HENRIQUE JOSÉ PENIDES CAMPOS FERREIRA
BSc in Informatics Engineering

HYFLEXCHAIN: A PERMISSIONLESS DECENTRALIZED LEDGER WITH HYBRID AND FLEXIBLE CONSENSUS PLANE

Dissertation Plan
MASTER IN COMPUTER SCIENCE

NOVA University Lisbon
February, 2023

NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE

HYFLEXCHAIN: A PERMISSIONLESS DECENTRALIZED LEDGER WITH HYBRID AND FLEXIBLE CONSENSUS PLANE

HENRIQUE JOSÉ PENIDES CAMPOS FERREIRA

BSc in Informatics Engineering

Adviser: Henrique Domingos
Associate Professor, NOVA University - Nova School of Sciences and Engineering

Dissertation Plan
MASTER IN COMPUTER SCIENCE

NOVA University Lisbon
February, 2023

Abstract

To address the tradeoffs involving scale, security and performance, together with responsiveness criteria for block-finality, new permissionless hybrid consensus models were proposed for scalable decentralized ledgers and permissionless blockchains. The hybrid consensus' core idea is based on bootstrapping efficient, scalable, and secure permissionless consensus for daily-base, combining slow or inefficient PoW (or Nakamoto inspired consensus protocols) mechanism with on-top efficient permissioned consensus, performed by dynamically formed fair committees operating under sybil-resistant sortition models. Hybrid consensus planes can use PoW consensus as a base blockchain layer, not to agree on transactions, but to agree on the sortition of committee members. Committees, in turn, can execute forms of permissioned-oriented consensus protocols for a daily-ledger blockchain layer.

In existent Hybrid Consensus approaches, the hybridization of the consensus plane is targeted to implement a specific form of sybil-resistant committee selection for a specific consensus model. In general, it is optimized for specific targeted cryptocurrency applications, not allowing other potential and diverse applications to express ways for flexible adoption of different consensus strategies, used as alternatives or used in combination.

In this dissertation we will propose the design, implementation and experimental evaluation of HyFlexChain: a permissionless ledger architecture supporting an hybrid and flexible consensus plane for unknown participants and operating with anti-sybil-resistance. For the expected solution the idea is to address a modular hybrid consensus service plane that can be leveraged by a base-blockchain architecture. HyFlexChain will expose APIs to support transactions with on-chain smart-contracts, with the necessary expressiveness for the possible alternative or combined use of different consensus mechanisms that will be provided by the designed runtime consensus service plane.

Keywords: Decentralized Ledgers, Consistent and Replicated Ledgers, Permissionless Blockchains, Parallel Blockchains, Consensus models, Hybrid and Flexible Consensus Plane, Sybil Resistance, Flexible Consensus with Unknown Participants, Smart Contracts, Security, Dependability

Resumo

De forma a abordar os *tradeoffs* relacionados com escala, segurança e desempenho, assim como a capacidade de resposta do sistema na finalização de blocos, foram propostas soluções baseadas em modelos de consenso híbrido para registos distribuídos com grande capacidade de escalabilidade. A ideia principal do Consenso Híbrido é construir um consenso eficiente, escalável e seguro em que é composto por uma combinação de dois tipos de consensos, um mecanismo lento e ineficiente (*PoW* ou outros protocolos de consenso inspirados em Nakamoto) com um consenso eficiente normalmente aplicado em sistemas privados, o qual é executado com base em comités formados dinamicamente com características resistentes a *sybil attacks*. Deste modo, planos de consenso híbrido podem utilizar *PoW* como base para eleger comités. Por sua vez, os comités são utilizados para executar modelos de consenso com o objetivo principal de ordenar transações.

Nas abordagens atuais de Consenso Híbrido, o plano de consenso é direcionado para implementar uma forma específica de eleição de um comité para um modelo de consenso específico. Em geral, este é otimizado para aplicações específicas de criptomoedas, o qual não permite outro tipo de aplicações diversificadas que expressem outras formas de consenso, utilizadas como alternativas ou usadas em combinação.

Nesta dissertação propomos o desenho, implementação e avaliação experimental da plataforma HyFlexChain: uma arquitetura baseada em registos públicos que suporta um plano de consenso híbrido e flexível para participantes anónimos, o qual executa com resistência a *sybil attackes*. Para a nossa solução esperada, a ideia é construir um plano de serviço de consenso híbrido modular baseado numa arquitetura base de *blockchain*. HyFlexChain irá expor *APIs* para suportar transações com *on-chain smart-contracts*, os quais necessitarão de expressividade necessária para a possibilidade de alternar entre os modelos de consenso ou um uso combinado dos mesmos que serão disponibilizados pelo plano de serviço de consenso projetado.

Palavras-chave: Registos Descentralizados, Registos consistentes e replicados, *Blockchains* públicas, *Blockchains* paralelas, Modelos de consenso, Plano de consenso Híbrido e Flexível, Resistência a ataques de *Sybil*, Consenso Flexível com participantes anónimos, *Smart*

Contracts, Segurança, Confiável

Contents

List of Figures vii

List of Tables viii

Acronyms ix

1 Introduction 1

1.1 Context . . . 1

1.2 Motivation . . . 3

1.3 Problem Statement . . . 3

1.4 Objectives and Contributions . . . 4

1.5 Report Outline . . . 4

2 Background 5

2.1 Decentralized Ledgers . . . 5

2.2 Blockchain Models as Decentralized Ledgers . . . 6

2.3 Smart Contracts . . . 9

2.4 Architecture and Service Planes for Permissionless Blockchains . . . 10

2.4.1 Timing, Fault Models and Scalability Issues . . . 12

2.4.2 Consensus Planes and Mechanisms . . . 12

2.5 Summary . . . 14

3 Related Work 15

3.1 Decentralized Ledgers, Scale and Consensus Planes . . . 15

3.1.1 Consensus Planes in Permissionless Blockchains . . . 15

3.1.2 Scalability Approaches in Permissionless Blockchains . . . 16

3.2 Decentralized Ledgers and Data Structures . . . 17

3.2.1 Chained Structures . . . 17

3.2.2 DAG Models . . . 19

3.2.3 Sharding, Multichains and Parallel chains . . . 20

3.3 Adversary Model, Consensus Types and Consistency Guarantees 21
3.3.1 Adversary Model 21
3.3.2 Intermediate Consensus 22
3.4 Sybil Resistant Elected Committees 23
3.5 Consensus for Permissionless Model under Unknown Participants . . . 25
3.6 Critical Analysis 26

4 Approach to Elaboration Phase **27**
4.1 HyFlexChain System Model and Architecture 27
4.2 Design Approach for System Components 29
4.2.1 Application Plane Components 29
4.2.2 Hybrid Consensus Plane 30
4.2.3 Flexible Consensus with Smart Contracts 31
4.2.4 Scale-In Criteria and Open Challenges 32
4.3 Design, Prototyping and Implementation Options 32
4.3.1 Algorand Leveraged Solution 32
4.3.2 Blockmess Inspired Solution 33
4.4 Validation and Experimental Evaluation 34
4.5 Elaboration Work Plan 35

Bibliography **36**

Annexes

I Elaboration Work Plan Activities **44**

List of Figures

2.1 Blockchain Models Framework 6
2.2 Generic Blockchain Architecture 10

3.1 Bitcoin [56] Block structure 18
3.2 Blockchain fork (from [15]) 18
3.3 Blockchain fork in Bitcoin-NG [33] 19
3.4 A direct knowledge connectivity graph containing a sink 25

4.1 HyFlexChain Architecture 28
4.2 Hybrid Consensus Plane 30
4.3 HyFlexChain Algorand Ledger 33
4.4 HyFlexChain Blockmess Ledger 34
4.5 Planned activities and tasks of the elaboration work plan 35

List of Tables

2.1 Different Blockchain Models and Application Domains 9
2.2 Different characteristics of Smart Contracts 10

3.1 Comparison of different consensus mechanisms in blockchain Platforms . . 16
3.2 Scalability approaches for different permissionless blockchains 17

Acronyms

AVM	Algorand Virtual Machine *(p. 36)*
BFT	Byzantine Fault Tolerant *(pp. 15, 16, 22, 25)*
CPU	Central Processing Unit *(p. 13)*
CUP	Consensus with Unknown Participants *(pp. 7, 15, 24–26)*
DAG	Direct Acyclic Graph *(pp. 16, 17, 19, 20, 26)*
DL	Distributed Ledger *(pp. 1, 2, 5, 7, 8, 14, 15, 17, 19, 21–24, 26, 28, 34, 35)*
DPoS	Delegated Proof-of-Stake *(p. 13)*
EVM	Ethereum Virtual Machine *(p. 39)*
IC	Intermediate Consensus *(pp. 22, 23, 26)*
JVM	Java Virtual Machine *(p. 40)*
LPoS	Leased Proof-of-Stake *(p. 13)*
LVI	Ledger-View Interface *(pp. 27, 30)*
P2P	Peer-to-Peer *(pp. 11, 27, 29)*
PBFT	Practical Byzantine Fault Tolerance *(pp. 2, 11, 14, 16, 23–26, 28–33, 44)*
PoET	Proof-of-Elapsed-Time *(pp. 2, 11, 13, 22, 26, 28, 30, 31, 45)*
PoS	Proof-of-Stake *(pp. 2, 11, 13, 16, 23, 24, 26, 28–32, 45)*
PoW	Proof-of-Work *(pp. 2, 8, 11, 13, 15, 16, 22, 26, 28, 30–32, 44)*
PPoS	Pure Proof-of-Stake *(pp. 13, 26, 32)*
SC	Smart Contract *(pp. 9, 10)*
SCMI	Smart Contract Management Interface *(pp. 27, 29)*

SGX	Software Guard Extensions *(pp. 2, 13, 45)*
SMR	State Machine Replication *(p. 12)*
SRE	Sybil Resistant Election *(p. 23)*
TEAL	Transaction Execution Approval Language *(p. 42)*
TEE	Trust Execution Environment *(p. 13)*
TI	Transaction Interface *(pp. 27, 29)*
UTXO	Unspent Transaction Output *(pp. 22, 30)*
VRF	Verifiable Random Function *(pp. 24, 26, 31)*
ZPoS	Anonymous Proof-of-Stake *(p. 13)*

1 Introduction

1.1 Context

Nowadays, the use of Decentralized Systems is a must for applications of considerable size. A Decentralized System is a system composed by multiple machines that interact with each other. This architecture allows the development of applications that do not have a single point of failure, which means that even if a machine fails, the system can continue to operate properly, thus providing high availability. Another advantage of these systems is that it is possible to combine the resources from multiple machines and possibly reduce the waiting time required to execute operations. For the context of this dissertation we first introduce the following related aspects: context of a DL, blockchains and their data structures, consistency and consensus in DLs, permissionless DLs and security issues, consensus models and types.

Distributed Ledgers (DLs). A Distributed Ledger (DL) is a decentralized system that must maintain shared state among a large number of nodes in physically separated machines, where no trusted central authorities coordinate the process. In the context of the dissertation, it is particularly interesting to consider blockchain platforms and their challenges to achieve decentralization, consistency and security requirements while maintaining good performance in large scale settings.

Blockchains and data structures. A blockchain is a data structure composed by an ordered DL of blocks, in which a block contains a set of transactions. This architecture requires to solve the problem of achieving a common state in Distributed Systems, in the sense that every node must have the same ordered ledger. In fact, blockchains are a very active research field [35, 77] and one of its major topics is the property of achieving decentralization through the use of consensus algorithms.

Consistency and consensus for DLs. Consistency and replication requirements for DLs are strongly related to the solution for the consensus problem. Consensus solutions in the classical permissioned setting have been extensively studied in distributed systems literature [7, 16, 48, 49, 53] and are based on the fact that nodes have well-known identities.

Permissionless ledgers and security issues. In the case of permissionless ledger systems, consensus models must address the required safety and liveness guarantees for the ledger consistency, with different challenges compared with the permissioned model, namely scalability. These challenges are closely related in exploring mechanisms to optimize tradeoffs imposed by the employed consensus mechanisms, such as: throughput, latency in block-finalization, etc. In these platforms, participants are able to read, write, verify and eventually enter in the consensus protocol leading to structural challenges for building blockchains of this type because the solutions presented above cannot be applied in this context due to the following issues [59]: unauthenticated communication (all participants are anonymous), scaling (the protocol participants may be uncertain about the exact number of active players in the protocol), malicious (byzantine) behaviour from some nodes and dynamic membership (nodes can enter or exit the system at any time).

Consensus models and types. Nevertheless, Bitcoin [56] was the first application to successfully address all the issues inherent with it, namely the *Double Spending* problem which states that a digital currency cannot be reused in two transactions at the same time. In order to achieve that result, Bitcoin implements a consensus algorithm based on a Proof-of-Work (PoW) [30] approach, in which all participating nodes compete for the right of writing a block into the DL by comparing computing power. Although this solution solves the issues presented above in a relatively fair and decentralized way, there are some tradeoffs that limit the wider adoption of this system, namely throughput, consistency and lots of wasted computing power on useless hash calculations. In fact, the throughput is close to 7 transactions/sec and is much lower when comparing with the traditional payment system *Visa* that can handle a peak rate of 56 000 transactions/sec [20].

To reduce the amount of consumed energy and the time required for consensus in PoW, Proof-of-Stake (PoS) [55] introduces the concept of stake related with the right for a node to add a block to the chain. However, the tradeoffs are the lack of decentralization of the base of trust which increases the possibility of the system to be attacked.

On the other hand, Proof-of-Elapsed-Time (PoET) [18] is a consensus algorithm used by the Hyperledger Sawtooth [44] similar to PoW in the sense that it relies on the concept of electing a leader in each round to propose the next block to the distributed ledger. Comparing with PoW, this algorithm is much more efficient because it only relies on the Intel Software Guard Extensions (SGX) capability which acts as a lottery mechanism to elect the leader, thus do not need to solve a computationally intensive cryptographic puzzle. The disadvantages is that it is necessary for a machine to have an Intel processor with SGX capability.

Unlike the above algorithms, Practical Byzantine Fault Tolerance (PBFT) [16] is a fast mechanism at achieving consensus and solves all the above tradeoffs at the price of limiting scalability because it requires more than 3 times the number of byzantine nodes in the system to operate properly. PBFT consensus was initially adopted in the design of permissioned blockchains [5, 42, 71]. However, due to its advantages, some permissionless blockchains are employing this type of algorithm by using a hybrid solution [33, 58, 24, 46, 1]. We will present the above consensus models in more detail in chapter 2.

