Blockchain technologies and their application to secure virtualized infrastructure control

Nikola Božić

THÈSE DE DOCTORAT DE
SORBONNE UNIVERSITÉ

Spécialité

Informatique

École doctorale Informatique, Télécommunications et Électronique de Paris

Présentée par

Nikola BOŽIĆ

Pour obtenir le grade de

DOCTEUR de SORBONNE UNIVERSITE

Sujet de la thèse :
Blockchain technologies and their application to secure virtualized infrastructure control

soutenue le 18 Octobre 2019

Devant le jury composé de :

M. Al Agha KHALDOUN	Rapporteur	Professeur - Université Paris-Sud
Mme. Michele NOGUEIRA	Rapporteur	Associate Professor - Federal University of Paraná, Brazil
Mme. Thi-Mai-Trang NGUYEN	Examinateur	Maitre de Conferencés Habilité - Sorbonne Université
M. Stefano SECCI	Examinateur	Professeur - Cnam, Cedric
Mme. Virginie DOTTA	Encadrant	Directrice Virtual Infrastructure - SQUAD
M. Guy PUJOLLE	Directeur de thèse	Professeur - Sorbonne Université

Abstract

Blockchain is a technology making the shared registry concept from distributed systems a reality for a number of application domains, from the cryptocurrency one to potentially any industrial system requiring decentralized, robust, trusted and automated decision making in a multi-stakeholder situation. Nevertheless, the actual advantages in using blockchain instead of any other traditional solution (such as centralized databases) are not completely understood to date, or at least there is a strong need for a *vademecum* guiding designers toward the right decision about when to adopt blockchain or not, which kind of blockchain better meets use-case requirements, and how to use it. At first, we aim at providing the community with such a vademecum, while giving a general presentation of blockchain that goes beyond its usage in Bitcoin and surveying a selection of the vast literature that emerged in the last few years. We draw the key requirements and their evolution when passing from permissionless to permissioned blockchains, presenting the differences between proposed and experimented consensus mechanisms, and describing existing blockchain platforms. Furthermore, we present the B-VMOA blockchain to secure virtual machine orchestration operations for cloud computing and network functions virtualization systems applying the proposed vademecum logic. Using tutorial examples, we describe our design choices and draw implementation plans. We further develop the vademecum logic applied to cloud orchestration and how it can lead to precise platform specifications. We capture the key system operations and complex interactions between them. We focus on the last release of Hyperledger Fabric platform as a way to develop B-VMOA system. Besides, Hyperledger Fabric optimizes conceived B-VMOA network performance, security, and scalability by way of workload separation across: (i) transaction execution and validation peers, and (ii) transaction ordering nodes. We study and use a distributed *execute-order-validate* architecture which differentiates our conceived B-VMOA system from legacy distributed systems that follow a traditional state-machine replication architecture. We parameterize and validate our model with data collected from a realistic testbed, presenting an empirical study to characterize system performance and identify potential performance bottlenecks. Furthermore, we present the tools we used, the network setup and the discussion on empirical observations from the data collection. We examine the impact of various configurable parameters to conduct an in-dept study of core components and benchmark performance for common usage patterns. Namely, B-VMOA is meant to be run within data center. Different data center interconnection topologies scale differently due to communication protocols. Enormous challenges appear to efficiently design the network interconnections so that

the deployment and maintenance of the infrastructure is cost-effective. We analyze the structural properties of several DCN topologies and also present some comparison among these network architectures with the aim to reduce B-VMOA overhead costs. From our analysis, we recommend the hypercube topology as a solution to address the performance bottleneck in the B-VMOA control plane caused by gossip dissemination protocol along with an estimate of performance improvement.

Table of contents

List of Figures

List of Tables

Chapter 1

Introduction

According to National Institute of Standards and Technology (NIST) [1], "Blockchains are immutable digital ledger systems implemented in a distributed fashion (i.e., without a central repository) and usually without a central authority." Distribute stands for the fact that each peer maintains a copy of the ledger. Immutable stands for the fact that once appended into the ledger, the transactions are protected from any sort of tampering and falsification except where the majority of the network's effort are devoted to change the registry. Hence it makes data more accessible and manageable by different network participants rather then a centralized entity. Transactions are ordered and grouped into "blocks" which captures several transactions per block while respecting a given block size limit. Eventually peers synchronize to an exact copy of the blockchain throughout the network. The blockchain updating procedure needs a consensus, i.e., an agreement among the network peers. Consensus in the network refers to the process of achieving agreement among the network participants as to the correct state of data on the system. Consensus leads to all nodes sharing the exact same data. Therefore a consensus algorithm (i) ensures that the data on the ledger is the same for all network nodes, and (ii) prevents malicious actors from manipulating the data. The consensus procedure varies with different blockchain implementations. While the Bitcoin blockchain [2] uses a PoW-based consensus mechanism [3], other blockchains and distributed ledgers are deploying a variety of consensus algorithms belonging to two main classes: (i) Proof-of-X-based algorithms and (ii) Byzantine Fault Tolerant algorithms. Respecting the consensus imposed by blockchain network, a new block of transactions is appended with the hash of the previous block committed on the ledger, thus forming a hash chain of blocks. This ordered back-linked chain of blocks by way of hashing gives it the name blockchain. Therefore, a blockchain network is a distributed transaction system where all the peers share information in a decentralized, trusted and secure manner.

In recent years we witness how blockchains shift the way of how digital assets are designed to work as a medium of exchange to secure financial transactions, control the creation of additional units, and

verify the transfer of assets [4]. Cryptocurrencies use decentralized control as opposed to centralized digital currency and central banking systems [5]. Thus, a cryptocurrency such as Bitcoin is an application running on the blockchain network allowing parties to settle transactions quicker, resulting in faster movement of goods and services. Introduction of blockchain networks in businesses and enterprises started the shift towards new ground-breaking changes in the way they function and operate [6]. It would result in a brand new business models as well as a transformation of the existing business models. The exponential rate at which technology is evolving creates an imperative for organizations to deconstruct their value chain to gain a competitive advantage. Blockchains are widely regarded as a promising technology for what is called digital transformation. What attracts different stakeholders to move to a blockchain is the ability to automate business transactions using smart contracts [7]. Smart contract is a collection of rules to create and/or change the asset by way of different processes on network known as transactions. A transaction refers to a sequence of information exchange and related work (such as data state updating) that is treated as a unit for the purposes of satisfying a request and for ensuring data integrity. This transactions are shared and validated collectively by a group of stakeholders via blockchains. Such smart contracts help execute any kind of processes in an automated and trusted manner.

Blockchain networks can be classified based on the following criteria: (i) who are allowed to submit transactions, (ii) which peers are allowed to order transactions (including consensus), c) how are new clients/peers authorized to join the network. In a public or permissionless blockchain network, anyone can participate in the network without a specific identity. Such networks usually involve a native cryptocurrency or other economic incentives. Popular examples are Bitcoin [2] and Ethereum [8]. Such networks use lottery-based consensus protocols such as proof-of-work (PoW). With the introduction of permissioned blockchains, users may opt for its adoption by placing constraints and customizing the behavior of network nodes. While with classical blockchainsit is possible to build a completely open and decentralized system, permissioned ones allow only a limited number of users to have the right of validating transactions. Validators constitute a set of nodes that can be publicly elected or selected by a central authority. Limiting the number of participants in the validation procedure can grant significant scalability improvements by using appropriate consensus mechanisms. Moreover, protocol's changes (in both the blockchain data and consensus structure) made to support the execution of Turing-complete codes, facilitate the deployment of distributed applications ('dapps') based on smart contracts. However, since full-permissioned blockchains have many similarities with classic shared databases, there can be situations where such a complex architecture is not indispensable.

A permissioned blockchain network is operated by known entities forming a consortium where members are whitelisted and bounded by strict contractual obligations to behave "correctly". New peers can be added by permission from the existing peers or by way of network admin privileges [9], leveraging voting-based consensus protocols such as Practical Byzantine Fault Tolerance (PBFT) [10] to improve network performance when compared to pubic blockchains. There is no inherent need of a cryptocurrency in such networks. A private blockchain network is a special case of permissioned blockchain operated within a single entity.

In this dissertation, we study blockchain technologies from different perspectives. We aim to give to the reader a comprehensive tutorial about when to use blockchain, which solution to use, and how to use it, based on use-case requirements. Our work refers to a synthetic collection of information con-

cerning a blockchain technique, having the goal to provide the reader with quick and concise responses on the different details of the blockchain technique. We refer to it as "Blockchain Vadamecum". During the past few years, research societies along with industrial and governmental institutions intensively worked on DLT and blockchain, trying to understand better this paradigm and its place in today's market. This resulted in many publications and standardization activities as well. In the following, we provide the reader with a decision model to understand *When* to use the blockchain technology and *Which* type of blockchain suits a certain use case best. The decision model is characterized by two decision paths (When and Which paths) that can be traversed either consecutively or independently; the decision points can be both direct questions or trade-off points. Once decided the type of blockchain, we provide the reader with a complete list of the most popular blockchain frameworks in the market accompanied by details on their structure, operation and implementation allowing the reader to find the one that comes closest to her use case. As of our knowledge, our effort is unique in the purpose and the style, tackling these important questions in a more direct, expressive and thorough way than in existing works.

Further, we investigate blockchain application in a cloud orchestration applying the proposed vademecum logic. Virtualization servers run virtual machines in such a way that they can be dynamically migrated, duplicated, scaled in-out or up-down, turned on or off, accordingly to arbitrary orchestration policies, acting at one or multiple distinct computing resources, concurrently. The interconnection network resources between servers can also be made programmable to serve dynamic reallocation tasks and the necessary abstraction and decoupling to IaaS (Infrastructure as a Service) communications (cloud access and inter-VM traffic) and computing facilities. A fully virtualized infrastructure allows a decoupling between the operating systems of the VMs and the operating system of the virtualization server physical machine. In such a case, the physical machine possesses a VM Manager (VMM), or hypervisor, running with kernel privileges to allow the management of multiple isolated VMs. The VMM offers specific functions to execute several distinct operating systems; the VMM receives instructions about how to create, destroy, modify, migrate VMs and reallocate resources either manually or automatically, from the command line interface or using orchestration protocols. When an external orchestrator is in charge of managing multiple physical machines acting as virtualization servers, there is the need to secure the communication between the orchestrator and the physical machine VMMs, in order to guarantee that a VM management command is authorized and legitimate. In fact, these architectures are very sensitive to attacks that can come from different horizons. For example, a virtual machine can be created by an attacker to run in a server and used to perform external DDOS attacks, and also internal attacks against data integrity and confidentiality. The goal of the Virtual Machine Orchestration Authentification (VMOA) method we describe in this dissertation is to protect and secure virtual machines, which in essence are software that is difficult to protect. Usually, authentication is achieved through third party mediation. The third party must be trusted, yet it also represents a vector of attacks against the integrity of orchestration commands. Having a fully decentralized authentication logic is therefore a rising requirement for such environments. In this respect, blockchain is a technology meant to enable a new ecosystem for digital asset transactions in a decentralized fashion. We aim to present how blockchain can be used to secure cloud/NFV orchestration operations and in particular to enhance the authentification of orchestration commands in the life cycle of cloud services. We design B-VMOA (bockchain VMOA) model using Hyperledger Fabric technology [11]. Hyperledger Fabric is currently deployed in more than 400 proof-of-concept and production distributed ledger systems

across different industries and use-cases [12].

