change in the network topology is detected                */
6      if currentTime() - time_of_last_change >
        TOPOLOGY_TOLERANCE_TIME then
           /* Enough time has elapsed since                           */
           /* the last detected change                                */
7          groupID ← CalculateGroupID(candidateList);
8          natChange ← DetectNatureChange(currentGroupList, candidateList);
9          currentGroupList ← candidateList;
10         time_of_last_change ← Clock();
11     else
           /* Not enough time has elapsed to                          */
           /* consider the topology stable                            */
12         return;
13     end
14 else
       /* No changes detected in the network topology                 */
15     return;
16 end
```

A.2 The Blockgraph Protocol

Algorithm A.2: Basic Transaction Treatment Algorithm

```
1  Function TransactionTreatment(Transaction T)
   /* Function executed upon the arrival of a transaction          */
2  if TransactionValid(T) == true then
   |  /* We check if transaction is present in mempool             */
3  |  if TxInMempool(T) == false then
   |  |  /* We check if transaction is present in the blockgraph   */
4  |  |  if TxInblockgraph(T) == false then
   |  |  |  /* We check if there is space in the mempool           */
5  |  |  |  if SpaceInMempool(T) == true then
   |  |  |  |  /* Adding the transaction to the mempool           */
6  |  |  |  |  mempool.add(T);
7  |  |  |  else
8  |  |  |  |  DropTransaction(T);
9  |  |  |  end
10 |  |  else
11 |  |  |  DropTransaction(T);
12 |  |  end
13 |  else
14 |  |  DropTransaction(T);
15 |  end
16 else
17 |  DropTransaction(T);
18 end
```

Algorithm A.3: Block Creation Algorithm

```
   /* Upon the conditions to create a block are meant, we decide
      what type of block we are creating                          */
 1 if SynchProcess == true then
   |  /* We generate a merge block                                */
 2 |  CreateMergeBlock();
 3 else
   |  /* We generate a data block                                 */
 4 |  CreateDataBlock();
 5 end
 6 Function CreateDatBlock()
 7 Block block ;
   /* Construct the block header                                  */
 8 block.SetLeader(this → node.GetID());
 9 block.SetGroupId(groupId);
10 block.SetBlockType(Data);
11 block.SetIndex(blockgraph.GetLastIndex + 1);
12 block.SetTimestamp(Clock());
   /* Get the last block from the local blockgraph as reference   */
13 prev_block_ref ← blockgraph.GetChildlessBlockHash() ;
14 block.SetReferences(prev_block_ref);
   /* transactions are selected from the mempool                  */
15 TransactionContainer  t_container;
16 t_container ← SelectTransactions();
17 block.SetTransaction(t_container);
   /* Send the block to the consensus module                      */
18 SendBlockToConsensus(block);
```

Algorithm A.4: Block Treatment Algorithm

```
1  Function BlockTreatment(Block block)
   /* function executed upon the arrival of a Block                    */
2  if BlockValid(block) == true then
       /* We check if the block is present in the blockgraph           */
3      if BlockInBlockgraph(block) == false then
           /* We check if the block is a missing block                 */
4          if BlockInMissingList(block) == true then
               /* We update the list of missing blocks                 */
5              UpdateMissingList(block.GetHash());
6          end
           /* We identify the type of block                            */
7          if block.GetType() == "Merge" then
               /* The node starts its synchronization process          */
8              StartSyncProcess(block.GetTransactions());
               /* The Merge Block is added to the blockgraph           */
9              blockgraph.add(block);
10         else
               /* The block is a data block                            */
11             if ParentBlockInBlockgraph(block) == true then
                   /* The block's predecessor is in the blockgraph.    */
                   /* The block can be added to the Blockgraph.        */
12                 blockgraph.add(block);
13             else
                   /* The block's predecessor is not in the Blockgraph */
                   /* Add the block's predecessor to the list of missing
                      blocks                                            */
14                 UpdateMissingList(block.GetParentsHash());
                   /* Add the block to a temporal emplacement          */
15                 temporal.add(block);
16             end
17         end
18     else
19         DropBlock(block);
20     end
21 else
22     DropBlock(block);
23 end
```

A.3 The merge synchronization procedure

Algorithm A.5: Send Childless block request to all followers.