1.2 Motivation

The consensus algorithm is a core component in the architecture of blockchain applications that is responsible for all the nodes to agree on the order of blocks, thus achieving a common state. Although many types of consensus are available, the choice of only one type of consensus is a limitation for the development of applications. In fact, if a single consensus is used any drastic changes in the membership of the system can cause the current consensus to not meet the needs of it and not being suitable for solving complex scenarios due to the fact that all of the mentioned consensus mechanisms suffer from some of the problems: scale, efficiency, consistency and security.

In order to improve and reduce the previously mentioned tradeoffs and the limitation for an application to select a single specific type of consensus, it is necessary to design a self adaptive consensus mechanism that must be able to dynamically adapt and choose an appropriate consensus algorithm according to the type of blockchain, user needs and the current state and conditions of the system. With this type of mechanism, the system can ensure high efficiency and stability of the employed consensus.

Following this approach, some solutions are being researched and the current state-of-the-art presents what is called as Pluggable Consensus [62, 81]. A Pluggable Consensus allows to design an architecture that must provide a single abstract interface to the consensus layer with the purpose to serve as a base for the self adaptive consensus mechanism. However, the proposed solutions have two major limitations: (1) The choice of consensus corresponds to a static setup approach. Although there can be a set of available consensus mechanisms to choose from, the choice is performed only once (before the system starts operation) and cannot be updated later. (2) The other one relates to the fact that all attempts to devise a solution to this problem can only be applied to permissioned blockchains and so it was not needed to deal with scalability issues.

1.3 Problem Statement

This leads to find answers for the following research questions:

How can we design a pluggable consensus approach to provide an adaptive consensus service plane for a permissionless ledger that must be able to dynamically adapt and choose an appropriate consensus algorithm, improving performance without compromising transactions throughput, scalability, consistency and decentralization?

How to address a modular approach in designing an hybrid consensus service plane to allow a flexible way of choosing between different consensus mechanisms in a permissionless ledger?

How to design and implement smart contracts, to allow a flexible use by applications to support different transaction types that can use alternatively or in combination different consensus mechanisms provided by the hybrid consensus service plane?

1.4 Objectives and Contributions

To find answers for the problem-statement questions, this dissertation will study how an hybrid consensus service plane can be designed to allow for a flexible use of different consensus mechanisms, by expressing consensus guarantees and requirements, at application-level, through the use of smart contracts. For this objective we will address the design and implementation of HyFlexChain: a permissionless ledger architecture supporting an hybrid and flexible consensus plane for unknown participants and anti-sybil-resistance. In concordance with the main objective, we enumerate the relevant contributions provided in this thesis:

- Definition and specification for the HyFlexChain design model and architecture providing a hybrid and flexible consensus model offering multiple consensus mechanisms;
- Design of smart contracts and required expressiveness to support transactions processed by the HyFlexChain permissionless ledger;
- Development of the HyFlexChain prototype as a modular approach using two base blockchains: Algorand [2] and Blockmess [15];
- Validation and experimental evaluation of the HyFlexChain implementations, considering the following aspects: 1) measurements of throughput and blocks' finalization time, comparing the benefits of the approach and analysing possible drawbacks; 2) measurement of HyFlexChain reaction to shifts between consensus models expressed by the provided smart contracts; and 3) performance comparison of workloads using the different consensus models provided by the HyFlexChain Hybrid consensus solution.

1.5 Report Outline

The remaining chapters of this document are organized as follows:

- **Chapter 2** introduces some background concepts in a top-down approach related to decentralized ledgers and different models, smart contracts and architecture of those systems for the permissionless model with a brief explanation of different consensus mechanisms.
- **Chapter 3** presents relevant references that cover the different facets of related work, considering the objectives and expected contributions of this thesis.
- **Chapter 4** discusses an initial approach to the elaboration phase, including the system model and software architecture considerations, implementation guidelines and an initial criteria for the expected experimental observations to validate the solution. Lastly, we present the work plan to the elaboration phase.

2

Background

In this chapter we address some relevant issues for the background of this dissertation. We begin with an overview of a DL (section 2.1) and discussing the different models of a blockchain (section 2.2). Next, we present smart contracts (section 2.3) with a comparative analysis of their implementation by various blockchain platforms. Additionally, we present the architecture of a blockchain and the different planes that compose it, with focus on the used consensus mechanisms and their limitations/tradeoffs (section 2.4). Finally, in section 2.5 we present a summary of this chapter.

2.1 Decentralized Ledgers

A Decentralized Ledger (DL) is a Data Storage Technology in which the data is replicated and accessible across many nodes. In contrast to a centralized database, a DL fits in an architecture that does not require a central administrator to process, validate or authenticate transactions, and consequently does not have a single point of failure. By their nature, these types of systems provide high availability with the possibility to combine the resources from multiple nodes for greater storage capacity and higher concurrent access. However, due to the fact that this type of system operates on a distributed setting, it is required to solve the consensus problem so that the ledger is correctly replicated through all of the participating nodes. Moreover, all the records in the ledger are timestamped and signed with a cryptographic algorithm, providing a way to verify and audit the history of the stored data.

A blockchain is essentially a specific type of DL that implements a secure, unmodifiable and append-only log of transactions. In this initial definition the word “transaction” is used in a broad sense. For example, in Bitcoin [56], a transaction is just a transfer of cryptocurrency value between two accounts with anonymous identifiers, where any node can join the system. In other blockchains, transactions do not necessarily involve money transfers and can be used to support trades of digital assets by using smart contracts. Additionally, a blockchain can be seen as a possible organization of a distributed database, in which different nodes manage locally replicated logs of transactions and related shared state, with given guarantees of consistency, availability, and network partition tolerance. Next, we present the main characteristics of a blockchain system [77]:

- Decentralization: Every node has the ability to get any block or to append one to the chain, unlike centralized systems that are controlled by a centralized authority;
- Detrusting: Data transfer between two nodes in the network does not require mutual trust, based on the fact that all transactions are stored in blocks and the blockchain uses a hash function with a consensus mechanism to add new blocks to the chain;
- Transparency: Every node has the ability to query blocks and transactions in the system, thus the information is transparent and consistent;
- Traceable and unforgeable: The use of timestamps in each transaction allows to keep an order of transactions (traceable) and once added to the chain it cannot be tampered with, thus making it unforgeable.
- Anonymity: It is not necessary to disclose the true identity of the participant, instead public keys based on asymmetric encryption are used for anonymous identification;
- Credibility: Each node stores the complete data and the operation of adding a block is done under conditions of anonymity, thus protecting privacy of all participants and credibility of transactions.

2.2 Blockchain Models as Decentralized Ledgers

There are many types or models for blockchains that are related to the nature, characteristics, and targeted applications. A model indicates how data is accessed (access-control) and the permissions for each participant. Blockchains are in general typified in the literature in different categories and terminology, corresponding to the following models: permissioned blockchains, permissionless blockchains, consortium blockchains, private blockchains, public blockchains or hybrid blockchains.

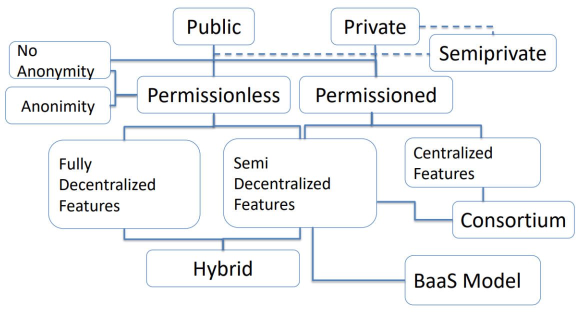

Figure 2.1: Blockchain Models Framework

In figure 2.1 is represented a framework to classify different blockchains according to the different models [34] which are presented as follows.

Public Model In the public model there is no central authority in the system, in the sense that every participant is able to read, write, verify and enter in the consensus protocol of the data on the chain. In this model, anyone can join as a node in the system without the need for being approved, which helps to achieve a self-governed and decentralized nature as permissionless ledgers or permissionless blockchains, with the facets introduced below.

Permissionless Model The permissionless category represents a fully decentralized blockchain where anyone can read the data and host a node anonymously without the need to be approved. For this purpose, these blockchains are based on open-source and transaparent code, allowing any participant to install and deploy its own network node. In this way, any user can validate transactions and decide the state of the DL that follows consistency rules based on consensus mechanisms defined by the code and runtime execution. In the context of this dissertation, we are particularly interested on permissionless blockchains, in which we emphasize the following aspects from our related work study:

- Permissionless operation: All participants are able to create and disseminate transactions where any one has equal access to validate transactions and to create, propose and validate blocks, following the same rules. For this purpose, it is required a consistent model for the verification of blocks that is related to the block finalization concept in such blockchains. Additionally, these blockchains have been the field for the support of many cryptocurrencies where block finalization requires consensus mechanisms that are related to the notion of "mining" for successful proposers;

- Block mining, finalization and incentivation: Any participant can create and propose blocks that must be verifiable by a certain number or majority of other participants, for security guarantees. In this case, blocks (and consequently their transactions) are finalized with a certain amount of cryptocurrency value being assigned to the miner as the incentive. This incentive is the way to create cryptocurrency value;

- Finalization with unknown participants: In permissionless blockchains, it is interesting to relate the block finalization with solving the consensus problem in a large-scale settings and in a network of unknown participants. In fact, these systems employ a dynamic membership approach in which participants can enter or exit the system at any time. Therefore, consensus solutions must address the required guarantees for safety and liveness conditions under the Consensus with Unknown Participants (CUP) model with an impact on tradeoffs between consistency, security and performance, including throughput and latency metrics for finality guarantees. Not surprisingly, different mechanisms used to solve consensus in a CUP model under scalable conditions and performance requirements can strongly influence the operation of the consensus plane of different blockchains and permissionless ledger models.

Permissioned Model In the permissioned model, only approved participants can join the system. The membership is typically established and defined by deployment and configurations as a management process, allowing for refined control over the blockchain. In general, there is also an access control of who has the write to finalize blocks and obtain the reward. In contrast to permissionless blockchains, the number of nodes is much more reduced and the risk of byzantine behaviour is almost negligible. This allows for choosing a more efficient and traditional consensus mechanism where there is no need for costly mining, like PoW. Consequently, the efficiency of this type of blockchain is much higher than the permissionless one, where the transaction approval can be reduced to an order of milliseconds. In fact, there are two categories that are inserted in this model, such as: **Private** and **Consortium or Federated**.

In **Private blockchains**, the access is restricted to authorized participants with the government and operation usually associated to a single entity which ensures a certain degree of trust. This type of blockchains are designed to target specific business requirements with high level of security guarantees and management control. A typical scenario for private blockchains is the support of internal processes for an organization or company, where a decentralized, trustable, and secure ledger is needed for tracking and recording data.

In addition to private systems, there are other solutions that try to extend that category to be able to integrate multiple organizations known as **Consortium or Federated systems**. In this type of systems, the operation to add a new block to the chain is determined by a set of pre-selected or leader nodes and it shares a similar scalability and privacy protection level with the private model. In this type, multiple organizations can join together and collaborate in a decentralized network with the advantage that it is more secure and eliminates the risks of having one entity controlling the network.

Hybrid Model The Hybrid Model fits in the middle of the permissionless and permissioned models in the sense that it combines elements from both types. In this model, a blockchain is set up by a single organization with an access-control to the data by the users and also with the possibility of allowing specific data to be publicly accessed. This means that transactions are not necessarily public but can be accessed for verification when it is needed, typically through a smart contract.

Summary To summarize the above blockchain models, we present in table 2.1 examples of such systems and their application domains. Regarding permissionless blockchains, there are many approaches for DLs successively addressed in the literature, as we will discuss in the related work chapter.

[1]Hybrid solutions

	Private	Consortium	Public	Hybrid	Application Domains
Permissioned	Hyperledger Fabric [5], R3-Corda [42], Quorum [61], Multichain [40], Tendermint [71]	Hyperledger Fabric [5], R3-Corda [42]	-	HybridChain [75], Hybrid-IoT [63], zkCrowd [82]	Financial Services, Logistics Supply-Chain, Healthcare Management, IoT[1], ECommerce[1]
Permissionless	-	-	Bitcoin [56], Ethereum [76], BitcoinNG [33], PeerCensus [24], Hybrid Consensus [58], Byzcoin [46], Algorand [39], Solida [1], Chainspace [11], Omniledger [47], RapidChain [80], Elastico [52], OHIE [79], Prism [9], Blockmess [15], Hasgraph [10], Blockmania [21], SPECTRE [66], IOTA [60], PHANTOM [67], Meshcash [13], GHOST [68], Conflux [51]	-	Cryptocurrency Ecosystems, P2P Electronic Transactions

Table 2.1: Different Blockchain Models and Application Domains

2.3 Smart Contracts

The term Smart Contract (SC) was first proposed in 1990 by Nick Szabo and refers to a digital transaction protocol that executes the terms of a contract [69]. In fact, this term derives from the definition of a contract that formalizes a relationship between certain parties. Thus, a SC is a piece of code that can be integrated in a blockchain with the purpose of allowing different peers to do certain distributed computations. Therefore, there is an opportunity for different peers to interact with each other in a deterministic way by leveraging from a pre-defined set of rules defined in the contract [19].

Since SCs are programs, they must follow some important properties: 1) The SC can have an associated state that references assets present on the chain; 2) It allows to express some business logic; 3) A SC should describe all possible outcomes when validating transactions, thus preventing fraud and enforcing an agreed law (imposed by the code) between two peers without the need for a trusted intermediary; 4) The code must be deterministic, i.e. the same input must produce the same output; 5) It should be triggered (executed) when it is referenced in a transaction; 6) Since SCs are deployed in the blockchain the code can be inspected by any peer; 7) All transactions are digitally signed meaning that all executions of SCs are verifiable.

Additionally, the implementation/support of SCs by a variety of blockchains tends to have different characteristics, namely in the terminology, utilized technology, type of deployment (installed, on-chain or off-chain) and programming languages used to develop these programs which can also be classified based on Turing Completeness. Altogether, we can also classify those implementations in the context of their runtime execution isolation and the expressiveness of such programs. As a result of this analysis, in table 2.2 are presented different combinations of characteristics for a variety of blockchains that support this technology.