We conceived B-VMOA to break vertical separation between orchestration management and virtualization servers only used to host VMs. B-VMOA breaks this separation by way of delegating to multiple stakeholders (virtualisation server) voting rights along with dispatching orchestration logic to servers which are no longer used just as hosting machines. By way of B-VMOA, we propose a new ecosystem to be run within private data centers to secure virtual machine orchestration. It is important to notice that B-VMOA omits direct communication between cloud orchestrator and VMM as in the centralized legacy solution. Hence B-VMOA system design brings the following features to cloud orchestration: (i) Orchestration decentralization, (ii) Orchestrator - VMM communication security by way of preserving instructions integrity, authenticity, non-repudiation and immutability, and (iii) orchestration auditability. The mix of the above features qualifies the conceived B-VMOA system at a quite high level of dependability, differentiating it from the legacy network orchestration solutions.

Although blockchain networks provide immense benefits, there are rising concerns about their achievable performance and whether they would match with the mainstream information technology (IT) and the required use-case needs. Unlike pubic blockchains, permissioned blockchains can be designed to use more efficient consensus mechanisms such as PBFT [13], resulting in much higher throughput and lower latency with less computation, bandwidth, and storage requirements, which is indeed a need for a private cloud orchestration. To validate our B-VMOA model we analyze system performance as a function of various configurable parameters. We deploy B-VMOA using Hyperledger Fabric (HLF) platform in private cloud environment. As the performance of blockchain platform is a major concern for B-VMOA application, we present an empirical study to characterize system performance and identify potential performance bottlenecks. Due to our B-VMOA system design, the system-level performance metrics directly impact B-VMOA performance by way of (i) transaction latency, i.e., the time elapsed between transaction request and when the transaction is confirmed by submitting client as a function of all delays during the transaction flow and (ii) transaction throughput, expressed in transactions per second (TPS), which represents the rate of appended transactions in a certain period.

As our last contribution, we study and analyze Data Center Network (DCN) topology impact on scaling up B-VMOA control plane. Due to B-VMOA nature and purpose, we consider the private cloud intra-DC deployment of B-VMOA. Large-scale data center network (DCN) provide the core infrastructure to meet computing and storage requirements for both enterprises and cloud-based services. Nevertheless, B-VMOA is meant to be run within a data center, hence the B-VMOA hosts are actually servers running B-VMOA agents. Different data center interconnection topologies scale differently due to communication protocols. Besides, Hyperledger Fabric optimizes B-VMOA network performance, security, and scalability by way of workload separation across: (i) transaction execution and validation peers, and (ii) transaction ordering nodes. It is essential to have a secure, reliable and scalable data dissemination protocol to ensure data integrity and consistency. To meet these requirements, a gossip data dissemination protocol is used. Overhead of B-VMOA control plane traffic is directly impacted by constant ‘heartbeat’ message unicasting between B-VMOA peers. Furthermore, to support the growing cloud computing needs, the number of servers in today’s data centers is increasing exponentially, thus resulting in enormous challenges to efficient network design for interconnecting these servers so that the deployment and maintenance of the infrastructure is cost-effective. We seek to minimize the

overhead between multiple B-VMOA peers by properly choosing the most adequate interconnection topology. To ascertain our topology choice, we need to consider several aspects, among which the control overhead problem is the most important one. Namely, we need to consider what properties affect B-VMOA DCN performance and how to minimize the gossip overhead cost. We analyze the structural properties of several DCN topologies and also present some comparison among these network architectures with aim to reduce B-VMOA overhead costs. More specifically, our analyses integrates the following features: first, it exploits gossip control plane overheads; second, it proposes a hypercube topology for B-VMOA DCN to reduce the transmission overhead. The evaluation results show that hypercube B-VMOA DCN can achieve higher performance than legacy fat-tree and DCell, with preserving servers' CPU and memory resources by way of packet processing on the smart NIC and saving infrastructural cost due to its switchless structural design.

This dissertation is organized as follows: In Chapter 2, we provide an overview of blockchain technology. This chapter explores all the layers characterizing the blockchain architecture (i.e., network layer, data model layer, execution layer, consensus layer and, application layer), particularly focusing on those that are crucial for deciding whether to adopt the technology or not and. In Chapter 3, we start our vadamecum. We provide the reader with a decision model to understand *When* to use the blockchain technology and *Which* type of blockchain suits a certain use case best. Once decided the type of blockchain, we provide the reader with a complete list of the most popular blockchain frameworks in the market accompanied by details on their structure, operation and implementation allowing the reader to find the one that comes closest to her use-case. In caper 4, we present the VMOA blockchain we conceived to secure virtual machine orchestration for cloud computing and network functions virtualization systems. We further develop the vademecum logic applied to cloud orchestration and how it can lead to precise platform specifications. We design and implement B-VMOA Proof of concept (PoC) with the aim of verifying its practical potential. In Chapter 5, we present an empirical study to characterize system performance and identify potential performance bottlenecks. In Chapter 6, we analyze Data Center Network (DCN) interconnection topology impact on scaling up B-VMOA control plane. Finally, we conclude this thesis in Chapter 7.

Chapter 2

Overview of the blockchain

Summary

Blockchain can be regarded as a quality leap from the distributed database technology [14] studied since the seventies, which consists in a transaction database shared by different users. Generally, Distributed Ledger Technologies (DLTs) are designed to deal with black database in the form of data shared in a distributed manner, and blockchain represents one possible DLT to do it (see Fig. 2.1). Blockchain allows sharing a ledger of transactions that are read, validated and stored in a chain of

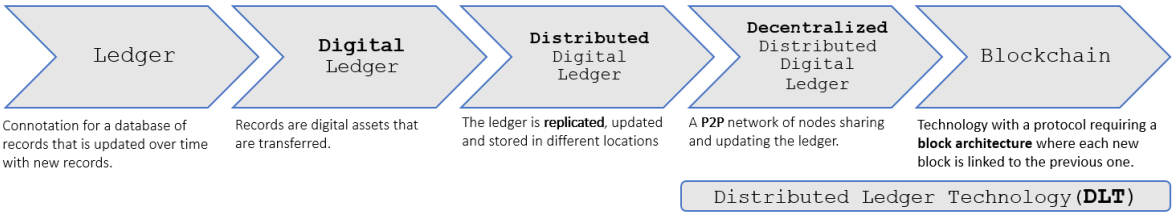

Figure 2.1: DLT evolution: from the traditional ledger to blockchain.

blocks. Systems based on the blockchain technology work in a distributed manner, involving multiple agents or participants that ought to be independent of each other, and which can use peer-to-peer communications (P2P) to structure themselves into a network collectivity. In contrast to legacy client-server architectures [15], P2P network nodes do not always have specific roles, a fixed hierarchy; roles may not exist, or may change over time depending on the actual operation behind a communication, i.e., a blockchain transaction. The adoption of P2P as communication paradigm adequately supports the goal that resources are shared and dispersed over a network which by construct forbids the existence of providers or servers centralizing tasks. The result is a decentralized ecosystem with no central authority [16]. Blockchain can hence be used in diverse sectors with several applications. However, it is crucial for users to understand whether the technology fits the problems that they are aiming to solve or not. There may be cases where the price paid for decentralization results commercially unreasonable [17; 18], and this is one of the reasons why regular databases are still widely used. Fundamental bricks in the design of a blockchain technology are as follows: (*i*) communications and transaction data storage are regulated by cryptographic security, network nodes have to agree on both the validity and the order in which transactions are listed in the blockchain, (*ii*) distributed consensus protocols solve these issues in a scenario where each node comes to vote. The first example of such a blockchain is Bitcoin, proposed in 2008 by its anonymous identity [2]. The Bitcoin behavior traces what can be defined as the 'classical' blockchain, consisting in a *permissionless* blockchain alternative enabling a digital, distributed and decentralized payment system.

The Bitcoin blockchain is structured in order to protect the ecosystem against attacks launched by malicious or simply rational nodes of the network. As attackers may exploit blockchain vulnerabilities in several ways to achieve a privileged position on the network, the Bitcoin blockchain was designed primarily for preventing the so called *double spending* and *Sybil* attacks, without addressing other important aspects [19; 20] such as: (*i*) complete anonymity – Bitcoin provides its users with only pseudonymity; (*ii*) blocks have a limited size, limiting both the number of transactions that can be validated with one block and the number of validated transactions per second (tps) – Bitcoin has a 1 MB limit with a transaction rate ranging from 3.3 to 7, incomparable to current credit card systems managing tens of thousands tps [21]; (*iii*) eco-sustainability of the validation process – Bitcoin is designed to make it difficult to validate blocks with validating agents or *miners* required to solve computationally heavy crypto-puzzles, and therefore consuming energy. As a consequence, even if Bitcoin remains the most successful cryptocurrency in circulation, a large number blockchain-based cryptocurrencies have been defined – as of [22], more than 50 alternative cryptocurrencies exist. Some of these 'Altcoins' [23] can guarantee anonymity, solve the energy consumption issue, or rely on a *permissioned* blockchain, accessible only to authorized nodes, in order to offer a more scalable and fast system.

Going beyond the Bitcoin case, the general blockchain technology aims at assuring the third party

benefits such as integrity, authenticity, security and non-repudiation in a distributed and decentralized environment. In addition to auditability and transparency, it offers immutability [1] (stored transactions are not editable once published) and pseudonymity [24] to its users. Besides being evident for currency systems, these features are useful for any transactional system that is to be used by multiple independent trustless parties. With the introduction of permissioned blockchains, users may opt for its adoption by placing constraints and customizing the behavior of network nodes. While with classical blockchains it is possible to build a completely open and decentralized system, permissioned ones allow only a limited number of users to have the right of validating transactions. Validators constitute a set of nodes that can be publicly elected or selected by a central authority. Limiting the number of participants in the validation procedure can grant significant scalability improvements by using appropriate consensus mechanisms. Moreover, protocol's changes (in both the blockchain data and consensus structure) made to support the execution of Turing-complete codes, facilitate the deployment of distributed applications ('dapps') based on *smart contracts*. However, since full-permissioned blockchains have many similarities with classic shared databases, there can be situations where such a complex architecture is not indispensable.

Although the blockchain technology is covered by many surveys so far, few ones analyze it in its entirety without dwelling on the permissionless part rather than on the permissioned one. This chapter explores all the layers characterizing the blockchain architecture (i.e., network layer, data model layer, execution layer, consensus layer and, application layer), particularly focusing on those that are crucial for deciding (*i*) whether to adopt the technology or not and, (*ii*) which of the available blockchain solution come closest to a certain use-case.

2.1 Distributed Ledger Technology (DLT)

Looking back to the last half century of computer technologies, architectures and related design practices, we can observe a fluctuation trend between the centralization and subsequent decentralization of computing resources such as computing power, storage, infrastructure, protocols, and code. Mainframe computers are largely centralized, housing most of computing resources. Today, computational capabilities are distributed on the clients, the clients facilities, and on distant servers. This approach gave rise to the 'client-server' architecture which supported the development of the Internet and relational database systems. Massive data sets, originally housed on mainframes, can move onto a distributed architecture, with data replicated from node to node, or server to server, and subsets of the data can be accessed and processed on clients, and then, synced back to one of the servers.