```
   /* Only executed by the leader node                                  */
1  if this.node == 'leader' then
   |  /* Create a CHILDLESSBLOCK_REQ packet                            */
2  |  ApplicationPacket packet("CHILDLESSBLOCK_REQ");
3  |  for nodeDest : currentConf do
   |  |  /* Send the CHILDLESSBLOCK_REQ to all nodes in the
   |  |     currentConf                                                */
4  |  |  SendPacket(packet, nodeDest);
5  |  end
6  else
7  |  return;
8  end
```

Algorithm A.6: Send Childless block reply from a follower node to a leader node.

```
   /* Upon reception of the CHILDLESSBLOCK_REQ packet                   */
1  senderAddr ← packet.GetIPv4();
2  ChildlessBlockReply(senderAddr);
3  Function ChildlessBlockReply(Ipv4Addr addr)
   /* Get the local list of childless blocks                            */
4  myChildless ← blockgraph.GetChildlessBlockList() ;
5  serie_hash ← "";
6  for hash : myChildless do
   |  /* Add each childless block hash to the message.                 */
7  |  serie_hashes += hash;
8  end
   /* Send the CHILDLESSBLOCK_REP to the leader                         */
9  ApplicationPacket packet("CHILDLESSBLOCK_REP", serie_hashes);
10 SendPacket(packet, addr);
```

Algorithm A.7: Treatment of the follower's childless blocks list by the leader node.

```
   /* Upon reception of the CHILDLESSBLOCK_REP packet                    */
 1 if this.node == "leader" then
 2 |  v_responses += packet.GetNodeId();
 3 |  senderAddr ← packet.GetIPv4();
 4 |  serie_hashes ← packet.GetPayload();
 5 |  for hash : serie_hashes do
 6 |  |  ChildlessBlockTreatment(hash, senderAddr);
 7 |  end
 8 |  if allFollowers ∈ v_responses then
   |  |  /* All new followers responded to the leader's request.
   |  |     The merge block can be created now.                          */
 9 |  |  CreateMergeBlock();
10 |  else
   |  |  /* Waiting for the followers reply.  After a certain time,
   |  |     the Merge Block is created if a majority of nodes has
   |  |     replied.                                                     */
11 |  end
12 else
13 |  return;
14 end
15 Function ChildlessBlockTreatment(string block_hash, Ipv4Addr addr)
   /* Get a list of the childless block hashes                           */
16 myChildless ← blockgraph.GetChildlessBlockList() ;
17 if block_hash ∈ myChildless then
   |  /* Both nodes have the same childless block                        */
18 |  return;
19 else
20 |  if block_hash ∈ blockgraph then
   |  |  /* The childless block is in the leader's Blockgraph           */
21 |  |  return;
22 |  else
23 |  |  if block_hash ∈ tmp_space then
   |  |  |  /* The childless block is in the leader's memory           */
24 |  |  |  return;
25 |  |  else
   |  |  |  /* The childless block comes from another network
   |  |  |     partition                                                */
26 |  |  |  if block_hash ∈ mapping_table then
   |  |  |  |  /* The childless block has already been mapped         */
27 |  |  |  |  return;
28 |  |  |  else
29 |  |  |  |  mapping_table.add(block_hash, addr);
30 |  |  |  end
31 |  |  end
32 |  end
33 end
```

Algorithm A.8: Creates the merge block by the leader node.

```
1  Function CreateMergeBlock()
2  Block block;
3  TransactionContainer t_Container;
4  block.SetLeader(this → node.GetID());
5  block.SetGroupId(groupId);
6  block_references ← blockgraph.GetChildlessBlockList() ;
   /* The leader node add its own childless blocks with its own ID
      as a transaction.                                              */
7  for block_hash : block_references do
8  |  Transaction transaction;
9  |  string payload_t = this → node.GetID() + "   " + block_hash;
10 |  transaction.SetPayload(payload_t);
11 |  transaction.SetTimestammp(Clock());
12 |  t_container.add(transaction);
13 end
   /* The leader node adds the information gathered in the mapping
      table as transactions.                                         */
14 for block_hash : mapping_table do
15 |  Transaction transaction;
16 |  string payload_t = block_hash.first + "   " + block_hash.second;
17 |  transaction.SetPayload(payload_t);
18 |  transaction.SetTimestamp(Clock());
19 |  t_container.add(transaction);
20 end
   /* The leader add the references of the different childless
      blocks in the new block.                                       */
21 parents = [];
22 for element : mapping_table.first do
23 |  if element ∉ parents then
24 |  |  parents.add(element);
25 |  end
26 end
27 block.SetTransaction(t_container);
28 block.SetReferences(parents);
29 block.SetBlockType(Merge);
30 block.SetIndex(blockgraph.GetLastIndex + 1);
31 block.SetTimestamp(Clock());
32 SendBlockToConsensus(block);
```

Algorithm A.9: Creation of a BranchRequest message from a node.