[2]Including reconfiguration of Consensus Plane Mechanisms

					Runtime exec. and isolation		Expressiveness Support		
	Terminology/ Technology	**Deployment**	**PL**	**Turing Comp.**	**VM for Lang Runtime**	**Containerized solution**	**App. Logic**	**Tx Process and Verif.**	**Reconf. Runtime and Service Planes**[2]
Bitcoin [56, 6]	Script	on-chain	Bitcoin Script Lang.	✗	✗	✗	✗	✓	✗
Ethereum [76]	SC	on-chain	Solidity [65]	✓	EVM [32]	✗	✓	✗	✗
Hyperledger Fabric [5, 43]	chaincode	installed	Go, Javascript	✓	✗	Docker [26]	✓	✗	✗
Hyperledger Sawtooth [57, 43]	Transaction Family [72]	installed & on-chain	Java, Go, Solidty ...	✓	JVM [45] & EVM [32]	Docker [26]	✓	✗	✗
Algorand [17, 4]	ASC1 [3]	on-chain	TEAL [70]	✗	AVM [8]	✗	✗	✓	✗
	ASC2	off-chain	-	✓	✗	✗	✗	✓	✗

Table 2.2: Different characteristics of Smart Contracts

An interesting aspect of SCs is the possibility to extend their functionality to permit a dynamic reconfiguration of certain settings of a blockckchain at runtime. An example of such approach would be a contract with the ability to change the employed consensus mechanism based on a set of rules. In order to implement such a feature, the system would have to be prepared to allow a Pluggable Consensus mechanism [62, 81]. However, this functionality cannot be found in permissionless blockchains which we will address in our proposed solution in chapter 4.

2.4 Architecture and Service Planes for Permissionless Blockchains

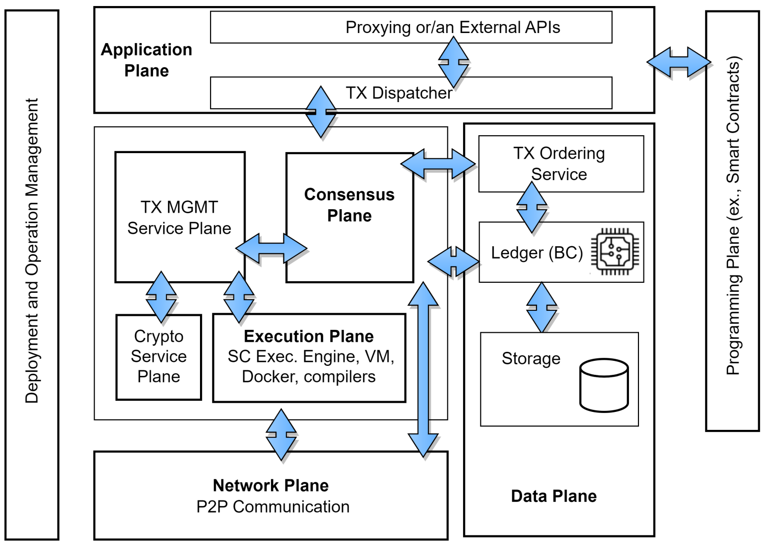

Figure 2.2: Generic Blockchain Architecture

When designing a system, it is important to separate concerns and a blockchain is no exception. In figure 2.2 is represented a modular specification of an architecture for a generic blockchain with different service planes. This specification derives from the one presented in [34] which describes the following service planes: application plane, execution plane, data plane, consensus plane and network plane. For each of these planes we present the corresponding explanation while referring to the presented specification.

Application Plane The application plane fits on top the blockchain system and supports the integration for applications that are built on top of it, mainly used by the end user. With this, blockchains must have an external API for applications to be able to interact with it. There are many types of applications that could use a blockchain and the most popular are cryptocurrencies.

Execution Plane The execution plane is responsible for executing smart contracts on the blockchain. The smart contracts may contain some business logic and be used when validating transactions (see 2.3). This plane is integrated in the *Transaction Management* plane, more specifically in the *Smart Contract Execution Engine*.

Data Plane There are many aspects when referring this plane, namely: transaction models, data structures which includes the block structure and Merkle trees, hash functions, encryption algorithms and many others (see 3.2). It is responsibility of this plane to define the way that data is stored in the ledger and also the data structures used to represent it. With this, the *Ledger* and consequently the *Storage* planes are integrated in this plane. Moreover, the hash functions (ex SHA256, etc) and encryption algorithms (RSA, ECC) are extremely important when verifying and validating transactions and also when identifying blocks, thus the *Crypto Services plane* is also integrated.

Consensus Plane The Consensus plane is responsible for ordering transactions/blocks in a way that ensures that every participant contains the same ordered ledger. This is the core plane in a blockchain that solves the common problem to Distributed Systems, namely the consensus problem. There are many consensus algorithms, each of them with different tradeoffs that can be used in different situations based on the type of blockchain and user preferences. In subsection 2.4.2 is presented the most common consensus algorithms and mechanisms used in blockchains such as: PoW, PoS, PoET and PBFT.

Network Plane The Network plane is the base plane for all these systems. It implements a Peer-to-Peer (P2P) network used for peer discovery and propagation of transactions and blocks. Depending on the type of blockchain it can have thousands of participants for permissionless or over a hundred for permissioned ones. Either way, the network serves the purpose of transmitting information for the system to be able to reach consensus and so, speed and stability are important characteristics that can affect the performance and efficiency of systems that operate on top of it. It is in this plane that different timing and fault models take place, specifically when referring to the time a message takes to be received or the type of faults the system can handle (see 2.4.1).

2.4.1 Timing, Fault Models and Scalability Issues

When we refer to decentralized systems, there are some important characteristics that must be considered. One of them is the time assumption of a system which can be of three types [14]: Asynchronous, Synchronous and Partially Synchronous. An **Asynchronous System** is a system in which there are no assumptions regarding the delay in the delivery of messages. With this, processes do not have access to any sort of physical clock and consequently there are no bounds on processing or communication delays. On the other hand, **Synchronous Systems** are exactly the opposite of the previous one in the sense that exists an upper bound for computation and communication delays. However, this type of systems do not actually exist. The last model, **Partially Synchronous** fits in the middle of the two other models in the sense that it is possible to break some timing assumptions for a limited time duration [29].

Another important concept is the fault model of a system which can be of three types [14]. The **Byzantine Fault Model** [49] states that a system must be able to defend against malicious attacks. The origin of these attacks might come from the nodes of the system, called as faulty nodes, which is a common case in permissionless blockchains, since there is no control on its participants. The other nodes that follow the protocol are called the correct nodes. Another model is the **Omission Fault Model** that represents inactive nodes. In this case, this type of nodes choose consciously and not maliciously to not participate in the system. The other nodes, the ones that contribute to the progress of the system are called the active nodes. The third model, the **Crash Fault Model** stands in the position that every node may realise that some other node has crashed and possibly do something to compensate that loss. The disadvantage of this model is that it is not possible to reliably detect when a node has failed under an asynchronous timing assumption.

Additionally, permissionless blockchains impose another challenge, namely scalability by supporting dynamic membership. With this, it is impossible for the system to expect a certain and fixed number of participants.

2.4.2 Consensus Planes and Mechanisms

As already stated, the Consensus Plane is the core of a blockchain in the sense that consensus is responsible for ensuring that all replicas agree on the same order in which operations are executed, thus achieving State Machine Replication (SMR). In the context of blockchains, the operations corresponds to transactions and all the nodes must validate and execute them in the same order, leading to a consistently replicated ledger. For this purpose, algorithms must comply with the following properties [14]: a) termination - every correct process eventually decides a value; b) validity - if a process decides v then v was proposed by some process p; c) integrity - no process decides twice; and d) agreement - no two correct processes decide differently.

Although, when applying consensus in blockchains, the underlying model corresponds to an asynchronous system and so, the FLP Impossibility emerges [36]. The

FLP Impossibility states that there is no deterministic protocol that solves the consensus problem in an asynchronous system where a single process may fail. This means that it is not possible to guarantee all properties of consensus. To circumvent this issue, implementations of consensus algorithms have been relaxed in a way that they can be used to reach consensus (most of the times), but at some cost. Next, we present the most common consensus algorithms applied in blockchains.

PoW Proof-of-Work [30] is an algorithm that reaches consensus by comparing computer power among the participants. It was initially introduced with the purpose to combat email spam via a computational effort of the sender [31] and is currently used in blockchains, namely in Bitcoin. In fact, Bitcoin uses a Central Processing Unit (CPU) bound variant in which the core idea of the algorithm is to issue a cryptographic puzzle where the only way to solve it is by generating a nonce, compute a hash and comparing if the hash is inferior with some pre-defined difficulty value. To this process of discovering the correct nonce is called mining. It is used a secured hash function which is pre-image resistant that guarantees that there is no deterministic way of solving the problem. Although this solution is completely fair and decentralized, in the sense that every node can participate, it consumes a lot of energy and has a very low throughput. Also, consistency is not guaranteed because it allows temporary inconsistencies in the ledger (forks on the chain).

PoS To mitigate the disadvantages of PoW, Proof-of-Stake [55] is introduced as a relatively fast and energy-saving consensus algorithm. In this algorithm, nodes that create and validate blocks need to have stake by depositing coins in an escrow account. After this, a group of nodes is selected to participate in the consensus (based on the amount of stake). But, in case of incorrect or malicious actions, a node (or stakeholder) can loose part or all of its stake. It is the purpose of this algorithm to incentivate good behaviour from nodes and thus, mitigating attacks on the network. However, there exists some tradeoffs like the lack of decentralization of the base of trust based on the elected participants. Also, PoS comes in different flavours, such as: Delegated Proof-of-Stake (DPoS), Leased Proof-of-Stake (LPoS), Anonymous Proof-of-Stake (ZPoS) and Pure Proof-of-Stake (PPoS).

PoET Proof-of-Elapsed-Time [18] is another approach in reaching consensus with the same benefits of PoS. In this case, it is very similar to PoW in the sense that it relies on the concept of electing a leader in each round to propose the next block to the distributed ledger. This algorithm was invented by Intel by leveraging the Trust Execution Environment (TEE), namely the SGX capability. With this algorithm, a validator executes a lottery function and waits for the specified time returned by that function. This algorithm is secured because the execution is done within the processor secure enclave and it is fair and easy for any participant to verify it. The disadvantage of this approach is that a node needs a CPU with SGX capability to participate in the algorithm.

PBFT The PBFT [16] algorithm was introduced by Miguel Castro and Barbara Liskov in 1999 and is a consensus protocol that can tolerate up to f Byzantine Faults in a system with $3f + 1$ replicas. This algorithm evolves by rounds and it starts by a client request to the primary replica. Then, the primary starts the PRE-PREPARE phase by multicasting a message to every other replica and the PREPARE and COMMIT phases follow it. Finally, the protocol terminates when the client receives $f + 1$ replies from different replicas. Although this is fast at achieving consensus and solves all the above tradeoffs, it is limited by scalability because it requires a fixed membership and any alterations to it must be decided as any other value, thus being very inefficient for permissionless blockchains.

2.5 Summary

In this chapter, we presented the background topics that cover this thesis, including the definition of a DL, different models of blockchains and an overview of the architecture of those systems while focusing on the consensus plane and the different consensus mechanisms and tradeoffs. Additionally, we analysed how different blockchains implemented smart contracts and also giving an intuition on how they could be used to permit the modification of the consensus mechanism at runtime.

3 Related Work

In this chapter we will discuss relevant references more closely related to the objectives and contributions specifically expected for our planned dissertation. These topics are organized in the following way: section 3.1 presents an analysis of the consensus plane and scalability approaches on permissionless blockchains; section 3.2 discusses chained structures, the DAG model and other techniques to increase performance; section 3.3 presents the adversary model for permissionless DLs with an analysis of different consensus types and consistency guarantees; section 3.4 relates to multiple element committees and how to perform a sybil resistant election; section 3.5 introduces the CUP model as a way to apply classical consensus mechanisms to permissionless systems with unknown participants; and in section 3.6 we perform a critical analysis of all the enumerated topics of this chapter.

3.1 Decentralized Ledgers, Scale and Consensus Planes

3.1.1 Consensus Planes in Permissionless Blockchains

As introduced before, permissioned blockchains are unsuitable for permissionless operation. First of all, they become less efficient when the number of nodes increase significantly, which corresponds to large-scale settings for cryptocurrency ecosystems, such as Bitcoin [56] or Ethereum [76]. In another relevant perspective, they reach consensus based on a voting approach, that requires to control and possibly authenticate in order to resist sybil attacks [27], namely collusion of faulty nodes.

Some recent proposals of permissionless blockchains leverage the benefits of Byzantine Fault Tolerant (BFT) algorithms while also using PoW to allow any node to be part of the system. This type of blockchains are based on hybrid consensus approaches. The main idea is that they can mix the two types of algorithms.

In table 3.1, we present a comparative analysis between classical and research oriented permissionless blockchains focused on the consensus plane characteristics and support for smart contracts. The comparison is focused on different approaches for the consensus

	Consensus Plane					
	Mechanism	**Participants**	**Resiliency**	**Committee Type**	**Committee Sybil-Resistant**	**Smart Contracts**
Bitcoin [56]	PoW	All	51%	-	-	✓
Ethereum [76]	PoS	All	51%	-	-	✓
Bitcoin NG [33]	PoW	All	51%	-	-	✓
PeerCensus [24]	PoW, PBFT	Committee	$3f+1$	Elected	✗	✗
ByzCoin [46]	PoW, PBFT	Committee	$3f+1$	Elected	✗	✗
OmniLedger [47]	PoW, PBFT	Committee	$3f+1$	Elected	✓	✗
Algorand [2, 17]	PPoS	Committee	51%	Elected	✓	✓
Solida [1]	PoW, PBFT	Committee	$3f+1$	Elected	✓	✗
Hybrid Consensus [58]	PoW, PBFT	Committee	$3f+1$	Elected	✓	✗
RapidChain [80]	PoW	All	51%	-	-	✗
Elastico [52]	PoW	All	51%	-	-	✗
OHIE [79]	PoW	All	51%	-	-	✗
Prism [9]	PoW	All	51%	-	-	✗
Blockmess [15]	PoW	All	51%	-	-	✗
Hashgraph [10]	PoW	All	51%	-	-	✗
Blockmania [21]	PoW	All	51%	-	-	✗
SPECTRE [66]	PoW	All	51%	-	-	✗
IOTA [60]	PoW	All	51%	-	-	✗
PHANTOM [67]	PoW	All	51%	-	-	✗
Meshcash [13]	PoW	All	51%	-	-	✗
GHOST [68]	PoW	All	51%	-	-	✗
Conflux [51]	PoW	All	51%	-	-	✗

Table 3.1: Comparison of different consensus mechanisms in blockchain Platforms

plane in which we explored the following characteristics: used consensus mechanism, the type of participation in the algorithm and the resiliency of the protocol.