Over time, Internet and cloud computing architectures enabled global access from a variety of computing devices; whereas mainframes were largely designed to address the needs of large corporations and governments. Even though such an Internet/Cloud architecture is decentralized in terms of hardware, it has given rise to application-level centralization. Currently, we are witnessing the transition

[1] With the term immutability we refer to the concept of "immutability unless the adversary thresholds exceedance": a permissionless blockchain become mutable whenever the majority of the network efforts are devoted for the purpose of replacing validated blocks, a permissioned one can become mutable following an attack by $\frac{1}{3}$ of the network (see Appendix).

from centralized computing, storage, and processing to decentralized architectures and systems. The DLT is the key innovation making this shift possible. Some distributed systems (e.g., permissionless blockchains) aim to give the control of digital assets to end users without the need for intermediate nodes. Others (e.g., permissioned blockchains), attempt at maintaining a logical centralization of some information while adopting a decentralized architecture. Not all DLTs make use of a block architecture and can therefore be defined as 'blockchains' (e.g., *The Tangle* and *BigchainDB* [25; 26]). First, we familiarize the reader with the terminology. Afterwards, we focus on the blockchain participation modes characterizing our vademecum.

2.1.1 Terminology

- *A distributed ledger* is a type of digital data structure residing across multiple computer devices, generally at geographically distinguished locations [27].
- *Distributed Ledger Technology (DLT)* designs a type of technology enabling storing and updating a distributed ledger in a decentralized manner. As shown in Fig. 2.1, the blockchain and all its variations belong to the spectrum of DLTs. While distributed ledgers existed prior to Bitcoin, the Bitcoin blockchain was novel in that since marking the convergence of a set of existing technologies (including timestamping of transactions, P2P networks, cryptography, and shared computational power) and enabling data sharing and storage without entrusting any central party for the ledger maintenance. DLTs consist of three basic components:
  1) a data model that captures the current ledger state;
  2) a communication language defined by transactions that change the ledger state;
  3) a protocol used to build consensus among participants around which transactions are accepted by the ledger and in which order.
- *A blockchain* is a P2P DLT structured as a chain of blocks, forged by consensus, which can be combined with a data model and a communication language enabling smart contracts and other assisting technologies. Cryptography lets blockchains overcome former DLTs by offering secure data-transmission and by enabling records immutability, in a decentralized environment (see Appendix 2.1.2.2). Hence, a blockchain is an immutable read-only data structure, where new entries (blocks) get appended onto the end of the ledger by linkage with the previous block's 'hash' identifier.

The collection of these features can be used to build a new generation of transactional applications that establish trust, accountability, and transparency at their core, while streamlining business processes and legal constraints. In all DLTs, there is an initial record - in a blockchain it is called a *genesis block*. Each block includes one or more transactions. Connecting to a blockchain involves users connecting to this distributed ledger via, typically, an application. The blockchain ledger consists of digital transactions representing interactions between nodes of a P2P network.

- *Transactions* are individual and indivisible operations that involve exchange or transfer of digital assets. The latter can be information, goods, services, funds or set of rules which can trigger

another transaction.

- *Blockchain nodes* are computing device connected to the blockchain that support the network by maintaining a copy of the ledger. Records replicas are stored by *full nodes* which verify blockchain data integrity. There can be nodes that, when connecting to the blockchain, do not download the whole ledger but just a subset of it; these *lightweight nodes* – served by full nodes allowing them to transmit their transactions to the network – download the headers of all blocks on the blockchain in order to verify only if a transaction has been included in a block. Whenever blockchain nodes exchange assets via transactions in the network they are considered as *blockchain users*. In order to transact with the network peers [2], they generate a cryptographic key-pair (see Appendix .1). If the private key is used to sign transactions, the public key is the one identifying the user(s) *address* storing exchangeable assets (e.g., addresses with tokens defined as accounts or wallets).

Blockchain transactions are grouped into blocks, and there can be any number of transactions per block while respecting a given block size limit. Nodes on a blockchain network group up these transactions and send them throughout the network. Eventually peers synchronize to an exact copy of the blockchain throughout the network. The blockchain updating procedure needs a consensus, i.e., an agreement among the network peers.

- *Consensus* in the network refers to the process of achieving agreement among the network participants as to the correct state of data on the system. Consensus leads to all nodes sharing the exact same data. Therefore a consensus algorithm (*i*) ensures that the data on the ledger is the same for all network nodes, and (*ii*) prevents malicious actors from manipulating the data.

The consensus procedure varies with different blockchain implementations. While the Bitcoin blockchain uses a *PoW*-based consensus mechanism, other blockchains and distributed ledgers are deploying a variety of consensus algorithms belonging to two main classes: (*i*) *Proof-of-X*-based algorithms and (*ii*) *Byzantine Fault Tolerant* algorithms. We elaborate about consensus algorithms used in DLT in Section 2.3.

Early blockchain-based systems were meant for managing digital currencies. However, a generic DLT can fit any digital asset exchange requirement. Contractual aspects of an exchange, involving nodes' rights and obligations, can be digitalized and controlled by proper digital (smart) contracts.

- A *smart contract* is a computer program that executes predefined actions when certain conditions within the system are met. Smart contracts provide the transactions language allowing the ledger state to be modified. They can facilitate the exchange and transfer of any asset (e.g. shares, currency, content, property). They reside into the blockchain structure and are triggered along with transactions. Smart contracts can be imagined as digital protocols used to facilitate and enforce the negotiation of a legal contract. Actions carried out by trusted third-parties during a trade are replaced by pieces of code.

[2]The term "peer" denotes those blockchain nodes that are directly connected. Nodes that are initially alone seek to establish new connections with a certain number of peers (e.g., 8 for Bitcoin) in order to be part of the network. The terms node and peer are therefore interchangeably used.

Having acknowledged that blockchain ledgers fit within a wider spectrum of technologies, our contribution focuses on the analysis of blockchain systems characterized by a permissionless or permissioned participation mode.

2.1.2 Blockchain Structure and Features

A fundamental element beyond the innovation brought by blockchain to the DLT ecosystem is its intelligent mix of encryption techniques [28] in data storage – preserving block structure through timestamping [29] – and in transacting – authenticating transfers with digital signatures. A blockchain ledger consists of a history of validated digital transactions collected in blocks; each block of transactions is linked to the immediately previous one (known as *parent block*) through a hash value; hence by traversing the transactions ledger one can trace back the genesis block, which has no parent block and contains the first processed transactions in the blockchain history. Cryptography characterizes the technology and attributes important properties to it.

2.1.2.1 Data structure

A block is the junction of (*i*) an **outer header** identifying the blockchain and specifying the size of the block, (*ii*) a **block header** – containing all the information on the block validation and on its parent block – and (*iii*) a **block body** – consisting in a list of transactions and a transaction counter. While the precise structure of a block varies from one blockchain to another, each blockchain is identified by the *magic number* [3] which is included in any block of transactions at the beginning together with the *blocksize* field reporting the maximum number of bytes in a block. The block header should include for every blockchain system, as in Fig. 2.2, the following elements (whose order can vary from one blockchain to another one):

- Block version number: it refers to the blockchain protocol and hence the used consensus algorithm followed by the (majority of the) nodes at the moment of the block validations.
- Parent-block hash: it is the output of the hashing function with the previous block header as input.
- Nonce: it is a string of fixed length crucial in the validation process (Section 2.2.3).
- Timestamp: it indicates the time elapsed since a predefined instant.
- Merkle tree root: it is the hash value descending from a hash tree procedure (patented in 1989 [30]) applied to the transactions present in the block body; transaction informations are iteratively hashed in pairs as showed in Fig. 2.3 (if the number of transactions is odd, the last transaction, hashed or not, in the list is duplicated).

The hash of the block header serves as a link to future new blocks on top of it. The block body consists of all the transactions involved in the Merkle root calculation and of a transaction counter

[3] The magic number consists in a data-structure identifier characterizing the different blockchain protocols (i.e., 0xD9B4BEF9 is the magic number identifying the Bitcoin blockchain)

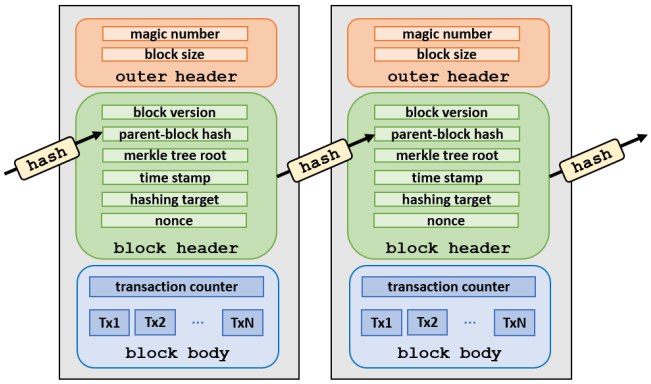

Figure 2.2: Representation of a blockchain structure.

providing the total number of transactions contained in the block. Note that the block size limit has a direct effect on the number of transactions that can be included in the block body.

2.1.2.2 Blockchain features

Thanks to the explanations of the previous paragraphs, we can now highlight six fundamental blockchain features, which are obviously dependent upon each others:

- *Decentralization*: DLTs enables P2P data sharing and storage without entrusting the ledger maintenance to any central authority. It does not mean completely cutting out intermediaries that validate transactions (*disintermediation*) like permissionless blockchains do, but rather decentralizing them along with their roles.
- *Immutability*: while shared ledgers allow data manipulation by a central authority, distributed ledgers working with replicated information protect data from any sort of tampering and falsification; except in situations where the majority of the network's efforts are devoted to change the registry [31] (e.g, the Ethereum DAO fork [32]) or where the adversary thresholds are exceeded (see Appendix .2.4). Data immutability makes data accessible and manageable by different en-

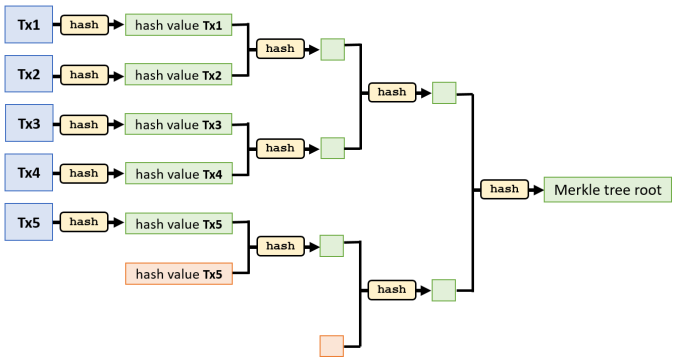

Figure 2.3: Merkle hash tree procedure example: duplicated (hashed) transactions are marked in orange.

tities that do not trust each other.

- *Integrity, Authenticity and Non-Repudiation*: the data hashing grants that data is not modified during its transmission (i.e., integrity). Moreover, the origin of a transaction can be ascertained by the senders' public key dissemination, while the evidence of the sending action is represented by the data signing procedure involving the private key (i.e., authenticity and non-repudiation). Blockchain signing scheme combining asymmetries cryptography and data hashing is presented in Appendix .1.
- *Auditability*: Transactions in blockchain systems must be validated and verified thus, each data transfer should be visible to all blockchains participants in its entirety. In this way, all blockchain operations are traceable via audits. Users accessing the first generation blockchains can see the data ledger in its entirety. Indeed, recent implementations enable multiple ledger to be isolated and maintained within the same blockchain system via private channels. Nevertheless, ledgers data is visible to all channel participants, thus the auditability is satisfied at channel level.