```
1  Function BranchRequest()
   /* A BRANCH_REQUEST packet is only sent to nodes holding
      childless block not present in the current node.            */
2  serie_hash ← "";
3  for block_hash : mapping_table do
4  |  if block_hash ∉ blockgraph.GetChildlessBlockList() then
5  |  |  serie_hash+ = block_hash.GetHash();
6  |  |  ApplicationPacket packet("BRANCH_REQUEST", serie_hash);
7  |  |  SendPacket(packet, block_hash.GetIpv4());
8  |  end
9  end
```

Algorithm A.10: Reply of a BRANCH_REQUEST message.
It sends a branch of blocks to the requester.

```
   /* Upon reception of a BRANCH_REQUEST packet                   */
1  senderAddr ← packet.GetIPv4();
2  serie_hashes ← packet.GetPayload();
3  BranchReply(serie_hashes, senderAddr);
4  Function BranchReply(string serie_hashes, Ipv4Addr addr)
   /* Obtaining the GroupId of the childless block hashes         */
5  list_groupId = [];
6  for block_hash : serie_hashes do
7  |  groupId_block ← blockgraph.GetGroupId(block_hash);
8  |  if groupId_block ∉ list_groupId then
9  |  |  list_groupId.add(groupId_block);
10 |  end
11 end
   /* Sending all blocks with the same GroupId to the requester   */
12 for group_id : list_groupId do
13 |  VectorBlock  v_blocks ← blockgraph.GetBlocksGroup(group_id);
14 |  for block : v_blocks do
15 |  |  ApplicationPacket packet("SEND_BLOCK", block);
16 |  |  SendPacket(packet, addr);
17 |  end
18 end
```

Appendix B

List of Publications

B.1 International Conferences

- D. Cordova, A. Laube, T. -M. -T. Nguyen and G. Pujolle, "Blockgraph: A blockchain for mobile ad hoc networks," 2020 4th Cyber Security in Networking Conference (CSNet), 2020, pp. 1-8, doi: 10.1109/CSNet50428.2020.9265532.

- D. C. Morales, P. B. Velloso, A. Laube, T. -M. -T. Nguyen and G. Pujolle, "C4M: A Partition-Robust Consensus Algorithm for Blockgraph in Mesh Network," 2021 5th Cyber Security in Networking Conference (CSNet), 2021, pp. 82-89, doi: 10.1109/CSNet52717.2021.9614651.

- David Cordova Morales, Pedro Velloso, Alexandre Guerre, Thi-Mai-Trang Nguyen, Guy Pujolle, Khaldoun Alagha, and Guillaume Dua. 2021. Blockgraph proof-of-concept. In Proceedings of the SIGCOMM '21 Poster and Demo Sessions (SIGCOMM '21). Association for Computing Machinery, New York, NY, USA, 82–84. https://doi.org/10.1145/3472716.3472866

- D. A. Cordova M, T. -M. -T. Nguyen, P. B. Velloso and G. Pujolle, "A Preliminary Assessment of Blockgraph - a Mobility-Aware Solution to Secure 6G Mesh Networks," 2022 1st International Conference on 6G Networking (6GNet), 2022, pp. 1-4, doi: 10.1109/6GNet54646.2022.9830495.

B.2 In Process of Publication

- D. Cordova, P. B. Velloso, A. Laube, T. -M. -T. Nguyen and G. Pujolle, "A performance evaluation of C4M consensus Algorithm". Accepted in Annals of Telecommunications Journal.