As a result of this analysis it is possible to extract that classical permissionless blockchains tend to use a single hardcoded consensus mechanism, thus not dealing with the problem of using multiple consensus. With this approach, the performance of these systems is limited to the tradeoffs of the employed consensus mechanism which tends to be PoW and PoS. However, there are some solutions that try to mitigate these tradeoffs, Bitcoin-NG [33] is an example of a blockchain designed to scale and inspired in Bitcoin [56] with the purpose of improving the scalability and throughput of this system which is only limited by the bandwidth of peers and propagation time of the network. Another approach consists of using a classical BFT consensus mechanism like PBFT, not so common in the permissionless model due to scalability issues but instead, applied to a committee of participants [46, 47] known as a Hybrid Consensus. Following this approach, an election procedure of a committee is needed to be able to execute the consensus algorithm. Note that, this election should be periodically executed with security guarantees, namely a non-sybil election like the one performed by the Omniledger blockchain [47]. Additionally, we can also extract that systems that apply other scale-in or scale-out approaches tend to use PoW as the consensus mechanism.

3.1.2 Scalability Approaches in Permissionless Blockchains

Together with the above solutions, the table 3.2 summarizes other approaches for permissionless blockcains in addressing scalability issues, together with relevant used mechanisms. In the next sections we will discuss more about these solutions, namely Direct Acyclic Graph (DAG) based blockchains, Sharding, Multichains and Parallel Chains.

		Scale-out	**Scale-in**						
	Consensus	**State-Sharding**	**Sharded Computa-tions**	**Multiple Parallel Chaining**	**Blocks in main chain**	**DAG Blockless**	**DAG Block oriented**	**Hybrid Con-sensus**	**Flexible use of Consen-sus Plane**
Bitcoin [56]	PoW	✗	✗	✗	✓	✗	✗	✗	✗
Ethereum I [76]	PoW	✗	✗	✗	✓	✗	✗	✗	✗
Ethereum II [76]	PoS	✗	✗	✗	✓	✗	✗	✗	✗
Bitcoin NG [33]	PoW	✗	✗	✗	✓	✗	✗	✓	✗
PeerCensus [24]	PoW, PBFT	✗	✗	✗	✓	✗	✗	✓	✗
Hybrid Consensus [58]	PoW, PBFT	✗	✗	✗	✓	✗	✗	✓	✗
ByzCoin [46]	PoW, PBFT	✗	✗	✗	✓	✗	✗	✓	✗
Solida [1]	PoW, PBFT	✗	✗	✗	✓	✗	✗	✓	✗
Algorand [2, 17]	PPoS	✗	✗	✗	✓	✗	✗	✗	✗
OmniLedger [47]	PoW, PBFT	✓	✓	✗	✓	✗	✗	✗	✗
RapidChain [80]	PoW	✓	✓	✗	✓	✗	✗	✗	✗
Elastico [52]	PoW	✗	✓	✗	✓	✗	✗	✗	✗
OHIE [79]	PoW	✗	✗	✓	✗	✗	✗	✗	✗
Prism [9]	PoW	✗	✗	✓	✗	✗	✗	✗	✗
Blockmess [15]	PoW	✗	✗	✓	✗	✗	✗	✗	✗
Hashgraph [10]	PoW	✗	✗	✓	✗	✓	✗	✗	✗
Blockmania [21]	PoW	✗	✗	✓	✓	✗	✗	✗	✗
SPECTRE [66]	PoW	✗	✗	✗	✓	✗	✓	✗	✗
IOTA [60]	PoW	✗	✗	✗	✓	✓	✗	✗	✗
PHANTOM [67]	PoW	✗	✗	✗	✓	✗	✓	✗	✗
Meshcash [13]	PoW	✗	✗	✗	✓	✗	✓	✗	✗
GHOST [68]	PoW	✗	✗	✗	✓	✗	✓	✗	✗
Conflux [51]	PoW	✗	✗	✗	✓	✗	✓	✗	✗

Table 3.2: Scalability approaches for different permissionless blockchains

As a result of this analysis, we can extract that DLs that use approaches like DAGs or parallel chains have the objective to optimize throughput by separating consensus from network dissemination. In the case of DAGs, the idea is for the consensus to be executed without any external communication between nodes. However, in the case of parallel chains this approach is not always addressed, with Prism [9] being an exception.

Additionally, as presented on the table, solutions employing a Hybrid Consensus restrict the use of the supported consensus mechanisms in the sense that one of them is used for committee election and the other one is for executing the consensus of blocks. This is captured by the table in the sense that there are no solutions that employ a Hybrid and Flexible use of the consensus plane, which we will address in our approach.

3.2 Decentralized Ledgers and Data Structures

3.2.1 Chained Structures

Chained structures or blockchain is the simplest and most popular implementation of a DL. In this kind of structures, a DL is modeled as a set of blocks in which a block contains 2 main fields: ***prev*** - a reference to the previous block in the chain, generally a hash of the block; and ***data*** - represents the data stored in the block which in most blockchains, the data are transactions. In figure 3.1 is presented the structure of a block.

As mentioned in the last chapter in section 2.1, a DL needs to solve the consensus problem in order to be able to add new information to the ledger which in this case corresponds to adding a new block to the chain. However, in this kind of systems, some

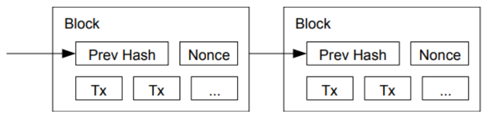

Figure 3.1: Bitcoin [56] Block structure

participants may have a different view of the system state or malicious intents which could lead to inconsistencies on the ledger, namely **forks**. As a result, this structure forms a tree in which the root corresponds to the *genesis* block, i.e. the first block on the chain and each chain (an ordered set of blocks with the first block being the *genesis* one) is assigned a **weight**. Since only one chain can contain blocks that can be finalized, the weight is used to determine the chain which has the biggest weight - **heaviest chain**. In fact, the chain weight can be measured in different ways, but the most common is called the **longest chain** rule, applied in Bitcoin [56] and consists of finding the chain that has the greater sum of weights of all the blocks that are contained in it.

In Bitcoin [56], a fork is created when two nodes accept a block and broadcast it for the same position in the ledger. In fact, the probability of a fork is proportional to the rate of proposed blocks and inversely proportional to the time required to disseminate a block [25]. As a result, if there are multiple blocks being proposed compared with their propagation time, the probability of those blocks referencing the same previous block is much higher and performance suffers greatly. This means that for the system to maintain certain safety guarantees, it is needed to lower the block creation rate which increases the confirmation time of transactions and thus, reducing the throughput of the system.

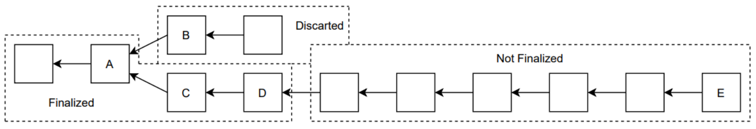

Figure 3.2: Blockchain fork (from [15])

In figure 3.2, is presented a fork on the Bitcoin blockchain after block A resulting in two chains initiated by blocks B and C which cannot be finalized. When block E is appended, blocks C and D can be finalized because the fork initiated by C is longer than the other one by 6 blocks which corresponds to the **finalization depth**. This ensures that the fork started by B cannot become the longest chain because miners will only propose blocks that they believe to be correct, i.e. blocks that belong to the heaviest chain, thus enforcing the rule of the longest chain. Since a fork can occur after block D, any block that follows D cannot be finalized because other chain could become the longest chain.

On the other hand, in Bitcoin-NG [33], the operation of the blockchain is different from Bitcoin in the sense that are introduced two kinds of blocks:

- **key blocks**: Blocks that are used to elect a leader which corresponds to the next proposer of the next batch of Micro-blocks. As in Bitcoin, these kind of blocks follow the same approach in the sense that they need to be mined in order to be appended to the chain.

- **Micro-blocks**: Blocks used to record and append transactions to the ledger by the current leader. This kind of blocks do not need to be mined as they are related to the current leader (the previous key block) in order to be valid.

Since micro-blocks do not need to be mined, the append rate of those blocks are much higher which increases the throughput of transactions. However, forks can still happen in this system and as in Bitcoin, nodes choose to mine on the chain with most work.

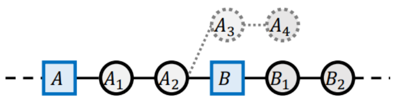

Figure 3.3: Blockchain fork in Bitcoin-NG [33]

In figure 3.3, is presented a fork in the Bitcoin-NG platform. In this example, node A mined a key block and appended two micro-blocks, A_1 and A_2. Node B mined a key block referencing A_2, thus creating a fork and enforcing a leader change. In this case, the chain initiated by a key-block from B wins and the micro-blocks A_3 and A_4 are discarded thus, resolving the fork.

3.2.2 DAG Models

Although the blockchain is the most popular implementation of a DL, there are some limitations that other implementations try to overcome, namely the throughput and scalability of the system. Based on this, alternative implementations relax properties and constraints of blockchains when referring to the block structure. This type of solutions form a DAG structure which can be of two types [74]: DAG Block oriented chains [66, 67, 13, 68, 51] and DAG-Blockless structures [10, 60].

3.2.2.1 DAG with Block oriented chains

The DAG model with Block oriented chains corresponds to a type of block graph that allows a block to reference a set of predecessor blocks. The main advantage of this structure over a blockchain (that only references a single previous block) is the possibility to incorporate the contents of all off-chain blocks (blocks that do not belong to the heaviest chain) that do not conflict with each other. With this, it is possible for nodes to validate and accept non conflicting transactions from blocks in different chains produced by a fork. To this purpose, an Inclusive Protocol is used as a method to define the set of accepted transactions. Therefore, blocks do not need to be linked in a single chain in order to be finalized thus, the block proposal rate is much higher and the negative impacts of forks can be reduced, namely the wasted resources used in mining blocks that turn to be invalid.

However, some solutions like [50, 67] are vulnerable to liveness attacks. In these cases, they perform a topological sort of blocks in order to get a total order of blocks and if the block proposal rate is high they are vulnerable to such attacks. Others, like [66] provide a weaker form of consistency in the sense that the system obtains a non-transitive partial order for any pair of blocks in the DAG. With this, systems that follow this approach are limited as it is very difficult to support smart contracts without having a total order. Despite the weaker consistency model supported by those systems, this model is strong enough to support cryptocurrencies where users can trade tokens while the system needs to verify and mitigate Double Spending attacks.

3.2.2.2 DAG with Blockless Data Structures

There are other proposes [38, 10] that try to decouple data dissemination from the consensus mechanism which allows to improve scalability conditions. This approach is based on efficient implementations of DAGs which allows to build simpler and efficient consensus mechanisms. In this type of DAG based blockless implementations, instead of grouping transactions into blocks, transactions are linked in a way that form a DAG graph. Then, every node can have the same graph structure and the consensus logic does not need to incur in a communication overhead because each node will only process its local view of the graph computing a total order without sending any messages.

3.2.3 Sharding, Multichains and Parallel chains

Sharding One of the most practical way of achieving horizontal scalability is by using a technique known as Sharding. Sharding is based on the idea of partitioning the state of the system into multiple parts, which avoids the duplication of resources by all participating nodes, such as: communication, data storage and computation. Unlike regular blockchains, in this type of solutions [78, 22, 52, 47, 80], a set of nodes is responsible for a single part (shard) of the system, thus benefiting from parallelism to achieve full performance improvements while preserving the necessary security and decentralization of the system. However, there are some issues when implementing such systems [78], such as: *intra-consensus safety* and *cross-shard-atomicity*. Since the system is divided into multiple shards with each one independent from each other, the adversary can perform a coordinate attack to a single shard instead of the entire network in which the probability of success is much higher. To mitigate this problem, the election of nodes to different shards must be a result of a randomized process with sybil resistant properties [27]. The latter problem is encountered when processing cross-shard transactions in which the state from multiple shards needs to be merged. This point of convergence is extremely important and must be secured in order to guarantee the atomicity of these transactions.

Multichains Along with Sharding, it is possible to increase the performance of DLs by supporting interoperability between platforms in order to build *cross-blockchain decentralized applications* [12]. Actually, applications that are built on top of these interoperability architectures benefit from the performance and advantages of multiple solutions. In fact, this approach promotes scalability in the sense that the workload of processing transactions can be load-balanced across different blockchains. With this, emerges the possibility to exchange assets between those systems. However, different blockchains may be incompatible with one another, therefore it is necessary to build a secure flow of assets between different networks that ensures atomicity, consistency and durability properties. As a result, the system must recognize different blockchain solutions and provide an Inter-chain Gateway Protocol that connects two systems securely.

Parallel chains Parallel Chains is an architecture composed by multiple blockchain instances that run in parallel and are independent from each other in which the aggregate contribution of those instances is more performant than using a single one. In fact, this approach is currently being researched [37] and there are many solutions that are able to increase the transaction's throughput by ensuring that each of the chains grow at the same time as a single blockchain, thus employing a *Sortition* procedure to map blocks to specific chains. As a result, the throughput can be optimized in the order of x, being x the number of parallel chains. OHIE [79] is a platform that is built upon multiple instances of the Bitcoin protocol in which blocks are linearized when delivered to the application, thus being indistinguishable from a single chain architecture. However, the finalization time of blocks is dependent on the smallest chain. Other solutions like Prism [9] try to mitigate that issue by reducing the finalization time between blocks balanced with the optimization of throughput. Lastly, Blockmess [15] is based on [37, 79] that is able to dynamically adapt the number of chains in order to increase the throughput of the system.

3.3 Adversary Model, Consensus Types and Consistency Guarantees

3.3.1 Adversary Model

When modeling a DL, just like any other Distributed System, it is important to define the Adversary Model so that the system can be designed and maintained to be able to eliminate/mitigate certain attacks. In the context of public DLs, the Adversary Model corresponds to the Byzantine Fault Model in which the adversary is capable of compromising machines with malicious intents. In fact, the presence of the Adversary in the system is bound to a certain maximum value and can be measured in two different ways. One is through a parameter f corresponding to the maximum nodes it can control in the system. The other way is by assigning a maximum percentage of nodes it can control.