The mix of the above features qualifies the technology at a quite high level of dependability, differentiating it from the classic distributed database. Blockchain features result strictly correlated with the consensus mechanism in use.

2.1.3 Permissionless and permissioned participation modes

In conventional central data storage systems, only a single entity, the owner or the administrator, keeps a copy of the database. Consequently, this entity controls what data is contributed and what other entities are permitted to contribute. With the advent of DLT this radically changes in favor of distributed data storage where multiple entities hold a copy of the underlying database and are naturally permitted to contribute. Data is replicated for all entities participating in a distributed ledger in a network of so-called peers. Due to distributed data storage, the difficulty arises to ensure that all nodes agree upon a common truth, i.e., the correctness of a ledger, as changes made by one node have to be propagated to all other peer nodes in the network. The result of arriving at a common truth is referred to as consensus among nodes.

With respect to accessing the blockchain network, there are two main modes of operation: *permissionless* and *permissioned* – it is worth noting that in the literature, these are often referred to as public and private blockchains, respectively, but we use in this article a more precise taxonomy as explained hereafter. The same division is adopted regarding the participation to the ledger maintenance procedures, i.e., the possibility to modify (update) the network state. In the first mode, participation is public and open-access: anybody is allowed to participate in the network and in the consensus process [33]; this mode is the one adopted by first generation blockchains (e.g., Bitcoin). On the other hand, if participation is permissioned, participants have either restrictions on writing (validation) rights only, or on both reading (access) and writing rights. In the first case, permissions concern the participation to the phases of the transaction journey (see Section 2.2) amending the log; any modification of the transaction ledger is entrusted to a selected set of nodes. Instead, the so-called *full-permissioned* blockchains select participants in advance and restrict any sort of activity in the network to these only.

The participation mode differentiates between decentralized blockchain-based ledgers and those that additionally offer disintermediation namely, that cut out any middleman (i.e., permissionless blockchains). It is worth stressing that in permissionless blockchains anyone with an Internet connection can join the network, as well as write and read transactions; this is why permissionless, public and open-access are terms used interchangeably to refer to such technologies. Participants here are pseudonymous, which is not preventing malicious nodes to act within the network. Contrariwise, full-permissioned blockchains, reduces these security risks by whitelisting authorizations to join the network. In this way, rather than displaying the transactions record to the entire Internet community, transactions remain visible only to a private network of nodes.

The differentiating points in the previous two paragraphs allow us to support what authors in [34] propose, i.e., differentiating full-permissioned blockchains from those allowing anyone to read the blockchain state, denoted in [34] and in the following as *open-permissioned blockchains*.

With respect to the nature of participants, permissioned blockchains can be further classified in 'private' blockchains – where the participants are within the same organization – and 'consortium' blockchains – where the permissioned blockchain is deployed among several organizations (consortium). A consortium blockchain represents a joint effort of several entities sharing a common goal or business need. Furthermore, 'private' and 'consortium' attributes can be linked to the blockchain governance system. There are some developed by a single enterprise, and others by a joint effort of several contributors (e.g., Corda and Hyperledger [35; 36]). The latter, for instance, is a cross-industry project led by the Linux Foundation to advance blockchain technology by coming up with common standards. The participation mode has a braking impact on the decentralization trend in distributed consensus, as we develop in Section 2.3.

2.1.4 Related Work

The blockchain technology is surveyed in many articles published after 2014. About DLT, a term coined in [37] in 2016, many works also address the comparison between blockchain and previous technologies.

Most of the articles focus on cryptocurrency blockchain-based systems, with different focus on all their aspects. *Tschorsch* and *Scheuermann* [38] present a complete work covering all aspects of the Bitcoin protocols, addressing security, network and privacy aspects. *Conti* et al. [19] survey security and privacy issues of the Bitcoin blockchain, while *Khalilov* et al. [24] focus on surveying techniques enhancing anonymity and privacy in blockchains based on PoW consensus with an emphasize on Bitcoin. Network aspects and related attacks are surveyed by *Neudecker* and *Hartenstein* [39]. Mining procedures for cryptocurrency are presented by *Mukhopadhyay* et al. [40]. Consensus mechanisms constructed using the Bitcoin architecture are surveyed by *Sankar* et al. [41] and *Garay* et al. [42].

Besides crypotocurrency-oriented works, general technology aspects are also covered by other articles presenting differences among permissioned and permissionless blockchains. *Zheng* et al. [43; 44] presented a key features overview for blockchains, covering both public and private modes. Consensus protocols in blockchains are surveyed in [45] and [46], the latter focusing on consensus evolution

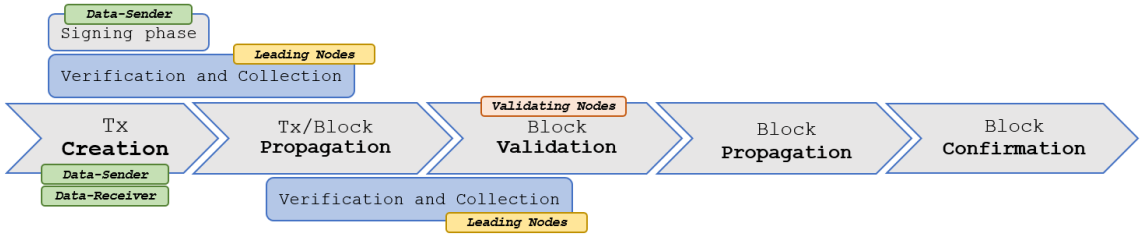

Figure 2.4: The Transaction Journey. Once created the transaction is signed by the data-sender. Verification checks are performed upon block creation by the leading nodes. Transaction can be collected in a block either before being transmitted to the validating nodes or afterwards. The block of transactions is then validated, propagated and confirmed.

from the Bitcoin blockchains to the private ones. *Wang* et al. [47] presented the design methodologies for consensus incentive mechanisms in blockchain. *Li* et al. [48] surveyed attacks against blockchain networks, while security issues and challenges are briefly presented in [49].

Furthermore, a comprehensive overview of blockchain applications and use-cases is provided by [50]. Generic IoT (Internet of Things) blockchain applications are presented by *Ferrag* in [51]. Recently published, two tutorials [52; 53] present comparisons between permissionless and permissioned blockchains relating them to technology use-cases.

Our article differs from the state of the art in that it aims at exploring all DLT aspects with the purpose of providing readers with all the instruments and key aspects for deciding which technology to use for their business. The evolution from permissionless to permissioned blockchains is presented along with their consensus protocols, features and properties, in order to let users choosing the most suitable blockchain. Unlike the decision patterns proposed so far [1; 54], our work is not only presenting a sequence of direct decision points (i.e., nodes of a decision tree where one decision excludes the other) leading to a final state. Our work is also focusing on those decision points where the reader has to make compromises between strongly related features (i.e., *trade-off points*). Moreover, we confront the technology with traditional database technology for the purpose of highlighting those cases in which deploying such a complex block architecture is not worth the effort.

2.2 Journey of a transaction

Generally, transactions in blockchain are not strictly financial and do not just carry and store transaction data. Hence, the usage of blockchain transactions is not limited to the simple assets exchange, but it also covers the execution of computing instructions such as *storing*, *querying* and *sharing*. Every transaction, once validated, is placed in a new block which is added in the transaction ledger and linked to the previous one. This results in an update of the system state and of users' local copy of the blockchain. Whenever a user aims at interacting with another one in the network, one or multiple transactions are created, propagated, validated and confirmed by the network. Each blockchain-based system differs from the others by the way in which the steps of the 'transaction journey' are performed. This journey starts at the moment in which the transaction is created and ends when the transaction is

recorded in the blockchain. Four crucial steps of the journey of a blockchain transaction can be identified as illustrated in Fig. 2.2:

2.2.1 Transaction Creation

Whenever a user aims at interacting with another one in the blockchain network, a transaction takes place. In general, a transaction indicates to the network that a user has authorized a data flow. Hence, it has to be properly constructed for its purpose before its propagation.

Firstly, the sender user has to build a transaction *proposal* specifying all the criteria according to which the information can flow to the transaction receiver(s). All blockchain transactions must specify the destination of the operation, in most cases provided with a unique transaction identifier. Moreover, a transaction field reporting the entity of the transfer must exist; i.e., in the case of cryptocurrencies a certain amount of tokens is specified in the amount field of the transaction. Blockchain technology supports the presence of both multiple *origins* and multiple *destinations*; a transaction sender may have more receivers and vice-versa.

The transaction proposal must be signed by the sender(s) to prove the ownership of the address(es) instantiating the transaction. Blockchain-based systems use digital signatures as authentication methods (as presented in Appendix .1). Once signed, the transaction can move on to be propagated in the P2P network. Privacy-preserving blockchains – trying to hide the source, the destination and the entity of a transaction – can make use of temporary addresses and special cartographic tool to sign and encrypt transactions before the propagation [24].

The data model of a blockchain transaction differs depending on the system implementation and its business application. For instance, the Bitcoin protocol imposes the transfer of *Unspent Transaction Outputs* (UTXOs [55]), presented hereafter. Post-Bitcoin data models have evolved in two different ways.

First, blockchains moved to the adoption of an *account-based model*, making use of a completely new transaction syntax (Turing complete) [8] and resulting more 'smart contract friendly'; Ethereum is one of the so-called 'second generation' cryptocurrencies [56] adopting this record-keeping model. Subsequently, blockchains' intention was to maintain the original Bitcoin data-structure along with its improvement proposals [57] to which integrate the benefits of an account-based model. General blockchains, going beyond cryptocurrencies and digital assets, may adopt basic models supporting smart contract execution. Offering more and more general operations corresponds to a data model supporting more and more complex logic, hence overcoming both the account and the UTXO models. Blockchain-based systems of this type adopt a *key-value data model* (also called table-data model) where the blockchain registers its state as data-tuples that can be updated. We present in the following these different models in more details; benefits and drawbacks are summarized in Table 2.1.

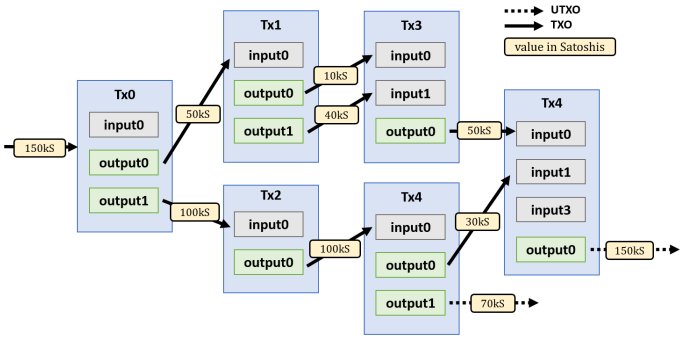

Figure 2.5: An example of UTXO-based transfers in Bitcoin.

2.2.1.1 UTXO model

This record-keeping model associates value to users' addresses as 'unspent' transaction outputs, i.e., cryptocurrency amounts that may be spent in the future through the construction of other transactions; UTXOs become inputs of a 'spending transaction' transferring the value previously received to another blockchain user. Transactions outputs (TXOs) can only be spent (i.e., transferred) once. Blockchain addresses keeps track of the received UTXOs; their sum corresponds to the address balance.