Bibliography

[1] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *https://bitcoin.org/bitcoin.pdf*, 2008.

[2] Vitalik Buterin et al. Ethereum white paper (2013). *URL https://github.com/ethereum/wiki/wiki/White-Paper*, 2013.

[3] Faten Adel Alabdulwahhab. Web 3.0: The decentralized web blockchain networks and protocol innovation. In *2018 1st International Conference on Computer Applications & Information Security (ICCAIS)*, pages 1–4, 2018.

[4] Marsh McLennan. The global risks report 2022 17th edition. World Economic Forum Cologny, Switzerland, 2022.

[5] Justin Sunny, Naveen Undralla, and V. Madhusudanan Pillai. Supply chain transparency through blockchain-based traceability: An overview with demonstration. *Computers & Industrial Engineering*, 150:106895, 2020.

[6] Filipe Calvão and Matthew Archer. Digital extraction: Blockchain traceability in mineral supply chains. *Political Geography*, 87:102381, 2021.

[7] PengCheng Wei, Dahu Wang, Yu Zhao, Sumarga Kumar Sah Tyagi, and Neeraj Kumar. Blockchain data-based cloud data integrity protection mechanism. *Future Generation Computer Systems*, 102:902–911, 2020.

[8] Claudia Pop, Marcel Antal, Tudor Cioara, Ionut Anghel, David Sera, Ioan Salomie, Giuseppe Raveduto, Denisa Ziu, Vincenzo Croce, and Massimo Bertoncini. Blockchain-based scalable and tamper-evident solution for registering energy data. *Sensors*, 19(14), 2019.

[9] Anton Hasselgren, Katina Kralevska, Danilo Gligoroski, Sindre A. Pedersen, and Arild Faxvaag. Blockchain in healthcare and health sciences—a scoping review. *International Journal of Medical Informatics*, 134:104040, 2020.

[10] Fahim Ullah and Fadi Al-Turjman. A conceptual framework for blockchain smart contract adoption to manage real estate deals in smart cities. *Neural Computing and Applications*, pages 1–22, 2021.

[11] Qin Wang, Xinqi Zhu, Yiyang Ni, Li Gu, and Hongbo Zhu. Blockchain for the iot and industrial iot: A review. *Internet of Things*, 10:100081, 2020. Special Issue of the Elsevier IoT Journal on Blockchain Applications in IoT Environments.

[12] Evrim Tan, Stanislav Mahula, and Joep Crompvoets. Blockchain governance in the public sector: A conceptual framework for public management. *Government Information Quarterly*, 39(1):101625, 2022.

[13] Omar Ali, Mustafa Ally, Clutterbuck, and Yogesh Dwivedi. The state of play of blockchain technology in the financial services sector: A systematic literature review. *International Journal of Information Management*, 54:102199, 2020.

[14] Raja Wasim Ahmad, Haya Hasan, Ibrar Yaqoob, Khaled Salah, Raja Jayaraman, and Mohammed Omar. Blockchain for aerospace and defense: Opportunities and open research challenges. *Computers & Industrial Engineering*, 151:106982, 2021.

[15] Peter Howson. Building trust and equity in marine conservation and fisheries supply chain management with blockchain. *Marine Policy*, 115:103873, 2020.

[16] Alexandre Laube, Steven Martin, and Khaldoun Al Agha. A solution to the split & merge problem for blockchain-based applications in ad hoc networks. In *2019 8th International Conference on Performance Evaluation and Modeling in Wired and Wireless Networks (PEMWN)*, pages 1–6, 2019.

[17] Michel Kadoch. Recent advances in mobile ad hoc networks, 2021.

[18] N Saranya, K Geetha, and C Rajan. Data replication in mobile edge computing systems to reduce latency in internet of things. *Wireless Personal Communications*, 112(4):2643–2662, 2020.

[19] Diego Ongaro and John Ousterhout. In search of an understandable consensus algorithm. In *2014 USENIX Annual Technical Conference (Usenix ATC 14)*, pages 305–319, 2014.

[20] Thomas Clausen and Philippe Jacquet. Rfc3626: Optimized link state routing protocol (olsr), 2003.