Another important aspect of the adversary is that it must be considered that it can coordinate an attack through all compromised nodes and also is possible to delay messages sent by correct nodes, but not indefinitely. However, it is assumed that it cannot compromise the security of cryptographic algorithms, namely forge Digital Signatures. Therefore, the adversary is able to conduct attacks that target Safety and Liveness properties of the system and it can be of two types: Slowly Adaptive or Strongly Adaptive. The Slowly Adaptive Adversary is capable of compromising any set of nodes given a time span while the Strongly Adaptive Adversary is able to corrupt any node instantly.

Some examples of blockchain's attacks consists of: **Double Spend** - the adversary tries to perform more than one transaction with the same Unspent Transaction Output (UTXO) as input; and **Invalid Transactions** - the adversary tries to issue invalid transactions using invalid signatures or using non-existent UTXO as input transactions.

3.3.2 Intermediate Consensus

The Consensus Plane is the core component of a DL as it solves consensus in the permissionless setting and must take into consideration the above discussed Adversary Model that corresponds to the Byzantine Fault Model. As discussed in 2.4.2, classical BFT solutions are unable to withstand sybil attacks while supporting large scale characteristics based on a dynamic membership [27]. Based on this, the consensus plane can be defined by using an Intermediate Consensus (IC) Protocol that given a committee of nodes it is possible to decide the next block to append to the chain.

In fact, unlike classical consensus algorithms in which every node participates in the protocol, in a DL this is different as it is necessary to elect a committee of nodes (in section 3.4 is presented how these committees can be elected with sybil resistant properties). After the creation of the committee, a node is then elected to be the proposer of the next block which is afterwards disseminated across all nodes in the network by some specific procedure. When defining a committee, depending on the consensus type employed it can be of two types: Single Element Committees 3.3.2.1 or Multiple Element Committees 3.3.2.2. Next, we present both committees with an analysis on the consistency guarantees of the different consensus types.

3.3.2.1 Single Element Committees

There are some implementations of DLs [56, 33] that adopt this type of committees in which there is only one element in it. In this case, the leader of a round is always the single element in the committee and so, it can issue a block and broadcast it without any further actions, which is the case of PoW and PoET consensus mechanisms.

This approach has some advantages which turned to be the most popular solution. The first is related to the consensus algorithm that is much easier to implement when compared to classical solutions in which it is needed to exchange messages in order to transition between states. Additionally, since this approach elects a single element to the committee it is possible to devise solutions that explore the concern of choosing a different participant for each round, thus ensuring fairness. With this, the system can be protected from a Strongly Adaptive Adversary.

However, this type of approach has some disadvantages in the sense that if the single node that belongs to the committee is the adversary, then it can create inconsistencies in the ledger, namely a fork. Additionally, it can broadcast malformed blocks and so the system must have a way to prevent such attacks that take the form of a Double Spending or Invalid transaction attacks. As a result, applications that execute on top of this approach need to implement a rigorous validation process that ensures a block is valid, with all its transactions. In order to perform this validation is necessary to check the validity of the digital signatures of transactions. Due to this process being slow it will delay the propagation of blocks, thus reducing the finality throughput of transactions.

3.3.2.2 Multiple Element Committees

As seen in Single Element Committees 3.3.2.1, this type suffers from some drawbacks. In order to tackle those limitations, there are solutions that employ the Multiple Element Committee [39, 44] in which it is comprised by a set of nodes that interact with each other to propose blocks.

Assuming the generated committees are fair and trustable it is possible to define the following advantages: 1) the ledger does not need to allow forks, therefore consistency is guaranteed; 2) blocks can be finalized as soon as they are appended to the chain; and 3) the application that is built on top of this type of system does not need to perform a rigorous validation of blocks in the sense that if a block is produced as a result of the operation of the committee then it is assured that it is not malformed. Therefore, the time required to disseminate a block is decreased.

On the other hand, the cost of these advantages lies on the complexity of the consensus algorithm. In the next section 3.4, we discuss different approaches to the problem of electing sybil resistant committees for two types of consensus mechanisms, namely PoS and classical solutions like PBFT.

3.4 Sybil Resistant Elected Committees

In the previous section we discussed the IC protocol 3.3.2 based on committees and how they are used to achieve consensus in DLs. In this section, we present the Sybil Resistant Election (SRE) protocol with the purpose of electing trustable and fair committees that can resist sybil attacks [27]. Note that, this approach is only relevant in the context of Multiple Element Committees as in the Single ones this problem does not apply.

Although it is not possible to define a generic implementation of the election protocol, all SRE implementations leverage the use of randomness (a random seed) to elect the next committee. Note that, in order for this election to be trustable and fair, the seed must not be biased in the sense that none of the participants can predict the next seed until it is used and also the seed must be used exactly once, thus protecting from the adversary to increase the possibilities of being elected. Next, we present two different approaches to perform the election based on the different consensus mechanisms used by blockchains.

PoS In this kind of algorithm, a committee of nodes is created with the purpose of proposing the next block to append to the chain. PoS algorithms leverage random functions that are used to determine the current membership of the committee. When taking into account that the adversary can corrupt any node, this kind of mechanisms adopt a procedure in which a node is the only one who can verify if it belongs to the committee without the need to communicate with others. Such mechanism is commonly referred as a Verifiable Random Function (VRF) and is currently adopted in the committee *Self-Selection* algorithm by Algorand [39].

A VRF [54] is a function that given two parameters, a node's Private Key and a seed, it verifies if the node belongs to the validation committee, without revealing its key. In fact, the VRF was introduced with the purpose of protecting user privacy when performing transactions in Zerocash [64] but it was later adopted by many DLs [23, 41]. In the context of Algorand [39], this function is similar to a weighted lottery in the sense that the probability of a node to be elected to a committee is related to the amount of stake it has.

Additionally, the selection procedure must be fair and is addressed by counting the number of times a user has participated in the committee, thus giving a chance to users with a lower stake. Note that, it is important for the committee to be refreshed in order to resist sybil attacks [27]. Therefore, every node in the committee participates only once by sending a single message. After that, the participants are replaced. As a result, an adversary does not know who are the participants of the committee and only realises that a node was selected to the committee after it participated in the consensus protocol, thus protecting from a Strongly Adaptive Adversary.

PBFT On the other hand, there are other approaches that try to benefit from classical solutions like PBFT [16] which are able to reach consensus in a relatively fast way while protecting against a bound number of byzantine faults. In order to guarantee safety conditions, the committee must be comprised of $3f + 1$ nodes, thus protecting the system from at maximum f byzantine nodes. Also, it ensures that once a value is decided it cannot be altered or reverted. In the context of DLs [44], this is extremely important because a block can be finalized as soon as it is appended to the chain and it does not allow forks.

However, when applying this type of solutions to permissionless blockchains, due to scalability issues, a committee of nodes must be elected to represent all participants, therefore it must be fair, trustable and resistant to sybil attacks. In the current state-of-the-art, there are no solutions to this problem as it is an object of research [81, 59] and one of the possible approaches is through the CUP model [73] discussed in the next section (3.5).

3.5 Consensus for Permissionless Model under Unknown Participants

Up to now, we have seen that current permissionless blockchains suffer from some tradeoffs/limitations when applying their single and *hardcoded* consensus mechanism, where any node can become a participant, such as: low performance, lack of consensus finality, etc. In the current state-of-the-art, solutions that integrate classical consensus algorithms in permissionless blockchains are currently an active research topic. In this section we present the CUP model [73] aiming to solve the mentioned problems.

The CUP model [73] is an approach for solving consensus where every participant can enter and leave the system without warning. In this model, a node starts its execution with a few known participants of the system based on a *participant detector*. This *participant detector* aggregates information about the known participants of the system and enables the possibility of finding other unknown nodes on the network. When joining the acquired (individual) information of all participants it is possible to build a *direct knowledge connectivity graph*, like the one presented in figure 3.4. In this graph, every node represents a process and an edge between two nodes represents the information one node has of another. The purpose of building this graph is to find a sink (a smaller group of participants with certain characteristics) in which it is possible to execute traditional BFT algorithms, like PBFT. When consensus has been solved, the nodes that participated in this task broadcast to the other ones the decided value.

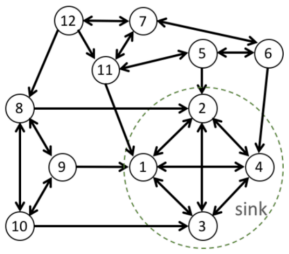

Figure 3.4: A direct knowledge connectivity graph containing a sink

However, the process of building the graph is quite complex in the sense that some properties must be guaranteed. In fact, the sink in the graph represents well connected nodes but that is not enough in which decentralized trust and fairness conditions must be ensured. This means that it is necessary to rotate the participants in the sink where each node has the same possibilities of entering the sink. Additionally, this model must not allow collusion of nodes and also it must be sybil resistant.

Based on the mentioned security properties, this model can then be applied to the consensus plane of permissionless blockchains. As a matter of fact, this model fits in the objectives of this dissertation in the sense that the main goal is to design a self adaptive consensus mechanism under the assumption of the CUP model, therefore a possible solution will be discussed in chapter 4.

3.6 Critical Analysis

Throughout this chapter we analysed different consensus mechanisms (section 3.1), some of them based on committees and how to apply them in permissionless blockchains. We have seen solutions that employ hybrid consensus planes in which PoW serves as a mean to elect a committee of nodes to participate in the consensus of blocks, usually using PBFT. We also discussed several scalability approaches that aim to improve DL's performance. In section 3.2, we presented an analysis of chained structures like Bitcoin [56] and Bitcoin-NG [33] and the consistency problems of forks. We also, analysed different scalability solutions covering a wide spectrum from DAG models, Sharding, Multichains and Parallel chains.

In relation to DAG models, we concluded that solutions based on this approach can be divided in DAG Block oriented or Blockless types. Although those solutions improve performance, some are vulnerable to liveness attacks [50, 67] or provide a weaker form of consistency [66] that prevents the support of smart contracts, which in our solution is a must. Regarding Sharding, we came across solutions that are able to scale-out by dividing the state across all nodes [47, 80]. Multichain solutions that allows to load-balance transaction processing to different systems and Parallel chains consisting of multiple blockchain instances which increases the rate of finalized blocks.

In section 3.3, we discussed different consensus types and consistency guarantees which comprise the IC protocol. When referring to PoW or PoET consensus mechanisms we concluded that their implementation is simpler but at the cost of reducing the throughput of the DL in order to ensure an acceptable ratio between the rate of proposed blocks and their transmission, thus ensuring safety properties of the system. On the other hand, we analysed solutions that are based on committees like PoS and PBFT which are able to achieve a higher throughput of transactions by delegating the validation procedure to the committee, so the remaining nodes perform a more lightweight validation.

However, when applying committee based consensus to permissionless blockchains, it is necessary to perform a fair, trustable and sybil resistant [27] election based on a randomized process. Additionally, the committee must be refreshed in order to resist committee corruption attacks. In section 3.4, we have seen the PPoS variant based on committees employed by Algorand [39] in which the election corresponds to a *Cryptographic Sortition* process that leverages VRF that when applied to the stake of the participants generates a committee. Additionally, the use of classical algorithms like PBFT improve significantly the performance of the system. However, in the current state-of-the-art there are no solutions to permissionless blockchains that apply a hybrid and flexible Consensus Plane with the PBFT mechanism, being a research challenge in which one of the possible approaches is based on the CUP model [73], presented in section 3.5.

Next chapter, we present our elaboration approach that consists on building a DL based on a Hybrid and Flexible Consensus that targets the CUP model to support the PBFT consensus mechanism. We believe that a solution that aggregates multiple consensus mechanisms can benefit from the tradeoffs of each one, thus achieving higher performance.

4 Approach to Elaboration Phase

As presented before, HyFlexChain is a permissionless solution for scalable ledger using a flexible and hybrid consensus plane, in which different consensus mechanisms will be used alternatively or in combination, by expressing these options with smart contracts. In this chapter we will address the main ideas and initial guidelines for design options, namely system model and architecture 4.1, design approach for system components 4.2, prototyping and implementation options 4.3, validation and experimental evaluation 4.4 and finally, the work plan 4.5.

4.1 HyFlexChain System Model and Architecture

In a high level view, HyFlexChain corresponds to a blockchain in which it is composed by different nodes that interact with each other in a P2P network. In figure 4.1, is presented a high level view of our system model and the different planes that compose it. According to the presented architecture, some components will be leveraged by a base system in which our solution will be implemented, therefore they have a gray background color (the darker color in the Crypto Service Plane means that it is provided by a library). Our focus relies on the following planes: Application Plane, Transaction Management Plane, Hybrid Consensus Plane and Execution Plane. Next, we provide a brief description of their purposes.

Application Plane The Application Plane is responsible for accepting and processing client requests via an external API (enabled by REST/HTTPS) endpoint. The provided operations in the API will be organized in three different interfaces: Transaction Interface (TI); Smart Contract Management Interface (SCMI) and a Ledger-View Interface (LVI). TI corresponds to the client support provided to send transactions that will be submitted to a preliminary validation which will then be dispatched to the Transaction Management Plane. SCMI externalized operations to install and revoke smart contracts in a on-chain model. Smart contracts must be validated according to execution constraints and security policies established in the runtime plane and disseminated for the consistent validation

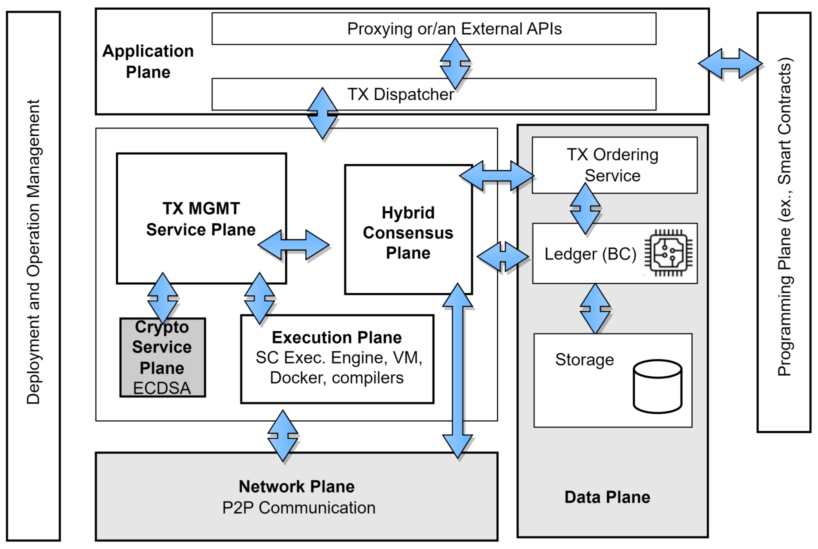
Application Plane
Proxying or/an External APIs
TX Dispatcher
TX MGMT Service Plane
Hybrid Consensus Plane
TX Ordering Service
Ledger (BC)
Storage
Crypto Service Plane ECDSA
Execution Plane SC Exec. Engine, VM, Docker, compilers
Network Plane P2P Communication
Data Plane
Deployment and Operation Management
Programming Plane (ex., Smart Contracts)

Figure 4.1: HyFlexChain Architecture

of other HyFlexChain nodes with special transactions to install and revoke the validated smart contracts in the replicated chain.