A peculiarity of the UTXO model is that transactions inputs and outputs must match; namely the entire value of the TXOs received in a prior transaction has to be transferred in order to be spent. More precisely, a user aiming at transferring data to another one does nothing more than 'endorsing' a previous received UTXO. Users unlock an output, appropriately transform it and generate a new one; the procedure, resembling the "*compare-and-swap*" (CAS) instruction in computer science, forces a synchronization in data accessing [58]. The problem arises whenever a user has no intention of spending the entire value of a TXO. The issue is solved with the proper use of multiple outputs; the system creates a transaction with two different outputs: (*i*) one destined to the receiving user, transferring the aimed value (lower in relation to the TXO) and, (*ii*) one transferring the difference back to the sender in the form of a new UTXO. In this way, the inputs value corresponds to outputs value. The UTXO model is designed in such a way that each UTXO has to be transferred/spent in its entirety as input of another transaction. That is why operations on UTXO-based blockchains are so reminiscent of exchanging cash. Fig. 2.5 shows how UTXO works in the Bitcoin blockchain marking the difference between TXOs and UTXOs.

The state of the whole blockchain is represented by the UTXOs state. Each transaction includes the state of the new output and in order to be updated it has to be included as input of a second transaction. This implies high verification, duplication and transmission costs. Because of these drawbacks, UTXO model forces blockchains to limit the amount of operations impacting the system state.

Bitcoin adopts a transaction structure with three basic fields: (*i*) the value to be transferred, (*ii*) a short script specifying the conditions under which the value can be redeemed (i.e., the Locking Script or Redeem Script [59]) and (*iii*) a *witness* field to unlock the previous transaction output. The script locks the transaction until spending conditions are met, i.e., when a witness is provided. The

approach works for simple transactions ("Pay-to-PubKeyHash" [28]) or simple contracts involving a small number of transactions locked with proper locking scripts ("Pay-to-ScriptHash" [60]), however it results not suitable for slightly more complex operations contemplated with smart contracts. UTXO-based applications in Bitcoin should limit the number of transactions involved, because of both the cost in terms of computations required to find a PoW (a *golden nonce* [61]) validating a transaction, and the scripting language supported by the model which is Turing incomplete [62].

2.2.1.2 Account-balance model

This model results more intuitive in keeping track of the balance of each account as a global state of the blockchain. State replication completely overcomes the concept of transaction input and output; more precisely, the blockchain state is an outcome of a transaction. Once a transaction is executed the states of the accounts involved in the transferring are updated.

There are different options for creating a transaction depending on the output and the finality; regular transactions between users have to simply specify the receiving account(s) and the entity of the transfer, while transactions dealing with contracts present rather complex structures. In terms of data model, a smart contract consists of a collection of standard transactions presenting locking conditions: contracts on the blockchain are created as transactions between addresses and they can be executed thanks to *triggering* transactions. For instance, Ethereum works with different types of accounts: Externally Owned Accounts (EOAs) holding only its balance, and Contract Accounts (CAs) holding the code of a smart contract and keeping an internal state. Once a transaction in a contract or a regular one is executed, the ledger is updated together with its state.

Contrary to UTXO-based blockchain, account-based systems have to deal with several security issues. First of all, the account model is not not immune to double-spending practice. Hence, it is necessary to secure the blockchain adopting this record-keeping model, preventing the same transaction being submitted more than once. Moreover, an anonymity issue arises when accounts are reused; the account model gives preference to balance updates rather than new account creation.

2.2.1.3 UTXO$^+$

The idea beyond the UTXO$^+$ model is to maintain the UTXO structure, to which appropriate changes are made in order to obtain the same benefits granted by the account-based models. There is no notion of 'account' and state is forced to be included in the transactions outputs. Such operations still result quite unnatural and require a deep-level of abstraction together with serious complexities. Corda, Chain Core and Qtum [35; 63; 64] appropriately mix the Bitcoin and the Ethereum data-structures in order to have an UTXO-based model supporting complex contract operations; both systems adopt powerful virtual machines supporting operations written in Turing-complete code but differently to Ethereum the EVM are stateless.

2.2.1.4 Key-value model

An evolution of the previous data models consists in including in the state of a blockchain more variables, presenting them as tuples or tables. Such a general approach allows to adopt an UTXO-like or an account-like structure depending on the business constructed on top of the blockchain. For instance, Hyperledger Fabric offers the possibility to deploy Bitcoin-like currency systems (*Fabric-Coin* [65]), digital assets exchange (i.e., a contract, liabilities, properties) and tangible assets exchange (i.e., real estate and hardware). Fabric represents general assets as collections of key-value pairs (KVP) and it records state changes as transactions outcomes [36]. Kadena [66] adopts a table-based data model operating modification at a *per-row* level. That is, the blockchain registers a columnar history and transactions, both regular and smart contract ones, can update multiple column values at once thanks to a proper object syntax.

Model Comparison

Major differences between the four models are summarized in Table 2.1 (mentioned frameworks are then detailed in Section 3.4). Transacting using a UTXO model is conceptually equivalent to banknotes exchanging; the amount of paper bills (UTXOs) in the purse is the balance of our wallet and, whenever users spend money, they pay with a bill covering the cost (existing UTXOs) and they receive a change back consisting in other bills (new UTXOs). Thanks to the analogy, it is easy to note that this record-keeping model provides higher levels of scalability and anonymity; multiple UTXOs can be processed in parallel and whenever a new address is receiving new UTXOs the identity of the user owning the address is hidden. The account data model is constructed to record each account's balance so as to allow the issue of valid transactions. With accounts resembling traditional banks' debit cards, the blockchain structure results more intuitive and efficient. Adopting a stateful approach, the balance of each debit card is registered in the system and it is not included in the transactions data as for the Bitcoin stateless model.

Table 2.1: Blockchains data model comparison.

Data model	Benefits in	Drawbacks in	Frameworks
UTXO	scalability, security, anonymity.	applicability, efficiency, intuitiveness.	*Bitcoin*, *Litecoin* [67], *Dodgecoin* [68], *ZCash* [69], *MultiChain* [70].
Account	intuitiveness, applicability, efficiency.	security, anonymity.	*Ethereum*, *Tezos* [71], *IOTA* [25], *Ripple* [72], *Stellar* [73].
UTXO$^+$	scalability, efficiency, security, anonymity.	applicability, intuitiveness, model complexity.	*Corda* [35], *Chain Core* [63], *Qtum* [64].
Key-value	as UTXO and Account.	model complexity.	*Hyperledger Fabric* [36], *Kadena* [66], *Sawtooth Lake* [74].

2.2.2 Transaction Propagation

This step results crucial for the correct functioning of the consensus mechanism in the network. In order to establish which transactions are valid or not, all the validating peers must have complete knowledge of the information to be agreed upon. Therefore, transactions must be propagated to validators as fast as possible.

In order to optimize blockchain network performance and scalability, *flooding* or *gossip* protocols [75] are used for the propagation. Transaction propagation is carried out by means of a message exchange amongst peers. Blockchains clients connect only to a limited number of peers (neighbors); the message is first propagated to the connecting peers that then propagate it to their neighbors, and so on until it reaches all network nodes. Data present in the messages can be encrypted or not. Blockchain-based systems can require sending peers' authentication via exchange of a public key that can be included in the message or communicated out of band. Hence, receiving peers' can verify the data integrity.

From a networking performance perspective, it is important to establish to which of its neighbors peers have to relay a message. Flooding protocols include message transmission to all neighbors, while according to gossip protocols messages are relayed to a subset of randomly selected neighbor nodes. Both approaches assure a fast information dissemination but they differ in term of bandwidth and delay performance. The design of the transmitted message can impact the transmission delay. Delay-aware or bandwidth-aware neighbor selection can obviously lead to clear forwarding delay and bandwidth gains. A Bitcoin-like *announce-and-request* signaling, adding two more steps in peers communications (i.e., two more round-trip time, RTT, latencies), can consume less network bandwidth at the expense of delayed transmission. Such signaling can also imply a more complex data model: the protocol has to rule peers' request mechanism, peers' access to the data-ledger and peers' verification of the message originality (i.e., whether the information is new or not).

Apart from bandwidth and delay aspects, message propagation has to deal with network privacy and security aspects: multiple connections per node implies a large attack surface, while a limited number of communications facilitates interrupting and avoiding attacks (i.e., eclipse and DoS attacks [76; 77]). Regarding the identity-privacy aspects in permissionless blockchains, P2P protocols can reveal information on nodes identity. Deanonymization practices are related to the blockchain network topology built on top of the P2P overlay network, which can be generally disclosed if global-view P2P network traces are available or can be collected from different peers.

Bitcoin and the first generation of Altcoins work with flooding protocols using an *announce-and-request* signaling, where information is first announced to the neighbors to be sent afterwards, if not already possessed. Even if propagation costs with flooding do increase sub-linearly with the number of neighbors, the dissemination protocol is prone to deanonimization attempts [78] along with destabilizing communication strategies [79]; starting from withholding (*relay-delay* [80]), ending with net-split and gold-finger attacks [19]. Moreover, even if the announce-and-request signaling can be improved (e.g., compressing information by announcing headers only) or appropriately mixed with the classical push (e.g., Ethereum), the added latencies elapse can be more or less significant.

Permissioned blockchains are superior to permissionless ones also in the communication perfor-

mance. In permissioned environments where anonymity, message encryption, Sybil attacks do not represent a major issue, the communication security is concentrated on the faulty nodes management, to which gossip dissemination is more resistant with respect to flooding. The dissemination protocol does not require fixed connectivity to work since it operates with an *unsolicited push propagation* [81] mechanism, providing a consistent data synchronization tolerant to node crashes. Permissioned blockchains can count on a fast propagation with low latency (due to the direct *push*) and low bandwidth costs. In order to further speed up the propagation, the push mechanism can be improved reducing the size of the broadcasted messages by disseminating the transactions ID instead of the whole transactions.

Model Comparison

Table 2.2 summarizes the differences. First generation cryptocurrencies opt for flooding protocols using announce-and-request signaling, leading to higher bandwidth consumption and lower delay performance. Concerning security, the level of attack resistance depends on other factors (e.g., relay-delay). In this respect, Ethereum represents a transition from flooding to gossip adopting a "hybrid" design where some information is pushed and the rest is sent selectively. The gossip protocol promises good performances *pushing* messages; however, it results more sensible to net-split attacks due to the fewer connections involved in the propagation.

Table 2.2: Blockchains propagation mechanism comparison.

Communication protocol ⟶	**Flooding**	**Hybrid Flooding**	**Gossip**
bandwidth consumption	●●●	●●●	●○○
delay performance	●○○	●○○	●●●
net-split attack resistance	●●○	●●○	●○○
scalability	●●○	●●○	●●●
Basic protocol design ⟶	**Announce-Request**	**Hybrid**	**Unsolicited push**
RTTs	3	2-3	1
delay performance	●○○	●●○	●●●
examples	*Bitcoin*	*Ethereum*	*Hyperledger*

High: ●●●, Medium: ●●○, Low: ●○○.