[21] David Cordova, Alexandre Laube, Thi-Mai-Trang Nguyen, and Guy Pujolle. Blockgraph: A Blockchain for Mobile Ad hoc Networks. In *2020 4th Cyber Security in Networking Conference (CSNet)*, pages 1–8, 2020.

[22] David Cordova, Pedro B. Velloso, Alexandre Laube, Thi-Mai-Trang Nguyen, and Guy Pujolle. C4M: A Partition-Robust Consensus Algorithm for Blockgraph in Mesh Network. In *2021 5th Cyber Security in Networking Conference (CSNet)*, pages 82–89, 2021.

[23] David Cordova Morales, Pedro Velloso, Alexandre Guerre, Thi-Mai-Trang Nguyen, Guy Pujolle, Khaldoun Alagha, and Guillaume Dua. Blockgraph Proof-of-Concept. In *Proceedings of the SIGCOMM '21 Poster and Demo Sessions*, SIGCOMM '21, page 82–84. Association for Computing Machinery, New York, NY, USA, 2021.

[24] David A. Cordova M, Thi-Mai-Trang Nguyen, Pedro B. Velloso, and Guy Pujolle. A preliminary assessment of blockgraph - a mobility-aware solution to secure 6g mesh networks. In *2022 1st International Conference on 6G Networking (6GNet)*, pages 1–4, 2022.

[25] Saifedean Ammous. *The fiat standard*. Saif House, 2021.

[26] Anwar Hasan Abdullah Othman, Syed Musa Alhabshi, and Razali Haron. Cryptocurrencies, fiat money or gold standard: An empirical evidence from volatility structure analysis using news impact curve. *International Journal of Monetary Economics and Finance*, 12(2):75–97, 2019.

[27] Anwar Hasan Abdullah Othman, Syed Musa Alhabshi, Salina Kassim, Adam Abdullah, and Razali Haron. The impact of monetary systems on income inequity and wealth distribution: a case study of cryptocurrencies, fiat money and gold standard. *International Journal of Emerging Markets*, 2020.

[28] Arthur J Rolnick and Warren E Weber. Money, inflation, and output under fiat and commodity standards. *Journal of Political Economy*, 105(6):1308–1321, 1997.

[29] Stuart Haber and W Scott Stornetta. How to time-stamp a digital document. In *Conference on the Theory and Application of Cryptography*, pages 437–455. Springer, 1990.

[30] Dave Bayer, Stuart Haber, and W Scott Stornetta. Improving the efficiency and reliability of digital time-stamping. In *Sequences Ii*, pages 329–334. Springer, 1993.

[31] Ralph Charles Merkle. *Secrecy, authentication, and public key systems.* Stanford university, 1979.

[32] Cynthia Dwork and Moni Naor. Pricing via processing or combatting junk mail. In *Annual international cryptology conference*, pages 139–147. Springer, 1992.

[33] Adam Back et al. Hashcash-a denial of service counter-measure. 2002.

[34] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, pages 382–401, July 1982.

[35] Miguel Castro, Barbara Liskov, et al. Practical byzantine fault tolerance. In *OsDI*, volume 99, pages 173–186, 1999.

[36] Sina Rafati Niya and Burkhard Stiller. Bazo: A proof-of-stake (pos) based blockchain. *IFI-TecReport No. 2019.03, Zürich, Switzerland, Tech. Rep.*, 2019.

[37] Fabian Schuh and Daniel Larimer. Bitshares 2.0: general overview. *accessed June-2017.[Online]. Available: http://docs. bitshares. org/downloads/bitshares-general. pdf*, 2017.

[38] Jae Kwon and Ethan Buchman. Cosmos whitepaper. *A Netw. Distrib. Ledgers*, 2019.

[39] Sunny King and Scott Nadal. Ppcoin: Peer-to-peer crypto-currency with proof-of-stake. *self-published paper, August*, 19(1), 2012.

[40] Nxt Community. Nxt whitepaper. 2014.

[41] Proof-of-Authority Chains, howpublished = https://openethereum.github.io/proof-of-authority-chains, note = Accessed: 2022-07-01.

[42] Stefano De Angelis, Leonardo Aniello, Roberto Baldoni, Federico Lombardi, Andrea Margheri, and Vladimiro Sassone. Pbft vs proof-of-authority: Applying the cap theorem to permissioned blockchain. 2018.