Transaction Management Plane This plane is responsible for receiving client transactions via the Application Plane and performing the necessary verifications, such as: transaction verification like ECDSA signatures, smart contract's execution and insertion on blocks which will then be ordered by the Consensus Plane.

Execution Plane This plane is composed by the mechanisms for the validation, compilation and execution of smart contracts. It is in this plane that transactions trigger the execution of smart contracts which can imply changes to the system's runtime settings. These settings affect the consistency model used for blocks that will be ordered under the same consensus verifications which are provided by the different consensus models supported by the Consensus Plane. For the intended purpose, the execution plane contains a virtual execution environment providing testing capabilities for the validation of smart contracts according to security and execution constrained policies (without execution) and execution of validated smart contracts used by transactions.

Consensus Plane This plane deals with the creation of blocks and their ordering in a way that ensures a correct replication of the DL. In our solution, this plane will be designed to support a hybrid consensus model, using different consensus mechanisms and their protocols. In the consensus plane we support multiple consensus mechanisms, that can be used in a flexible way, targeting on the following models: PoW, PoS, PoET and PBFT. By offering a pluggable consensus component [62, 81], the consensus plane provides an interface to access the common primitives of each supported mechanism, in a modular way. Some algorithms (PoS and PBFT) require the existence of formed committees, with anti-sybil resistance capabilities. For the purpose, a core-component in the Consensus Plane will be a Sybil Resistant Committee Election mechanism, with parameterizations to select committees for PoS and PBFT. Moreover, these committees will be managed in order to achieve fairness conditions with randomized assumptions to rotate or to select committee members with sybil-resistance properties.

Data Plane This plane implements the data structures used to record all transaction/blocks in the ledger which then communicates with the storage plane. It is also responsible for saving smart contracts which can then be referenced in future transactions by an identifier in order to be executed. Depending on the type of system we are going to extend, there are different type of structures that we will consider (see 4.3).

Network Plane The Network Plane is a base plane that provides a way to exchange messages between nodes in a P2P network. It is responsible for propagating information (transactions, blocks, consensus messages, etc) to all nodes across the different stages of the system's operation.

4.2 Design Approach for System Components

In this section we present our design approach specifically focused on the main components we will address, namely: the Application Plane with the 3 REST APIs, the Hybrid Consensus Plane, smart contract's flexibility and expressiveness and some scale-in approaches and open challenges.

4.2.1 Application Plane Components

The TI corresponds to an external API that receives transactions proposed by clients. This API will need to verify if the submitted transaction is valid, starting by verifying if the it has the correct syntax and validating the digital signatures. For this, we will use ECDSA signatures which will be provided by an external library. In case of a transaction being considered invalid, it will not be accepted and an error code is returned to the client.

Regarding the SCMI, smart contracts can be submitted through it by any client. Each smart contract (on-chain) will have a unique identifier based on a public-key and digital signature. Only the creator of the smart contract can revoke it on demand, but it will be also considered revoked when the time validity has expired. Only valid smart contracts (consistently stored in finalized blocks) can be used by transactions. Transactions can refer smart contracts to be used in the validation and processing phase. In addition to the transaction's data, this plane is also responsible for accepting embedded smart contracts that will be verified, installed on-chain and executed in the context of a transaction when it is referenced, thus allowing to process Application Specific Validations.

The operations provided by the LVI externalizes the state of the full ledger maintained by the node. The result of operations (that will include ledger searching capabilities) are data view structures derived from the full ledger whose state is obtained by applying all transactions. For performance reasons, pre-computed views may be stored in the storage plane to be accessed in an authenticated fashion. We must notice that permissionless blockchains do not implement this optimization at application level. Miners would need days to reconstruct UTXO sets (which can be considered as a "current" view) from the beginning of time. On the other hand, for external wallet-applications the LVI provides a necessary support for users to manage their UTXOs (representing unspent transactions in the case of cryptocurrency applications) or management of sent or owned digital assets, in the current state of the ledger. In general, a view can be obtained from an arbitrary function of the full ledger (not being necessarily UTXO sets). As a piece of data in the storage plane, each view must be determined either implicitly or explicitly by the consensus plane and must allow authentication for the read operation against it. In our design model, views will be supported via replication. For example, Bitcoin [56], Ethereum [76] and other popular decentralized cryptocurrencies require all consensus nodes to verify all transactions and so, based on the result of the computation, updating their respective views (UTXO sets), locally. In our case, view-states are an implicit output of the Consensus Plane and may be regarded as pre-processed and residing in the storage plane. Assuming it is correctly computed, it represents the computation of an honest set of consensus nodes (each with the same view) providing high availability.

4.2.2 Hybrid Consensus Plane

As previously stated, the purpose of HyFlexChain is to design and implement a Hybrid Consensus Plane that integrates multiple consensus algorithms, such as: PoW, PoS, PoET and PBFT. In figure 4.2 we present the architecture of this component.

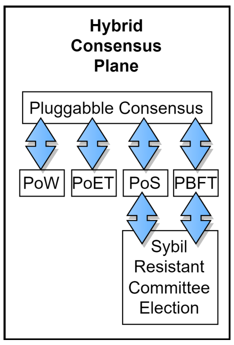

Figure 4.2: Hybrid Consensus Plane

We will build the consensus plane services on top of underlying blockchain protocols, which we consider as bootstrap slow chains providing a base PoW consensus mechanism. Another possibility is to use a base-blockchain already providing a PoS model based on fair committees based on the participant's stake but using a randomization scheme to defend from the effect of possible collusions. From the base-slow chain we can form anti-sybil committees for the HyFlexChain which will be used by the consensus plane to support transactions and their aggregations in finalized blocks. For HyFlexChain daily operations and supported consensus mechanisms, committees will work with a temporary-validation assumption of two types: elapsed time and a certain number of consecutive participation on block's ordering. To form a new committee from an older one, without leaving gaps of inactivity in between is somewhat tricky. We will tackle this by studying an approach with the following characteristics.

When the new committee is selected, it starts a procedure to be sure that all the elements are committed to start operations in a certain committee-based permissioned consensus model (PoET or PBFT). A fair sortition of PoS committees is already addressed by blockchains we can consider to leverage HyFlexChain, for example Algorand [2]. For PoW based committee sortition we will design a mechanism using a cryptographic primitive running a public randomness source (ex. drand [28]) and a cryptographic sortition protocol based on prior groups and a current list of participants involved in the last N transactions, with N possibly parameterized to correspond to the size of the committees. To enable validators to prove that they belong to the selected subset, they need a public/private key pair to be able to generate a proof using a VRF. One open possibility to mitigate the latency in the formation of committees, is to create a set of new committees, to start operations in different future instants of time.

At the same time, a new selected committee is formed and the old committee members previously involved in the type of consensus model initiates a stopping procedure of the existing consensus instance. This would introduce several issues because in transient windows, two (or more) instances of different consensus protocols may be concurrently executing which, in these cases, nodes need to correctly linearize the outputs of the multiple instances. Also, we need to ensure concurrent composition of the multiple instances.

4.2.3 Flexible Consensus with Smart Contracts

The design of the support for smart contracts will be one of the first tasks in the elaboration phase, together with the design of the APIs at the level of the application plane. Different directions can be explored here.

The first approach will be the use of smart contracts programmed in Java. The bytecodes corresponding to the smart contracts can be executed in a docker component in the execution plane, offering an isolated JVM [45] together with pre-established sealed security policy management rules. In this case, we will define smart contracts as implementations of a defined interface model for a Java serializable object (with a run entrypoint function, where the contract will be executed).

Another alternative is to use smart contracts developed using the Solidity programming language [65] and using EVM [32] in a docker component included in the execution plane.

4.2.4 Scale-In Criteria and Open Challenges

Beyond the possibility for applications to use alternatively or in combination, different consensus mechanisms provided by the HyFlexChain consensus plane, we will address complementary scale-in conditions, particularly targeting the use of Blockmess [15] as the leveraging solution. Blockmess uses parallel chains to improve the throughput of the system while degrading the finalization time. Then, a possible improvement is the use of the parallel chains to also improve finalization time, as done in Prism [9], while the application demands for throughput are low and gradually migrate the use of the parallel chains to increase throughput as the load in the application increases.

While Blockmess estimates the load of the system and creates new parallel chains to adapt application demands, the number of chains the system settles in equilibrium is sub-optimal, as more chains are used than the required. We will study how different parallel chains with different consensus mechanisms can mitigate the problem by complementing the Blockmess solution with a fixed number of parallel chains per consensus-type, effectively creating parallel Blockmess instances usable for different consensus models. If it is known that an application requirements will not drop bellow a certain threshold, we do not need to merge chains bellow a certain number. By not requiring the logic of spawning and merging chains, the fixed parallel chains per consensus types will have a more uniform distribution of load then the original Blockmess. To address this improvement, we must modify the current mechanism used by Blockmess to deterministically allocate transactions to different parallel chains.

4.3 Design, Prototyping and Implementation Options

In this section we explore two alternatives to implement HyFlexChain. As stated before, HyFlexChain will be designed and implemented on top of existing systems as a way to improve their performance by focusing on the consensus plane of those solutions. Then, for each of those approaches we specialize the generic architecture of HyFlexChain and discuss the tradeoffs of both solutions, namely through Algorand [39] and Blockmess [15].

4.3.1 Algorand Leveraged Solution

One of the approaches to develop HyFlexChain is through the Algorand blockchain. This blockchain is public and is inserted on the permissionless model that uses PoS as the consensus mechanism in order to perform the agreement on the next block to append to the chain. More specifically, this system uses the PPoS variant which needs a committee of participants to execute the protocol.

Since this system already provides an implementation for the PoS consensus mechanism, our objective is to integrate two more mechanisms, such as: PoW and PBFT. In order to add more consensus mechanisms, we will analyse the code and refactor it to implement a pluggable component which will allow to insert more algorithms and to switch between them at runtime.

Regarding the PBFT implementation, it will be necessary to perform a sybil-resistant committee election. In fact, Algorand already does this election procedure, therefore our approach is to try to utilize the elected committee in the PBFT mechanism. However, this approach is dependent on the modularity of the election component and how it is integrated with the original consensus algorithm it was developed for. In case of not being possible to reuse that component, we will design an algorithm that randomly chooses a set of nodes based on their correct participation on previous consensus rounds while taking into account a fair election to avoid collusion of nodes.

Note that, the data structures that form the ledger of Algorand are based on a single chain. Based on this, since we are going to develop a component that allows to switch between consensus mechanisms, different blocks will be ordered by different mechanisms. To illustrate this result, we present in figure 4.3 an example of the ledger.

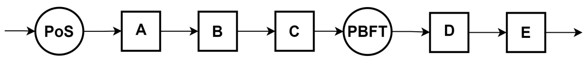

Figure 4.3: HyFlexChain Algorand Ledger

Finally, as we already mentioned, our solution will be based on smart contracts with enough expressiveness in order to allow a flexible switch of consensus mechanisms. In this case, Algorand already provides an implementation for smart contracts, therefore our approach corresponds to extend their implementation with the necessary features that we discussed.

4.3.2 Blockmess Inspired Solution

Another approach to implement HyFlexChain is inspired by Blockmess, a solution that applies parallel chains in which their number is dynamically adapted to meet application demands, limiting the latency cost and minimizing bandwidth waste in high demand applications. This system works by the assumption that when the rate of proposed transactions is greater than the throughput of the system, it creates more chains and each chain can then spawn even more chains, case it is necessary. This structure forms a tree where each chain represents a path from the *genesis* block and the last block on each chain.

Based on this architecture, our approach will be based on exploring different consensus mechanisms in different chains. In fact, this approach will leverage the algorithm employed by Blockmess to decide in which chain a transaction will be added. Its main idea is for every chain to have a mask and so, the masks decide where transactions can be placed by applying a pattern matching between them and the transaction's identifier.

Our approach consists of using smart contracts to decide which consensus mechanism must be used to order a block in which that transaction is inserted. Since Blockmess does not implement smart contracts and it is a requirement for our solution we will implement that technology which will be focused on providing enough expressiveness in order to define rules for when each consensus mechanism must be used. After deciding the mechanism it is necessary to decide which chain a block with that transaction should be placed. That decision will leverage the algorithm of Blockmess.

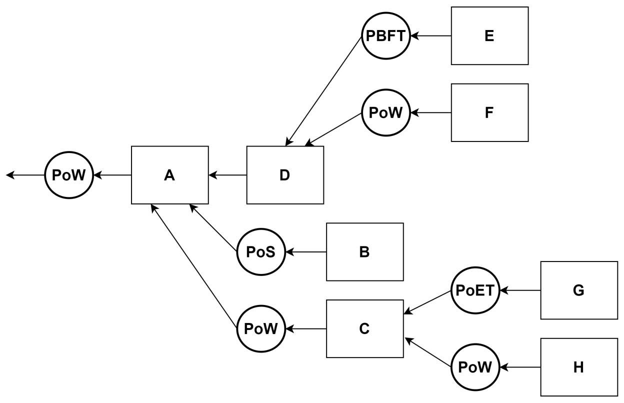

Figure 4.4: HyFlexChain Blockmess Ledger

In figure 4.4 is presented a possible state of the ledger when applying different consensus mechanisms to different chains. According to the system rules, when the rate of proposed transactions is greater than the throughput of a chain, two more chains are created. Additionally, with our proposed solution we modify that rule to spawn more chains based on the old rule and on different consensus mechanisms.

4.4 Validation and Experimental Evaluation

The main objective of HyFlexChain is to improve performance in permissionless DLs and so, we are going to evaluate specific metrics, such as: throughput and blocks' finalization time measurements. These measurements will allow us to attest the benefits of our approach, which in that case can prove that HyFlexChain can indeed achieve a higher performance than other permissionless DLs. Also, we will analyze possible drawbacks that might appear as a result of our approach.