2.2.3 Transaction (Block) Validation

Before being collected in blocks, transactions must pass the verification checks, i.e., they must have been created in accordance with the network rules. Once verified and inserted in the blocks, validators check whether the blocks meet all the protocol requirements necessary to assign the 'valid' entry and to proceed with the publication. These validation criteria must be deterministic and uniform across the network. While the transactions verification consists in a trivial cryptographic check, the block-validation phase is considered a key passage since it attributes to every blockchain-based system a distinctive character. After verifying that the block proposal has been correctly carried out, nodes

have to find an agreement on the validity of the block. More precisely, nodes in the network must agree on a unique record of transactions following a collaborative consensus protocol.

Transactions-ordering and consensus establishment can be considered as separated phases, or can be combined as in most of the existing consensus protocols. Bitcoin combined the two processes in the consensus procedure proposed in [2]. Validators in the Bitcoin network, known as *miners*, have to agree on both the order and the validity of the blocks. Some permissioned blockchains separate these steps (e.g., Hyperledger [36]): peers can agree on the ordering of the transactions that are validated in a second moment, right before their publication.

The agreement – on both publication and ordering of the transactions in the ledger – is reached through a distributed protocol executed by the nodes involved in the validation procedure. The consensus protocol must solve the Byzantine Generals (BG) problem [82], which consists in reaching consensus among trustless nodes (i.e., generals can be traitors). Since systems must accomplish this agreement state in a distributed manner, protocols should provide a *consistent* (or at least *eventually consistent*) view of the blockchain in the whole network. Thus, protocols adopt data *replication*, meaning that nodes hold replicas of the transaction ledger. Replicating data over nodes in the network makes blockchains resilient.

Building a proper consensus protocol is a challenge, as we develop in detail in Section 2.3. Since blockchain technology has many different use-cases, consensus protocols have been designed to meet specific system requirements. In permissionless blockchain applications, everyone is allowed to participate in the network, executing the consensus protocol and maintaining the shared ledger. The availability of these systems results in a substantial amount of computational power (hence energy) for maintaining a distributed ledger at a large scale (e.g., as in Bitcoin). Permissioned blockchains, with the presence of restrictions on who is allowed to participate in the network, adopt differently designed agreement procedures. More specifically, since the participants using blockchain are whitelisted, consensus protocols in permissioned blockchains guarantee higher performances.

2.2.4 Transactions (Block) Confirmation

Block confirmation coincides with its inclusion in the valid transactions history. Confirmation is the direct consequence of *consensus finality* (i.e., an agreed transactions never change or disappear) characterizing the so-called "consensus-based" blockchains. In this case, confirmation consists of a transaction predicate obtained when the majority of nodes get to decide to validate, and then publish the block containing the given transaction. However, in general, decentralized distributed ledgers may ensure a *probabilistic and economic* consensus finality – since they rely on eventually consistent consensus algorithms [83] – referring to cases in which the block-confirmation probability/cost (depending on the type of consensus) is increasing with the number of validated children blocks . In fact, despite the robustness of permissionless blockchains against double spending attempts (they need the involvement of the majority of the network to be successful), reversals are very common by means of forking attitudes that do not correspond necessarily to malicious intents. Confirmed blocks that cannot be discarded give way to the proposed exchange in the collected transactions. Therefore, in this case block confirmation is not a formal step explicitly notified to blockchain nodes, but it is implicitly inferred

by the actual presence of the validated block in the blockchain branch where the majority of nodes concentrate their efforts.

2.2.5 Blockchain actors and corresponding roles

We highlight in the following the key roles that blockchain nodes (with at least reading permissions) can assume.

2.2.5.1 Transacting parties

A blockchain transaction involves two different types of actors related to single or multiple blockchain users: the *data-sender* and the *data-receiver*. Interactions take place at address level: the *sending-address(es)* and the *receiving-address(es)* digitally track the data-flow (i.e., the transfer of digital assets) between the parties.

- *Data-Sender*: The data-sender is the node transferring data through an atomic operation (i.e, transaction) to a receiving node. The data-sender is not necessarily coinciding with (*i*) the transaction creator, (*ii*) the node with the right of initiating a data-transfer or, (*iii*) the data-holder [84]. Smart contracts involve the creation of a 'locked' transactions sequence that can be triggered by an authorized node (or even by a node outside the network) that may not be the owner of the transferred data. However, the data-sender is the one responsible for signing transactions (with its private key) in order to authenticate the origin of the object of the transfer (i.e., digital asset).
- *Data-Receiver*: Any user receiving a signed transaction that can: (*i*) recover the sender's public-key from the message and (*ii*) verify the transaction authenticity (i.e., transaction author and signature correspondence), is a data-receiver. Any blockchain node (user or contract account) can recover and verify the signature allowing tamper-proof transfers in the network.

2.2.5.2 Leading nodes

Consensus can be established by the election of a temporary *leader* node acting as a 'dictator'. The leader is responsible for both deciding which block to propose as a candidate to be included in the blockchain ledger and verifying the block proposal correctness. The leader goes out of power immediately after the validation of its block proposal. During its *round* (i.e., time interval where the leader has decisional power), the peer has no certainty that its block will be confirmed. Whenever a round expires, a new leader election starts.

The leaders election procedure is inherent in the consensus mechanism adopted by blockchain systems. Permissioned and permissionless blockchains adopt different methods to establish the peer in charge of proposing blocks to validators.

2.2.5.3 Validating nodes

As mentioned before, validating actors run the consensus algorithm and are responsible for establishing the agreement on the proposals made by the leading nodes. The validation of a block corresponds to the consensus among validating nodes on which block to publish and in which order.

Based on the journey of the transaction presented so far, it can be seen that it is nothing but the actors assuming the roles just described to characterize it.

At the first stage the transaction meets (*i*) the transacting parties, namely data-sender and data-receiver; the transaction is then transmitted to the (*ii*) leading peers responsible for verifying the correctness of the transactions, collecting them in blocks and proposing the block as a good candidate for the validation; at the final stage, (*iii*) validating peers proceed with the validity attribution.

In permissioned environments, each actor has a different role with no overlap in the procedures of block proposal and validation. This is due to the scalable voting-based consensus procedure adopted in permissioned blockchains (see Section 2.3). Instead, open-access blockchains foresee overlapping roles for the mining nodes. Indeed, mining can be interpreted as a simulation of the leader election in traditional consensus protocols. Table 2.3 shows the different actors of the most prominent blockchain platforms (reviewed in detail in Section 3.4) assuming the relevant roles previously presented.

A blockchain transaction is intended to meet these three main actors, but not only them. Some permissioned blockchains improve their scalability by designating to other peers different tasks such as execution-verification checks, leader election and ordering (e.g. Hyperledger *endorsers* and *ordering service nodes* [65]).

Table 2.3: Blockchain peers acting as 'transacting parties', 'leaders' and 'validators' in the different platforms.

Platform	Senders-Receivers	Leaders	Validators
Bitcoin [2]	Users/Clients	Miners	Miners
Ethereum [85]	Accounts	Miners	Miners
Hyperledger Fabric [86]	Clients	Ordering Services	Validating Peers
Corda [87]	Transacting parties	Transaction(s) issuer(s) only	
Tendermint [88]	Accounts	Virtual miners	Committee
Chain Core [63]	Users/Clients	Block Generators	Block Signers
Quorum [89]	Accounts	'Makers'	'Voters'

2.3 Consensus Mechanisms

The word "consensus" refers to a convergence to a common *interest*. Consensus is the task of getting multi-agent systems with interacting agents to achieve a common goal. Agents must reach an agreement regarding a certain interest (a value or an action, etc.) depending on their state. An example

of how consensus concerns systems that we use every day is *Wikipedia*; in such a case consensus is implicit, since every time an edit is submitted to the community before being published, it must be accepted by other editors; whenever an edit is revised by another editor and the revision is accepted, the system moves to a new consensus abandoning the previous one.

2.3.1 Consensus in distributed systems and blockchains

Agreement problems see abundant applications in complex systems dynamics [90] as well as in computer science and communications [91]. In such systems, consensus protocols must deal with dynamic agents that may fail during the agreement process.

The *two phase commit* (2PC) protocol [92], proposed in 1978, enables transaction processing in a distributed environment where nodes can atomically commit transactions trough *pre-commit* and *validation* phases. However, with 2PC any node failure compromises the consensus procedure. In this context, fault-tolerance (see Appendix) is defined as a property such that the system continues operating properly in the event of both process and communication failures caused by both honest nodes (i.e, crash failures) and nodes that act maliciously (i.e, Byzantine failures).

The *state machine replication* (SMR) technique [93] enables the construction of fault-resilient consensus protocols; robust against crash failures in trusted environments (e.g., Paxos and RAFT [94; 95]) and additionally capable of tolerating Byzantine failures in networks of untrusted parties (e.g., BFT). Any computation is considered as a state machine mutating its state through request receiving. In a distributed environment, state machines are replicated and executed across multiple nodes. Though they do not evolve simultaneously, they have to agree on a common sequence of requests (state transformations) they are going to accept in order to have consistent replicas. A popular class of state-machine replication protocol is the one of *Byzantine Fault Tolerant* (BFT) protocols [82; 96].

We develop in Appendix .2 desirable consensus protocol properties and behaviors with respect to asynchronous communications and data consistency guarantees, while recalling the strong relationship of these aspects with fault tolerance and the fact that in blockchain the consensus needed is about both on the elements of the ledger and their order.

The first approaches to consensus in distributed databases (2PC, atomic broadcast, SMR, BFT) can be considered as the predecessors of consensus solutions for DLTs. First generation blockchains (e.g., Bitcoin, Litecoin, Ethereum) establish consensus among millions of users in a probabilistic manner [42] thus, eventual consistency [97] took over from the initial need to maintain a coherent view of the system among participants. Failure-resilience characterize the systems as long as malicious nodes remain a minority in the P2P network (see Appendix .2.4). The idea is to introduce computational costs – to find a proof-of-work that validates a block of transactions – for charging peers who deviate from the default behavior (e.g., Bitcoin adopts previous approaches for fighting email spam [98] and preventing *Sybil attack* [99]). With the increase in popularity for cryptocurrencies, scalability and performance requirements changed significantly. Weaknesses of first generation blockchains led to a deeper analysis of the underlying technology through the lens of distributed computing. At a closer look PoW consensus procedure with its limited scalability and hight latency wastes too much computa-

tional resources. Appropriate amendments to the PoW procedure can guarantee challenging scalability levels without energy wastages.

2.3.2 Consensus Algorithms

Several alternatives to PoW were proposed in order to compensate for its complexity and scalability issues. The idea was to replace the wasteful computations characterizing the PoW consensus with alternative proofs of a performed effort in validating transactions. PoW consensus together with protocols characterized by an effort-based leader election form the class of *proof-of-X* (PoX) consensus algorithms as defined by *Tschorsch and Scheuermann* in [38].