[43] Intel: Sawtooth Lake(2017). https://intelledger.github.io/. Accessed: 2022-07-01.

[44] Victor Costan and Srinivas Devadas. Intel sgx explained. Cryptology ePrint Archive, Paper 2016/086, 2016. https://eprint.iacr.org/2016/086.

[45] Lin Chen, Lei Xu, Nolan Shah, Zhimin Gao, Yang Lu, and Weidong Shi. On security analysis of proof-of-elapsed-time (poet). In *International Symposium on Stabilization, Safety, and Security of Distributed Systems*, pages 282–297. Springer, 2017.

[46] Aniruddh Rao Kabbinale, Emmanouil Dimogerontakis, Mennan Selimi, Anwaar Ali, Leandro Navarro, Arjuna Sathiaseelan, and Jon Crowcroft. Blockchain for economically sustainable wireless mesh networks. *Concurrency and Computation: Practice and Experience*, 32(12):e5349, 2020.

[47] Lorenzo Ghiro, Leonardo Maccari, and Renato Lo Cigno. Proof of networking: Can blockchains boost the next generation of distributed networks? In *2018 14th Annual Conference on Wireless On-demand Network Systems and Services (WONS)*, pages 29–32. IEEE, 2018.

[48] Caciano Machado and Carla Merkle Westphall. Blockchain incentivized data forwarding in manets: Strategies and challenges. *Ad Hoc Networks*, 110:102321, 2021.

[49] Magnus Skjegstad, Anil Madhavapeddy, and Jon Crowcroft. Kadupul: Livin'on the edge with virtual currencies and time-locked puzzles. In *Proceedings of the 2015 Workshop on Do-it-yourself Networking: an Interdisciplinary Approach*, pages 21–26, 2015.

[50] Jehan Tremback, Justin Kilpatrick, Deborah Simpier, and Ben Wang. Althea whitepaper. *Tech. Rep.*, 2019.

[51] Jason Ernst, Zehua David Wang, Saju Abraham, John Lyotier, Chris Jensen, Melissa Quinn, and Dana Harvey. A decentralized mobile mesh networking platform powered by blockchain technology and tokenization. 2018.

[52] Chandrima Chakrabarti and Souvik Basu. A blockchain based incentive scheme for post disaster opportunistic communication over dtn. In *Proceedings of the 20th International Conference on Distributed Computing and Networking*, pages 385–388, 2019.

[53] Xin Jiang, Mingzhe Liu, Chen Yang, Yanhua Liu, Ruili Wang, et al. A blockchain-based authentication protocol for wlan mesh security access. *Comput. Mater. Continua*, 58(1):45–59, 2019.

[54] Atharv Chandratre and Yash Chaturvedi. Blockchain based raspberry pi mesh network. *Available at SSRN 3800557*, 2020.

[55] May Thura Lwin, Jinhyuk Yim, and Young-Bae Ko. Blockchain-based lightweight trust management in mobile ad-hoc networks. *Sensors*, 20(3):698, 2020.

[56] Nadav Schweitzer, Ariel Stulman, Asaf Shabtai, and Roy David Margalit. Mitigating denial of service attacks in olsr protocol using fictitious nodes. *IEEE Transactions on Mobile Computing*, 15(1):163–172, 2015.

[57] Hector Garcia-Molina. Elections in a distributed computing system. *IEEE transactions on Computers*, 31(01):48–59, 1982.

[58] Yongxin Liu, Jian Wang, Houbing Song, Jianqiang Li, and Jiawei Yuan. Blockchain-based secure routing strategy for airborne mesh networks. In *2019 IEEE International Conference on Industrial Internet (ICII)*, pages 56–61. IEEE, 2019.

[59] Frederic Amiel, Karine Villegas, Benoit Feix, and Louis Marcel. Passive and active combined attacks: Combining fault attacks and side channel analysis. In *Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC 2007)*, pages 92–102. IEEE, 2007.

[60] Conglin Ran, Shuailing Yan, Liang Huang, and Lei Zhang. An improved aodv routing security algorithm based on blockchain technology in ad hoc network. *EURASIP Journal on Wireless Communications and Networking*, 2021(1):1–16, 2021.