Another type of measurements that must be taken into account is the HyFlexChain reaction to shifts between consensus models in application demands expressed by the provided smart contracts. In this case, we will test different smart contract's rules and submit a high number of different types of transactions in order to verify if the cost of switching from one type of consensus mechanism with some consistency guarantees to other with different characteristics has an impact on the overall performance of the system. With this, it will be possible to measure the benefits from switching consensus mechanisms at runtime while also taking into account the needed processing to such switches.

Additionally, we will perform a series of workloads with the purpose of comparing the performance of the system observed by benchmarks at the application level. These workloads will explore the different consensus models that must be employed by the

Hybrid Consensus Plane solution. Initially, we will begin with workloads of only one consensus mechanism. Then, execute for 2, 3 and 4 consensus mechanisms. The expectation of the results of those benchmarks is that with the increase of the number of consensus mechanisms on a workload, the higher the overall performance of the system.

Initially, we intend to run our experiments on virtual machines provided by the DI-FCT-UNL computation cluster in order to verify and validate our solution with the above test cases. Since we are testing a permissionless DL, it will be necessary to conduct tests with a high number of HyFlexChain nodes in order to obtain plausible results. Afterwards, we will test our solution in a series of geographically distributed servers based on a global cloud provider infrastructure. With this, we intend to run a total of 200 HyFlexChain nodes.

4.5 Elaboration Work Plan

Lastly, we present a schedule for the elaboration plan of the present dissertation during the 2^{sn} semester of 2022/2023. In figure 4.5 is summarized the elaboration phase work plan. Additionally, in annex I, we present a detailed description of planned activities.

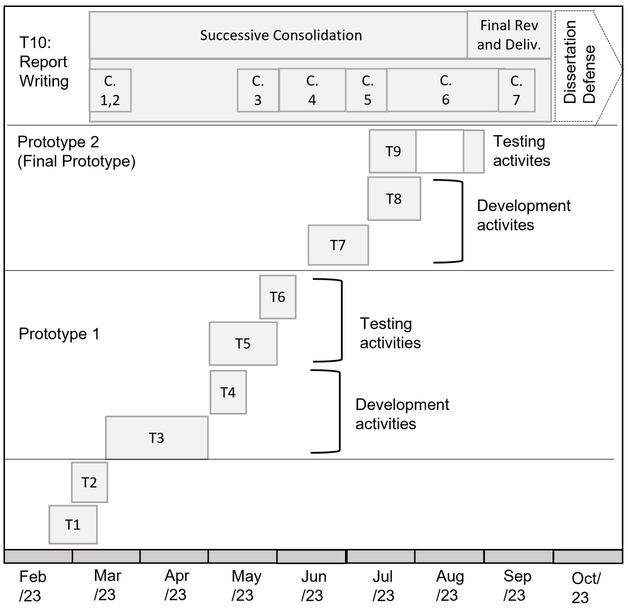

Figure 4.5: Planned activities and tasks of the elaboration work plan

Bibliography

[1] I. Abraham et al. *Solida: A Blockchain Protocol Based on Reconfigurable Byzantine Consensus*. Cryptology ePrint Archive, Paper 2017/1118. https://eprint.iacr.org/2017/1118. 2017. URL: https://eprint.iacr.org/2017/1118 (cit. on pp. 2, 9, 16, 17).

[2] *Algorand*. URL: https://www.algorand.com (visited on 2022-11-21) (cit. on pp. 4, 16, 17, 31).

[3] *Algorand Smart Contracts (ASC1)*. URL: https://developer.algorand.org/docs/get-details/dapps/smart-contracts (visited on 2023-01-13) (cit. on p. 10).

[4] *Algorand's Smart Contract Architecture*. URL: https://www.algorand.com/resources/blog/algorand-smart-contract-architecture (visited on 2023-01-13) (cit. on p. 10).

[5] E. Androulaki et al. "Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains". In: *Proceedings of the Thirteenth EuroSys Conference*. EuroSys '18. Porto, Portugal: Association for Computing Machinery, 2018. ISBN: 9781450355841. DOI: 10.1145/3190508.3190538. URL: https://doi.org/10.1145/3190508.3190538 (cit. on pp. 2, 9, 10).

[6] N. Atzei et al. "SoK: Unraveling Bitcoin Smart Contracts". In: *Principles of Security and Trust*. Ed. by L. Bauer and R. Küsters. Cham: Springer International Publishing, 2018, pp. 217–242 (cit. on p. 10).

[7] P.-L. Aublin, S. B. Mokhtar, and V. Quéma. "RBFT: Redundant Byzantine Fault Tolerance". In: *2013 IEEE 33rd International Conference on Distributed Computing Systems*. 2013, pp. 297–306. DOI: 10.1109/ICDCS.2013.53 (cit. on p. 1).

[8] *Algorand Virtual Machine (AVM)*. URL: https://developer.algorand.org/docs/get-details/dapps/avm/ (visited on 2023-02-08) (cit. on p. 10).

[9] V. Bagaria et al. "Prism: Deconstructing the Blockchain to Approach Physical Limits". In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. CCS '19. London, United Kingdom: Association for Computing

Machinery, 2019, pp. 585–602. ISBN: 9781450367479. DOI: 10.1145/3319535.3363213. URL: https://doi.org/10.1145/3319535.3363213 (cit. on pp. 9, 16, 17, 21, 32).

[10] L. Baird. "The swirlds hashgraph consensus algorithm: Fair, fast, byzantine fault tolerance". In: *Swirlds Tech Reports SWIRLDS-TR-2016-01, Tech. Rep* 34 (2016), pp. 9–11. URL: https://eclass.upatras.gr/modules/document/file.php/CEID1175/Pool-of-Research-Papers%5C%5B0%5C%5D/31.HASH-GRAPH.pdf (cit. on pp. 9, 16, 17, 19, 20).

[11] M. Al-Bassam et al. *Chainspace: A Sharded Smart Contracts Platform*. 2017. DOI: 10.48550/ARXIV.1708.03778. URL: https://arxiv.org/abs/1708.03778 (cit. on p. 9).

[12] R. Belchior et al. *A Survey on Blockchain Interoperability: Past, Present, and Future Trends*. 2020. DOI: 10.48550/ARXIV.2005.14282. URL: https://arxiv.org/abs/2005.14282 (cit. on p. 21).

[13] I. Bentov et al. *Tortoise and Hares Consensus: the Meshcash Framework for Incentive-Compatible, Scalable Cryptocurrencies*. Cryptology ePrint Archive, Paper 2017/300. https://eprint.iacr.org/2017/300. 2017. URL: https://eprint.iacr.org/2017/300 (cit. on pp. 9, 16, 17, 19).

[14] C. Cachin, R. Guerraoui, and L. Rodrigues. *Introduction to reliable and secure distributed programming*. 2011. ISBN: 978-364215259-7. DOI: 10.1007/978-3-642-15260-3 (cit. on p. 12).

[15] P. Camponês. *Blockmess: Adaptive Parallel Chains with Deterministic Transaction Allocation*. 2021 (cit. on pp. 4, 9, 16–18, 21, 32).

[16] M. Castro and B. Liskov. "Practical Byzantine Fault Tolerance". In: *Proceedings of the Third Symposium on Operating Systems Design and Implementation*. OSDI '99. New Orleans, Louisiana, USA: USENIX Association, 1999, pp. 173–186. ISBN: 1880446391 (cit. on pp. 1, 2, 14, 24).

[17] J. Chen and S. Micali. *Algorand*. 2016. DOI: 10.48550/ARXIV.1607.01341. URL: https://arxiv.org/abs/1607.01341 (cit. on pp. 10, 16, 17).

[18] L. Chen et al. "On Security Analysis of Proof-of-Elapsed-Time (PoET)". In: *Stabilization, Safety, and Security of Distributed Systems*. Ed. by P. Spirakis and P. Tsigas. Cham: Springer International Publishing, 2017, pp. 282–297. ISBN: 978-3-319-69084-1 (cit. on pp. 2, 13).

[19] K. Christidis and M. Devetsikiotis. "Blockchains and Smart Contracts for the Internet of Things". In: *IEEE Access* 4 (2016), pp. 2292–2303. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2016.2566339 (cit. on p. 9).

[20] K. Croman et al. “On Scaling Decentralized Blockchains”. In: *Financial Cryptography and Data Security*. Ed. by J. Clark et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 106–125. ISBN: 978-3-662-53357-4 (cit. on p. 2).

[21] G. Danezis and D. Hrycyszyn. *Blockmania: from Block DAGs to Consensus*. 2018. DOI: 10.48550/ARXIV.1809.01620. URL: https://arxiv.org/abs/1809.01620 (cit. on pp. 9, 16, 17).

[22] H. Dang et al. “Towards Scaling Blockchain Systems via Sharding”. In: *Proceedings of the 2019 International Conference on Management of Data*. SIGMOD ’19. Amsterdam, Netherlands: Association for Computing Machinery, 2019, pp. 123–140. ISBN: 9781450356435. DOI: 10.1145/3299869.3319889. URL: https://doi.org/10.1145/3299869.3319889 (cit. on p. 20).

[23] B. David et al. “Ouroboros Praos: An Adaptively-Secure, Semi-synchronous Proof-of-Stake Blockchain”. In: *Advances in Cryptology – EUROCRYPT 2018*. Ed. by J. B. Nielsen and V. Rijmen. Cham: Springer International Publishing, 2018, pp. 66–98. ISBN: 978-3-319-78375-8 (cit. on p. 24).

[24] C. Decker, J. Seidel, and R. Wattenhofer. “Bitcoin Meets Strong Consistency”. In: *Proceedings of the 17th International Conference on Distributed Computing and Networking*. ICDCN ’16. Singapore, Singapore: Association for Computing Machinery, 2016. ISBN: 9781450340328. DOI: 10.1145/2833312.2833321. URL: https://doi.org/10.1145/2833312.2833321 (cit. on pp. 2, 9, 16, 17).

[25] C. Decker and R. Wattenhofer. “Information propagation in the Bitcoin network”. In: *IEEE P2P 2013 Proceedings*. 2013, pp. 1–10. DOI: 10.1109/P2P.2013.6688704 (cit. on p. 18).

[26] *Docker*. URL: https://docs.docker.com/ (visited on 2023-02-08) (cit. on p. 10).

[27] J. R. Douceur. “The sybil attack”. In: *International workshop on peer-to-peer systems*. Springer. 2002, pp. 251–260 (cit. on pp. 15, 20, 22–24, 26).

[28] *drand API*. URL: https://drand.love (visited on 2023-02-10) (cit. on p. 31).

[29] C. Dwork, N. Lynch, and L. Stockmeyer. “Consensus in the Presence of Partial Synchrony”. In: *J. ACM* 35.2 (1988-04), pp. 288–323. ISSN: 0004-5411. DOI: 10.1145/42282.42283. URL: https://doi.org/10.1145/42282.42283 (cit. on p. 12).

[30] C. Dwork and M. Naor. “Pricing via Processing or Combatting Junk Mail”. In: *Proceedings of the 12th Annual International Cryptology Conference on Advances in Cryptology*. CRYPTO ’92. Berlin, Heidelberg: Springer-Verlag, 1992, pp. 139–147. ISBN: 3540573402 (cit. on pp. 2, 13).

[31] C. Dwork and M. Naor. “Pricing via Processing or Combatting Junk Mail”. In: *Proceedings of the 12th Annual International Cryptology Conference on Advances in Cryptology*. CRYPTO ’92. Berlin, Heidelberg: Springer-Verlag, 1992, pp. 139–147. ISBN: 3540573402 (cit. on p. 13).

[32] *Ethereum Virtual Machine (EVM)*. URL: https://ethereum.org/en/developers/docs/evm/ (visited on 2023-02-08) (cit. on pp. 10, 31).

[33] I. Eyal et al. “Bitcoin-NG: A Scalable Blockchain Protocol”. In: *Proceedings of the 13th Usenix Conference on Networked Systems Design and Implementation*. NSDI’16. Santa Clara, CA: USENIX Association, 2016, pp. 45–59. ISBN: 9781931971294 (cit. on pp. 2, 9, 16–19, 22, 26).

[34] C. Fan et al. “Performance Evaluation of Blockchain Systems: A Systematic Survey”. In: *IEEE Access* 8 (2020), pp. 126927–126950. DOI: 10.1109/ACCESS.2020.3006078 (cit. on pp. 6, 10).

[35] M. S. Ferdous et al. *Blockchain Consensus Algorithms: A Survey*. 2020. DOI: 10.48550/ARXIV.2001.07091. URL: https://arxiv.org/abs/2001.07091 (cit. on p. 1).

[36] M. J. Fischer, N. A. Lynch, and M. S. Paterson. “Impossibility of Distributed Consensus with One Faulty Process”. In: *J. ACM* 32.2 (1985-04), pp. 374–382. ISSN: 0004-5411. DOI: 10.1145/3149.214121. URL: https://doi.org/10.1145/3149.214121 (cit. on p. 12).

[37] M. Fitzi et al. *Parallel Chains: Improving Throughput and Latency of Blockchain Protocols via Parallel Composition*. Cryptology ePrint Archive, Paper 2018/1119. 2018. URL: https://eprint.iacr.org/2018/1119 (cit. on p. 21).

[38] A. Gągol et al. *Aleph: Efficient Atomic Broadcast in Asynchronous Networks with Byzantine Nodes*. 2019. DOI: 10.48550/ARXIV.1908.05156. URL: https://arxiv.org/abs/1908.05156 (cit. on p. 20).

[39] Y. Gilad et al. “Algorand: Scaling Byzantine Agreements for Cryptocurrencies”. In: *Proceedings of the 26th Symposium on Operating Systems Principles*. SOSP ’17. Shanghai, China: Association for Computing Machinery, 2017, pp. 51–68. ISBN: 9781450350853. DOI: 10.1145/3132747.3132757. URL: https://doi.org/10.1145/3132747.3132757 (cit. on pp. 9, 23, 24, 26, 32).

[40] G. Greenspan. *MultiChain White Paper*. 2015. URL: https://www.multichain.com/white-paper (cit. on p. 9).

[41] T. Hanke, M. Movahedi, and D. Williams. *DFINITY Technology Overview Series, Consensus System*. 2018. DOI: 10.48550/ARXIV.1805.04548. URL: https://arxiv.org/abs/1805.04548 (cit. on p. 24).