Table 2.4: Summary about consensus mechanisms comparative analysis

Property	PoW	PoS	DPoS	PoET	PoI	PBFT-&-variants	Consortium BFT	Hybrid BFT-based
Node identity management	permission-less	both cases	both cases	both cases	both cases	permissioned	permissioned	both cases
Energy saving	no	partial	partial	partial	yes	yes	yes	yes
Tolerated power of the adversary	$< 25\%$ power	$< 51\%$ stake	$< 51\%$ peers	TEE	$< 50\%$ importance	$< 33.3\%$ replicas	variable (20%-33.3%)	$< 33.3\%$ replicas
Finality	✗	✗	✗	✗	✗	✓	✓	✓
Msg overhead	$O(1)$	$O(1)$	$O(1) - O(n)$	$O(1)$	$O(1)$	$O(n^2)$	$O(n^2)$	$O(n) - O(n^2)$
Nodes scalability	> 1000	> 1000	> 1000	> 1000	> 1000	< 100	$100 - 1000$	$100 - 1000$
Throughput (tps)	7-30	100-200	millions	1000	4000	up to 110k	up to 10k	1000
Latency (s)	up to 600	up to 600	unknown	unknown	unknown	less than 1	less than 1	up to 20

2.3.2.1 Proof-of-X Consensus

PoX protocols are designed for permissionless blockchains and relay on a probabilistic leader election process. In permissionless environments every node has the chance to become a leader simply proving that it made some "effort". The latter may have a computational, a monetary, or a storage nature or it may be an effort to assert itself on the blockchain network. The elected leader maintains his voting role till the new election's results are available. In the following we list and briefly introduce the most used PoX-based algorithms. A detailed analysis is provided in Appendix .2.5.

2.3.2.1.1 Proof-of-Work. The blockchain nodes aiming at validating a block of transaction (i.e., *miners*) should find an hash value of the block that meet a certain difficulty requirement. The winner of this competition can validate the created block of transactions. Hence, winning miners act as both leading and validating nodes. PoW does not guarantee consensus finality (see Appendix .2.3); transactions can be considered as confirmed only when included in the *longest chain* (See Appendix .2.5.1).

2.3.2.1.2 Proof-of-Stake and Virtual Mining Alternatives. The PoS mechanism resumes the PoW one while passing from a real mining to a *virtual mining* (i.e., consumption-free mining). The leader election in these mechanisms is based on the *stakes* owned by the network users determining the voting power in the consensus. The idea beyond the mechanism is that users with more commitments would not be likely to attack the blockchain. Several variations of the PoS consensus exist in order to (*i*) avoid the centralization of voting power in "rich" committees and, to (*ii*) overcome an attack known

as *nothing-at-stake* [100]; it consists in validating as many blocks as possible resulting convenient for the low computational cost to validate blocks (i.e., the same cost to cryptographically sign a block). These variations generally require validators to (*i*) weight their coin/stake or to (*ii*) allocate some resources during the validation phase (see Appendix .2.5.2). Alternative performing implementations such as PoET [74] and PoI [101] fight against centralization trends (i.e.,coin/resources accumulation) by respectively (*i*) relying on a random timer to chose the leader of the round and, (*ii*) incentivizing the eligible leaders to increase their transaction flow and volume in the network. Moreover, in order to be more efficient the mechanism can work with restricted elections i.e, *delegated proof-of-stake* (DPoS [102]).

2.3.2.2 BFT and Hybrid BFT-based Algorithms

BFT protocols work well in blockchains with a limited number of participants, therefore they do not fit for public systems but for closed ones. BFT algorithms guarantee both liveness and safety (see Appendix .2.1) of a network given that at least 2/3 of the participant are honest (i.e., PBFT protocol [103]). The different BFT-based variations (see Appendix .2.6) work with additional permissions on the validating nodes. In order to scale up the protocol, hybrid algorithms have been created. It is possible to combine PoX and BFT by using the former to create committees (i.e., communities of nodes) and the latter to come to an agreement (see Appendix .2.7). This class of algorithms decouples the *block generation* phase from the *block validation* phase; the two process are independent and managed by different actors (that can be the same nodes but with different roles).

2.3.3 Comparison between blockchain consensus protocols

Previous sections presented the problem of reaching consensus in a distributed system. Traditional consensus protocols have opened the way to PoX-type mechanisms and then reconsidered in permissioned blockchains fro their performances. *Vukolic* [104] work is one of the first at addressing a comparative analysis on the different consensus procedures however, it focuses only on the PoW-based algorithm and traditional BFT scheme. Recent works [3; 44; 45; 46; 105] compare different agreement protocols in terms of (*i*) *node identity management*, (*ii*) *energy saving*, (*iii*) *tolerated power of adversary*, (*iv*) *transaction finality*, (*v*) *communication complexity*, (*vi*) *nodes scalability*, (*vii*) *throughput* and, (*viii*) *latency level*.

Table 2.4 summarizes these comparative studies. The data shows the tendency to implement safer and high-performance (1000 tps) blockchain-based systems with low energy impact and low latency, that reach a final agreement with the guarantee that the validated blocks will not be discarded. It can be deduced that further work needs to be done regarding the message overhead between the consensus participants (n in Table 2.4).

In this chapter we provide an overview of blockchain technology, exploring all the layers characterizing the blockchain architecture (i.e., network layer, data model layer, execution layer, consensus layer and, application layer), particularly focusing on those that are crucial for deciding whether to adopt the technology or not and.

Chapter 3

Blockchain Vadamecum

Summary

3.1 Introduction

Leveraging on the important background presented in the previous section, this section is meant to start our blockchain vademecum [1], to give to the reader a comprehensive tutorial about when to use blockchain, which solution to use, and how to use it, based on use-case requirements.

[1] 'Vademecum' is a term that may not be well-known by the reader. It derives from the latin expression 'Vade Mecum', literally meaning 'go with me'. It refers to a synthetic collection of information concerning a specific field or technique (blockchain in our case), having the goal to provide the reader with quick and concise responses on the different details of the specific field or technique.

Table 3.1: Reading list on blockchain application domains

(*i*)	clearing, collateralization, real estate [4; 106; 107; 108; 109].
(*ii*)	personal data-management [84; 110].
(*iii*)	energy [111; 112; 113], health-care [114; 115; 116; 117; 118; 119; 120; 121; 122].
(*iv*)	storage, authentication, e-commerce [123; 124; 125; 126] communications & networking [127; 128; 129].
(*v*)	supply chain [130; 131; 132], transportation [133; 134].

During the past few years, research societies along with industrial and governmental institutions intensively worked on DLT and blockchain, trying to understand better this paradigm and its place in today's market. This resulted in many publications and standardization activities as well. In the following, we provide the reader with a decision model to understand *When* to use the blockchain technology (Section 3.2) and *Which* type of blockchain suits a certain use case best (Section 3.3). The decision model is characterized by two decision paths (When and Which paths) that can be traversed either consecutively or independently; the decision points can be both direct questions or trade-off points [2]. Once decided the type of blockchain, we provide the reader with a complete list of the most popular blockchain frameworks in the market accompanied by details on their structure, operation and implementation (see Sections 3.4.2 and 3.4.6) allowing the reader to find the one that comes closest to her business case. As of our knowledge, our effort is unique in the purpose and the style, tackling these important questions in a more direct, expressive and thorough way than in existing works reviewed in Section 2.1.4 focusing on specific usages, modes, or blockchain use-cases.

The design of the current blockchain-based systems comes as a response to market needs (i.e., good level of performance and scalability). Closed blockchains, where the decentralization target is not met, may one wander whether the new technology can bring benefits with respect to traditional solutions.

In the following, we use Fig. 3.1 as a support for the When and the Which questions in Sections 3.2 and 3.3. Then, we address the How question in Section 3.4. We start by analyzing the major available blockchain platforms and their characterizing elements, following the information provided in Sections 2.2 and 2.3. Afterwards, we differentiate platforms according to the representative features of the different blockchain layers. In addition, we present relevant blockchain use-cases, strongly advertised and tested at industrial level, applying to them the proposed vademecum logic.

General purpose reading list: In developing this tutorial we made use of a broad spectrum of documents going beyond academic literature, and including books, white-papers, technical reports, blockchain forums, discussion papers, and online encyclopaedias. We concentrated on works showing real applications of blockchain in the industry going beyond the well-known digital payment systems proposed by cryptocurrencies. The main investigated areas were: (*i*) finance, (*ii*) security-and-privacy, (*iii*) public, (*iv*) Internet-of-Things (IoT), (*v*) smart business. We report such reference works in Table 3.1. Some of these blockchain applications are presented in Section 3.4.6 for the ways in which

[2]Trade-off points represent situations that involve a choice between two or more aspects, where the loss of value for one aspect constitutes an increase in value for the other one(s). In the proposed decision tree alternatives are (*i*) blockchain or traditional database features for Section 3.2 and, (*ii*) permissionless or permissioned blockchain features for Section 3.3.

the blockchain technology has been chosen. In addition, our reading list includes works investigating when a blockchain can revolutionize a business [1; 54; 135], benefits and drawbacks of both permissioned and permissionless blockchains [52; 70; 104; 136; 137], and links with traditional solutions [9; 16; 138; 139; 140].

3.2 When to use blockchain?

This section focuses on the first general question of the vademecum: when to use blockchain as a technology? Our complete answer to the When question is given passing through the following direct questions and trade-off points (see Fig. 3.1).

1. *Do you need to store and share a ledger state?* We start from a situation where a ledger database is required i.e., data in transaction form needs to be stored and shared. Data constitute the ledger state, which is subject to updates that must be shared over the network. Whenever it is not needed to share a stored state, complex cryptographically-based architectures result unnecessary for simply letting stored data to be accessible. Therefore, in the presence of a negative answer blockchain is certainly not needed and traditional solutions are preferable.

2. *Are there multiple potential writers?* The adoption of blockchains makes sense only when data need to be stored by multiple users and shared among them. Indeed, in a blockchain multiple users (not necessarily all network users) are supposed to have writing access and permission to participate in the procedure to establish consensus among parties. Blockchain lets business move from hierarchical client-server systems with locked writing rights to decentralized P2P interactions with multiple (if not all) nodes able to write to the distributed ledger.

3. *Who do you entrust with the ledger maintenance?* Blockchain enables interactions among trustless actors circumventing any intervention by a central authority. The need for decentralized systems arises whenever network participants lose their trust on a (alternative or pre-existing) centralized system. However, the transition from a centralized to a decentralized system is not necessarily radical; blockchains can decentralize some functions while keeping others centralized. Blockchain has revolutionized the concept of 'trust', which is no more related to the identity of the actors in charge of the validation procedure, but it is related to the protocol architecture. Clients trust the technology that is forcing validators to follow the protocol punishing or making unfeasible any possible deviation. For such a key strategic question on the trust, we can spot three possible types of answers:

a) An external third party: the system maintenance is entrusted to an external entity which in case of failure could be switched. In such a case, designers should opt for a centralized architecture that is easy to deploy and maintain by the trusted third party.

b) A group of selected actors: nodes in charge of updating the ledger participate to the system. Their identity can be known or unknown, however, the methods for selecting these nodes and the targeted activities are important aspects. Indeed, the class of partially-centralized systems in-