[61] Sonia Alice George, Arunita Jaekel, and Ikjot Saini. Secure identity management framework for vehicular ad-hoc network using blockchain. In *2020 IEEE Symposium on Computers and Communications (ISCC)*, pages 1–6. IEEE, 2020.

[62] Qi Feng, Debiao He, Sherali Zeadally, and Kaitai Liang. Bpas: Blockchain-assisted privacy-preserving authentication system for vehicular ad hoc networks. *IEEE Transactions on Industrial Informatics*, 16(6):4146–4155, 2019.

[63] Benjamin Leiding, Parisa Memarmoshrefi, and Dieter Hogrefe. Self-managed and blockchain-based vehicular ad-hoc networks. In *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct*, pages 137–140, 2016.

[64] AFM Akhter, Mohiuddin Ahmed, AFM Shah, Adnan Anwar, ASM Kayes, and Ahmet Zengin. A blockchain-based authentication protocol for cooperative vehicular ad hoc network. *Sensors*, 21(4):1273, 2021.

[65] Sandeep Kumar Arora, Gulshan Kumar, and Tai-hoon Kim. Blockchain based trust model using tendermint in vehicular adhoc networks. *Applied Sciences*, 11(5):1998, 2021.

[66] Chao Lin, Debiao He, Xinyi Huang, Neeraj Kumar, and Kim-Kwang Raymond Choo. Bcppa: A blockchain-based conditional privacy-preserving authentication protocol for vehicular ad hoc networks. *IEEE Transactions on Intelligent Transportation Systems*, 22(12):7408–7420, 2020.

[67] Yonatan Sompolinsky and Aviv Zohar. Secure high-rate transaction processing in bitcoin. In *International Conference on Financial Cryptography and Data Security*, pages 507–527. Springer, 2015.

[68] Yoad Lewenberg, Yonatan Sompolinsky, and Aviv Zohar. Inclusive block chain protocols. In *International Conference on Financial Cryptography and Data Security*, pages 528–547. Springer, 2015.

[69] Yonatan Sompolinsky, Yoad Lewenberg, and Aviv Zohar. Spectre: A fast and scalable cryptocurrency protocol. *Cryptology ePrint Archive*, 2016.

[70] Yonatan Sompolinsky, Shai Wyborski, and Aviv Zohar. Phantom and ghostdag: A scalable generalization of nakamoto consensus. *Cryptology ePrint Archive*, 2018.

[71] George Danezis and David Hrycyszyn. Blockmania: from block dags to consensus. *arXiv preprint arXiv:1809.01620*, 2018.

[72] Federico Matteo Benčić and Ivana Podnar Žarko. Distributed ledger technology: Blockchain compared to directed acyclic graph. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, pages 1569–1570. IEEE, 2018.

[73] Leemon Baird. The swirlds hashgraph consensus algorithm: Fair, fast, byzantine fault tolerance. *Swirlds Tech Reports SWIRLDS-TR-2016-01, Tech. Rep*, 34, 2016.

[74] Adam Gągol, Damian Leśniak, Damian Straszak, and Michał Świętek. Aleph: Efficient atomic broadcast in asynchronous networks with byzantine nodes. In *Proceedings of the 1st ACM Conference on Advances in Financial Technologies*, pages 214–228, 2019.

[75] Serguei Popov. The tangle. *White paper*, 1(3), 2018.

[76] Colin LeMahieu. Nano: A feeless distributed cryptocurrency network. *Nano [Online resource]. URL: https://nano. org/en/whitepaper (date of access: 24.03. 2018)*, 4, 2018.

[77] Anton Churyumov. Byteball: A decentralized system for storage and transfer of value. *URL https://byteball. org/Byteball. pdf*, 2016.

[78] Kolbeinn Karlsson, Weitao Jiang, Stephen Wicker, Danny Adams, Edwin Ma, Robbert van Renesse, and Hakim Weatherspoon. Vegvisir: A partition-tolerant blockchain for the internet-of-things. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, pages 1150–1158. IEEE, 2018.

[79] Joseph Poon and Thaddeus Dryja. The bitcoin lightning network: Scalable off-chain instant payments, 2016.