[42] M. Hearn and R. G. Brown. *Corda: A distributed ledger*. 2019. URL: https://www.r3.com/wp-content/uploads/2019/08/corda-technical-whitepaper-August-29-2019.pdf (cit. on pp. 2, 9).

[43] “Hyperledger Architecture, Volume II: Smart Contracts”. In: *The Linux Foundation* (2018). URL: https://www.hyperledger.org/wp-content/uploads/2018/04/Hyperledger_Arch_WG_Paper_2_SmartContracts.pdf (cit. on p. 10).

[44] *Hyperledger Sawtooth*. URL: https://www.hyperledger.org/use/sawtooth (visited on 2022-11-20) (cit. on pp. 2, 23, 24).

[45] *Java Virtual Machine (JVM)*. URL: https://docs.oracle.com/javase/specs/jvms/se19/html/ (visited on 2023-02-08) (cit. on pp. 10, 31).

[46] E. Kokoris-Kogias et al. "Enhancing Bitcoin Security and Performance with Strong Consistency via Collective Signing". In: *Proceedings of the 25th USENIX Conference on Security Symposium*. SEC'16. Austin, TX, USA: USENIX Association, 2016, pp. 279–296. ISBN: 9781931971324 (cit. on pp. 2, 9, 16, 17).

[47] E. Kokoris-Kogias et al. "OmniLedger: A Secure, Scale-Out, Decentralized Ledger via Sharding". In: *2018 IEEE Symposium on Security and Privacy (SP)*. 2018, pp. 583–598. DOI: 10.1109/SP.2018.000-5 (cit. on pp. 9, 16, 17, 20, 26).

[48] R. Kotla et al. "Zyzzyva: Speculative Byzantine Fault Tolerance". In: *ACM Trans. Comput. Syst.* 27.4 (2010-01). ISSN: 0734-2071. DOI: 10.1145/1658357.1658358. URL: https://doi.org/10.1145/1658357.1658358 (cit. on p. 1).

[49] L. Lamport, R. Shostak, and M. Pease. "The Byzantine Generals Problem". In: *ACM Trans. Program. Lang. Syst.* 4.3 (1982-07), pp. 382–401. ISSN: 0164-0925. DOI: 10.1145/357172.357176. URL: https://doi.org/10.1145/357172.357176 (cit. on pp. 1, 12).

[50] Y. Lewenberg, Y. Sompolinsky, and A. Zohar. "Inclusive Block Chain Protocols". In: *Financial Cryptography and Data Security*. Ed. by R. Böhme and T. Okamoto. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 528–547. ISBN: 978-3-662-47854-7 (cit. on pp. 20, 26).

[51] C. Li et al. "A decentralized blockchain with high throughput and fast confirmation". In: *2020 {USENIX} Annual Technical Conference ({USENIX}{ATC} 20)*. 2020, pp. 515–528 (cit. on pp. 9, 16, 17, 19).

[52] L. Luu et al. "A Secure Sharding Protocol For Open Blockchains". In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. CCS '16. Vienna, Austria: Association for Computing Machinery, 2016, pp. 17–30. ISBN: 9781450341394. DOI: 10.1145/2976749.2978389. URL: https://doi.org/10.1145/2976749.2978389 (cit. on pp. 9, 16, 17, 20).

[53] J.-P. Martin and L. Alvisi. "Fast Byzantine Consensus". In: *IEEE Transactions on Dependable and Secure Computing* 3.3 (2006), pp. 202–215. DOI: 10.1109/TDSC.2006.35 (cit. on p. 1).

[54] S. Micali, M. Rabin, and S. Vadhan. "Verifiable random functions". In: *40th Annual Symposium on Foundations of Computer Science (Cat. No.99CB37039)*. 1999, pp. 120–130. DOI: 10.1109/SFFCS.1999.814584 (cit. on p. 24).

[55] P. R. Nair and D. R. Dorai. "Evaluation of Performance and Security of Proof of Work and Proof of Stake using Blockchain". In: *2021 Third International Conference on Intelligent Communication Technologies and Virtual Mobile Networks (ICICV)*. 2021, pp. 279–283. DOI: 10.1109/ICICV50876.2021.9388487 (cit. on pp. 2, 13).

[56] S. Nakamoto. *Bitcoin: A peer-to-peer electronic cash system*. 2009. URL: https://bitcoin.org/bitcoin.pdf (cit. on pp. 2, 5, 9, 10, 15–18, 22, 26, 30).

[57] K. Olson et al. "Sawtooth: an introduction". In: *The Linux Foundation* (2018) (cit. on p. 10).

[58] R. Pass and E. Shi. *Hybrid Consensus: Efficient Consensus in the Permissionless Model*. Cryptology ePrint Archive, Paper 2016/917. https://eprint.iacr.org/2016/917. 2016. URL: https://eprint.iacr.org/2016/917 (cit. on pp. 2, 9, 16, 17).

[59] R. Pass and E. Shi. "Rethinking Large-Scale Consensus". In: *2017 IEEE 30th Computer Security Foundations Symposium (CSF)*. 2017, pp. 115–129. DOI: 10.1109/CSF.2017.37 (cit. on pp. 2, 24).

[60] S. Popov. "The tangle". In: *White paper* 1.3 (2018), p. 30 (cit. on pp. 9, 16, 17, 19).

[61] *Quorum*. URL: https://consensys.net/quorum (visited on 2023-01-03) (cit. on p. 9).

[62] M. Rasolroveicy and M. Fokaefs. "Dynamic Reconfiguration of Consensus Protocol for IoT Data Registry on Blockchain". In: *Proceedings of the 30th Annual International Conference on Computer Science and Software Engineering*. CASCON '20. Toronto, Ontario, Canada: IBM Corp., 2020, pp. 227–236 (cit. on pp. 3, 10, 28).

[63] G. Sagirlar et al. "Hybrid-IoT: Hybrid Blockchain Architecture for Internet of Things - PoW Sub-Blockchains". In: *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*. 2018, pp. 1007–1016. DOI: 10.1109/Cybermatics_2018.2018.00189 (cit. on p. 9).

[64] E. B. Sasson et al. "Zerocash: Decentralized Anonymous Payments from Bitcoin". In: *2014 IEEE Symposium on Security and Privacy (SP)*. Los Alamitos, CA, USA: IEEE Computer Society, 2014-05, pp. 459–474. DOI: 10.1109/SP.2014.36. URL: https://doi.ieeecomputersociety.org/10.1109/SP.2014.36 (cit. on p. 24).

[65] *Solidity - High-level language for implementing smart contracts*. URL: https://docs.soliditylang.org/en/latest (visited on 2023-01-06) (cit. on pp. 10, 31).

[66] Y. Sompolinsky, Y. Lewenberg, and A. Zohar. *SPECTRE: A Fast and Scalable Cryptocurrency Protocol*. Cryptology ePrint Archive, Paper 2016/1159. https://eprint.iacr.org/2016/1159. 2016. URL: https://eprint.iacr.org/2016/1159 (cit. on pp. 9, 16, 17, 19, 20, 26).

[67] Y. Sompolinsky, S. Wyborski, and A. Zohar. "PHANTOM GHOSTDAG: A Scalable Generalization of Nakamoto Consensus: September 2, 2021". In: *Proceedings of the 3rd ACM Conference on Advances in Financial Technologies*. AFT '21. Arlington, Virginia: Association for Computing Machinery, 2021, pp. 57–70. ISBN: 9781450390828. DOI: 10.1145/3479722.3480990. URL: https://doi.org/10.1145/3479722.3480990 (cit. on pp. 9, 16, 17, 19, 20, 26).

[68] Y. Sompolinsky and A. Zohar. *Accelerating Bitcoin's Transaction Processing. Fast Money Grows on Trees, Not Chains*. Cryptology ePrint Archive, Paper 2013/881. https://eprint.iacr.org/2013/881. 2013. URL: https://eprint.iacr.org/2013/881 (cit. on pp. 9, 16, 17, 19).

[69] N. Szabo. "Smart Contracts : Building Blocks for Digital Markets". In: 1996 (cit. on p. 9).

[70] *Transaction Execution Approval Language (TEAL)*. URL: https://developer.algorand.org/docs/get-details/dapps/avm/teal/ (visited on 2023-02-08) (cit. on p. 10).

[71] *Tendermint*. URL: https://tendermint.com (visited on 2023-01-03) (cit. on pp. 2, 9).

[72] *Transaction Family Overview*. URL: https://sawtooth.hyperledger.org/docs/1.2/app_developers_guide/overview.html (visited on 2023-01-13) (cit. on p. 10).

[73] R. Vassantlal, E. Alchieri, and A. Bessani. *Scaling Permissionless Blockchains using Consensus with Unknown Participants Model*. 2022 (cit. on pp. 24–26).

[74] Q. Wang et al. "SoK: Diving into DAG-based Blockchain Systems". In: *ArXiv* abs/2012.06128 (2020) (cit. on p. 19).

[75] Y. Wang et al. "Hybrid-chain: An Innovative and Efficient Mixed Blockchain Architecture". In: *Proceedings of the 2018 3rd International Conference on Electrical, Automation and Mechanical Engineering (EAME 2018)*. Atlantis Press, 2018/06, pp. 262–268. ISBN: 978-94-6252-538-2. DOI: 10.2991/eame-18.2018.55. URL: https://doi.org/10.2991/eame-18.2018.55 (cit. on p. 9).

[76] G. Wood et al. "Ethereum: A secure decentralised generalised transaction ledger". In: *Ethereum project yellow paper* (2014), pp. 1–32. ISSN: 1098-6596 (cit. on pp. 9, 10, 15–17, 30).

[77] L. Yang. "The blockchain: State-of-the-art and research challenges". In: *Journal of Industrial Information Integration* 15 (2019-09), pp. 80–90. ISSN: 2467-964X. DOI: 10.1016/j.jii.2019.04.002 (cit. on pp. 1, 6).

[78] G. Yu et al. "Survey: Sharding in Blockchains". In: *IEEE Access* 8 (2020), pp. 14155–14181. DOI: 10.1109/ACCESS.2020.2965147 (cit. on p. 20).

[79] H. Yu et al. "OHIE: Blockchain Scaling Made Simple". In: *2020 IEEE Symposium on Security and Privacy (SP)*. 2020, pp. 90–105. DOI: 10.1109/SP40000.2020.00008 (cit. on pp. 9, 16, 17, 21).

[80] M. Zamani, M. Movahedi, and M. Raykova. “RapidChain: Scaling Blockchain via Full Sharding”. In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. CCS ’18. Toronto, Canada: Association for Computing Machinery, 2018, pp. 931–948. ISBN: 9781450356930. DOI: 10.1145/3243734.3243853. URL: https://doi.org/10.1145/3243734.3243853 (cit. on pp. 9, 16, 17, 20, 26).

[81] S. Zhang et al. “Research on Self-Adaptive Consensus Method Based on Blockchain”. In: *2022 IEEE 14th International Conference on Advanced Infocomm Technology (ICAIT)*. 2022, pp. 292–297. DOI: 10.1109/ICAIT56197.2022.9862639 (cit. on pp. 3, 10, 24, 28).

[82] S. Zhu et al. “zkCrowd: A Hybrid Blockchain-Based Crowdsourcing Platform”. In: *IEEE Transactions on Industrial Informatics* 16.6 (2020), pp. 4196–4205. DOI: 10.1109/TII.2019.2941735 (cit. on p. 9).

I

Elaboration Work Plan Activities

The future work can be divided into 4 main activities with a total of 11 tasks, including the preparation of defense of the dissertation and is defined as follows:

Refinement of specifications and setup for prototype development

- **Task 1:** 22/Feb/2023 - 08/Mar/2023: Refinement of system design model and consolidation of design specification for system model components and software architecture. This includes the definition of development environment and planned micro and macro testbenches;

- **Task 2:** 01/Mar/2023 - 15/Mar/2023: Preparation and setup for the development environment, with the selected blockchain platform and related cloud-environment (as a Service Solution) that will be adopted for the first prototype.

HyFlexChain Platform prototype 1 development and testing

- **Task 3:** 15/Mar/2023 - 03/May/2023: Development of HyFlexChain in its prototype 1 phase, planned to include the first model of hybrid multi-consensus service plane using PoW and PBFT based mechanisms and related sybil-resistant committee election;

- **Task 4:** 03/5/23 - 24/5/23: Support for consensus-change transactions based on verifiable smart contracts in prototype 1;

- **Task 5:** 03/May/2023 - 31/May/2023: Experimental tests and validation of HyFlex-Chain prototype.

HyFlexChain Platform prototype 2 (final) development and testing

- **Task 6:** 31/May/2023 - 07/Jun/2023: Experimental tests and benchmark verifications related to the operation of consensus-change smart contracts in prototype 1;

- **Task 7:** 07/Jun/2023 - 12/Jul/2023: Development of HyFlexChain in its prototype 2 phase, with extended support of prototype 1 to support a PoS consensus mechanism in its consensus plane;
- **Task 8:** 12/Jul/2023 - 04/Aug/2023: Depending on the success of the plan until 19/July, it is open a possible extension and proof-of-concept for the inclusion of a PoET mechanism for the consensus plane of prototype 2. This is planned to be analyzed with PoET elected/selected committees with some members running the consensus plane in isolation in a hardware-SGX-enabled Intel Cloud node, and other members running in emulated software SGX-virtualized nodes. In contingency, this task can be suspended;
- **Task 9:** 05/Jul/2023 - 04/Aug/2023, 25/Aug/2023 - 31/Aug/2023: Experimental evaluation of HyFlexChain prototype 2.

Dissertation report writing and preparation for defense

- **Task 10:** 08/Mar/2023 - 29/Jul/2023: Report writing effort, conducted in the following evolutive way:
  - Chap. 1 - Introduction and Chap 2 - Background (08/Mar/2023 - 22/Mar/2023);
  - Chap. 3 - Related Work (17/May/2023 - 31/May/2023);
  - Chap. 4 - HyFlexChain platform: System model and Architecture (Jun/2023);
  - Chap. 5 - HyFlexChain Implementation (until 15/Jul/2023);
  - Chap. 6 - Experimental evaluations and validation (15/Jul/2023 - 15/Sep/2023);
  - Chap. 7 - Conclusions and future work (15/Sep/2023 - 22/Sep/2023);
  - Final report review and delivery: 01/Sep/2023 - 22/Sep/2023.
- **Preparation for defense:** 25/Sep/2023 - 11/Oct/2023: Preparation of presentation materials for the thesis defense, including presentation slides and publication of the GitHub-based prototype implementation.