[80] Jaynti Kanani, S Nailwal, and A Arjun. Matic whitepaper, 2019.

[81] Demetri P Spanos, Reza Olfati-Saber, and Richard M Murray. Dynamic consensus on mobile networks. In *IFAC world congress*, pages 1–6. Citeseer, 2005.

[82] Leonidas Georgopoulos, Alireza Khadivi, and Martin Hasler. Speeding up linear consensus in networks. In *Selected Topics in Nonlinear Dynamics and Theoretical Electrical Engineering*, pages 389–405. Springer, 2013.

[83] Zhongkui Li, Guanghui Wen, Zhisheng Duan, and Wei Ren. Designing fully distributed consensus protocols for linear multi-agent systems with directed graphs. *IEEE Transactions on Automatic Control*, 60(4):1152–1157, 2014.

[84] Martin Kenyeres and Jozef Kenyeres. Average consensus over mobile wireless sensor networks: weight matrix guaranteeing convergence without reconfiguration of edge weights. *Sensors*, 20(13):3677, 2020.

[85] Paolo Braca, Stefano Marano, and Vincenzo Matta. Enforcing consensus while monitoring the environment in wireless sensor networks. *IEEE Transactions on Signal Processing*, 56(7):3375–3380, 2008.

[86] Leonidas Georgopoulos and Martin Hasler. Training distributed neural networks by consensus. Technical report, 2011.

[87] Dana Angluin, Michael J Fischer, and Hong Jiang. Stabilizing consensus in mobile networks. In *International Conference on Distributed Computing in Sensor Systems*, pages 37–50. Springer, 2006.

[88] Wanlu Sun, Erik G Ström, Fredrik Brännström, and Mohammad Reza Gholami. Random broadcast based distributed consensus clock synchronization for mobile networks. *IEEE Transactions on Wireless Communications*, 14(6):3378–3389, 2015.

[89] Prasanna Padmanabhan and Le Gruenwald. Managing data replication in mobile ad-hoc network databases. In *2006 International Conference on Collaborative Computing: Networking, Applications and Worksharing*, pages 1–10. IEEE, 2006.

[90] Aekyung Moon and Haengrae Cho. Energy efficient replication extended database state machine in mobile ad hoc network. In *IADIS International Conference on Applied Computing*, volume 224228, 2004.

[91] Leslie Lamport. Paxos made simple. *ACM SIGACT News (Distributed Computing Column) 32, 4 (Whole Number 121, December 2001)*, pages 51–58, 2001.

[92] Michel Kadoch. Recent advances in mobile ad hoc networks. *Electronics*, 10(12), 2021.

[93] Diaa Eldein Mustafa Ahmed and Othman O Khalifa. An overview of manets: applications, characteristics, challenges and recent issues. 2017.

[94] Green Communication Technology, howpublished = https://www.green-communications.fr/technology/, note = Accessed: 2022-07-21.

[95] Tomasz Ciszkowski and Zbigniew Kotulski. Distributed reputation management in collaborative environment of anonymous manets. In *EUROCON 2007 - The International Conference on "Computer as a Tool"*, pages 1028–1033, 2007.

[96] Pino Caballero-Gil and Candelaria HernÁndez-Goya. Zero-knowledge hierarchical authentication in manets. *IEICE transactions on information and systems*, 89(3):1288–1289, 2006.

[97] Eric Brewer. Cap twelve years later: How the" rules" have changed. *Computer*, 45(2):23–29, 2012.

[98] Robert Tarjan. Depth-first search and linear graph algorithms. *SIAM journal on computing*, 1(2):146–160, 1972.

[99] Edward F Moore. The shortest path through a maze. In *Proc. Int. Symp. Switching Theory, 1959*, pages 285–292, 1959.

[100] P Jacquet, P Muhlethaler, T Clausen, A Laouiti, A Qayyum, and L Viennot. Optimized link state routing protocol (olsr), 2003.

[101] Charles E Perkins and Pravin Bhagwat. Highly dynamic destination-sequenced distance-vector routing (dsdv) for mobile computers. *ACM SIGCOMM computer communication review*, 24(4):234–244, 1994.