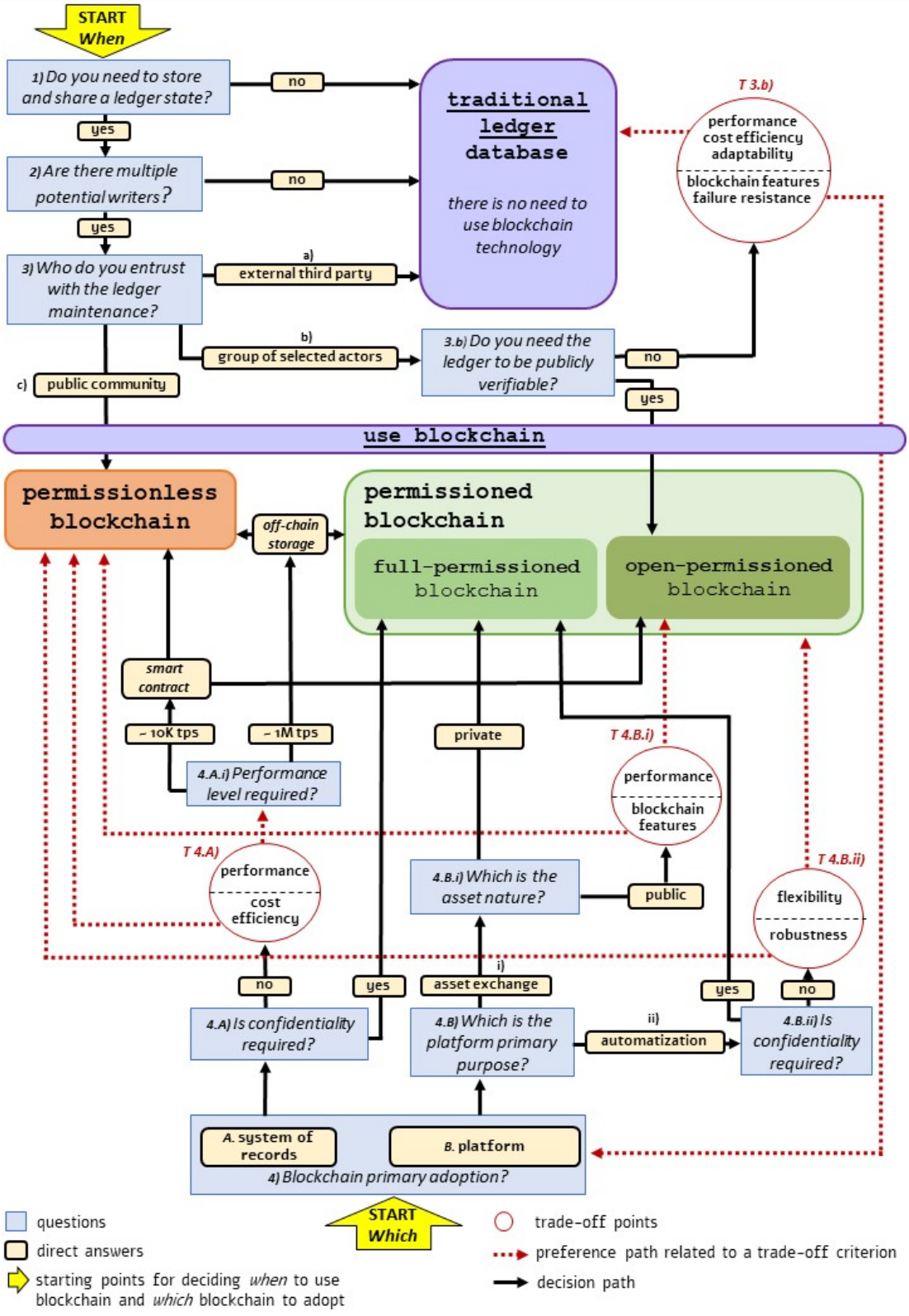

Figure 3.1: When to use blockchain, and which type, instead of adopting a traditional database system. Red circles represents trade-off points between crucial aspects for the different blockchain use-case. The red arrows indicate the consequence of giving priority to one aspect rather than the other, while black arrows report answers to all the questions – coming with an order – of anyone interested in the blockchain technology. 'tps': transactions per second.

cludes a spectrum of possibilities such as adopting private distributed ledger, creating *consensus committee* [46], and structuring the communication with external trusted systems [141]. Instead of providing open-access to anyone, blockchains can bind certain of their functionalities (read and write) arranging *permissions*. We may therefore have an escalation of permissions, from the single permission to read the transaction log to the ability of validating transactions. At first, permissioned blockchains select participants with network access controls; their identity must be known. Then, permissions are given to implement any type of change to the data registry; different trust levels can be associated with different nodes' roles (see Section 2.3). Moreover, whenever the validation of a transaction is linked to an external variable realization, one may choose whether to trust or not the actor designated to communicate with the outside.

Regardless of any restrictions on the node roles, once decided to trust a restricted entourage for the validation process, one may wander which actor to entrust the verification of the operation correctness. Let us recall that block verification consists in a repeated check of both the chained blocks integrity and authenticity – carried out in most cases by the validators themselves – and the chained blocks validity. Blockchain transparency allows any network participant to verify whether a published block was validated according to the protocol since all network nodes have the same view on the log. On the other hand, verification checks are entrusted to a central authority whenever participants differ in the view they have of the ledger. Thus, the next question at this point is:

3.b) Do you need the ledger to be publicly verifiable?
Whenever a system requires public verifiability, one may keep restrictions on writing rights but at the same time leave the freedom to everyone to observe the system state – as for open-permissioned blockchains. For those cases in which verification checks may not be in the public domain, the choice between a private blockchain (full-permissioned) and a traditional solution is clearly linked to the nature of the verifier(s). Verifying peers coincide with the so-called validating peers in a private environment where transactions validation is performed by trusted parties. The choice now is between a *centralized verifier* – leading to the adoption of a traditional central database where the group of trusted nodes organize themselves in a central authority (with both reading a writing rights) representing however, a potential single point of failure – and a *distributed verifier* – consisting in several trusted validators known to the network operating in a P2P framework where all the participants in the system may connect to each other. The adoption of a blockchain (permissioned in this case) rather than a traditional solution is dominated by trade-offs regarding mainly the impact on the throughput, the costs, the presence of the basic blockchain features, the failure resistance level and the adaptability to different business cases.

Trade-off 3.b) performance, cost efficiency and adaptability VS blockchain features and failure resistance
Traditional centralized databases are widely used both for their simple architecture – easy to adapt to each use case and often affordable as the data is stored and maintained from a single central computing node – and, for the speed and ease in updating the data they manage – every change is managed by the central authority and immediately communicated to users [15]. In fact, the central authority can easily modify data with CRUD (Create, Read, Update, and Delete)

commands. Thus, the technology strengths consist in high levels of performance (in terms of transaction processing rate), low costs in adopting the technology (in terms of design and management cost, as conventional softwares are cheaper than blockchain solutions) and high degree of adaptability in managing any type of data and its use.

Despite the countless advances made by blockchain technologies to reach higher levels of scalability, throughput and latency, blockchain will likely always be less performing than a centralized database. This is because processing any change in a distributed system – through transactions – requires additional efforts consisting in: (*i*) applying and verifying the digital signature, (*ii*) agreeing on a unique vision of the data ledger, (*iii*) replicating data across the network and, (*iv*) updating the ledger only with *write*-operations. In blockchain the idea is that the validating nodes independently process transactions and then at a second stage compare the obtained results with the rest of the network until they come to an agreement. However, blockchain offers, at the same time, the six important features presented in Section 2.1.2.2 (decentralization, immutability, confidentiality, integrity, authenticity and transparency), that are absent (in their entirety) in traditional databases. In addition, since blockchain is first and foremost a distributed ledger, it is robust against node failures [3]. Adopting or not blockchain is therefore a matter of which set of quality properties to privilege between (*i*) performance, cost efficiency and adaptability and, (*ii*) blockchain fundamental features and failure resistance.

c) The public community: Whenever trust cannot be laid on a set of network nodes, it is better to have confidence in a protocol (i.e., a set of rules) that guarantees the correct functioning of a system maintained by the public community. Permissionless blockchains enable untrusted parties to interact without relying on any *man-in-the-middle* (i.e., disintermediation). Transaction history is fully transparent to everyone. Validation and verification are carried out in a fully open and distributed fashion; any network node can participate in the process possibly remaining pseudonymous.

3.3 Which blockchain to use?

Thanks to the attractive blockchain properties (Section 2.1.2.2), the development community has worked hard to broaden its range of applicability. At this point, the vademecum suggests to apply blockchain also to multi-access shared ledger situation such that there is a circle of trust, and concessions in terms of performance, cost efficiency and adaptability can be acceptable.

Permissionless blockchains require users to direct their trust towards cryptography and related mathematics, while permissioned ones ask for confidence in few (or all) nodes of the network. Therefore, given that blockchain is the right technology after the When questioning above described, at this stage the first question the designer may wander is in which of the two categories falls its use-case. In addition, if directed to a permissioned blockchain one may choose whether or not to put restrictions on

[3] However, it should be noted that for permissioned blockchains any centralized procedure (such as validation, verification or external communication) can be considered as a single point of failure.

data ledger access. The vademecum chart in Fig. 3.1 can now be read from the bottom to the top.

4) Which is the blockchain primary adoption?: Blockchain can be primarily adopted as (*i*) a *system of records* (SOR) and as a (*ii*) *platform*. Polarization toward the former or the latter application class is important to characterize the blockchain nature.

A. *Blockchain as a system of records* SOR's principal goal is storing data and wisely processing it in order to re-present to users the history of data. Blockchain constitutes an innovative solution to track the history of information modifications that is characterized by interesting features, including its transparency. The question now is which blockchain solution between a permissionless and a permissioned one is best for a SOR. Firstly, one should realize if there are disclosure issues. Once understood the desired privacy level (between anonymity and confidentiality), the choice is a matter of trade-offs; high performance comes at a cost.

4.A) Is confidentiality[4] *required?* Privacy and confidentiality within blockchains are controversial; what permissionless blockchains can hide to the network is the users' identity only, conversely, every operation performed in the network is in the public domain. Hence, permissionless blockchains guarantee users some degree of anonymity (pseudonymity) without offering any confidentiality in transacting on the blockchain. On the other hand, private blockchains (with restrictions on both writing and reading operations - and where participants are known in the network) can ensure that 'what happens in the network remains in the network'. Therefore, if operations are not to be disclosed to the public, the most appropriate solution is a blockchain that is not accessible to everyone, i.e., a full-permissioned blockchain; otherwise, the following trade-off allows discriminating among a permissionless blockchain and a permissioned one.

Trade-off 4.A) performance VS cost efficiency: In the absence of confidentiality constraints, one should concern about the importance of performance over cost efficiency. In order to achieve a processing rate of the order of thousands tps, the classical permissionless blockchain structure must be abandoned. Blocks of transactions should no longer be processed one at a time; blockchain needs to adopt an architecture favoring the processing of multiple blocks in parallel. These result in a more complex technology structure with high design costs. Permissioned blockchains (both open-permissioned and full-permissioned) offer good performance due to their restricted nature where data validation, verification, replication and modification are faster with respect to a public environment. Thus, whenever priority is given to the throughput, the best choice is in favor of permissioned solutions (both full-permissioned and open-permissioned).

4.A.i) Which is the performance level required? If it is required to have performance comparable to that of a centralized system, a possible solution is to store data (*i*) *off-chain* or (*ii*) on-chain via smart contracts. Blockchain initial aim was to enable data-storing on-chain; however the kind of data stored was the transaction history. In the Bitcoin blockchain external data was initially stored on the ledger through unofficial transaction manipulation (e.g., writing in a coinbase trans-

[4]We mean by the term 'confidentiality' the non-disclosure to the public of the operations performed by blockchain users.

action or using a fake account address) discovered and disseminated by avid network users [142]. Due to the limited space provided by the OP-RETURN, second-generation blockchains proposed alternative solutions based on smart contracts and off-chain solutions. Data can be included in a smart contract at variable or event level directly on-chain (on a blockchain – no matter the nature – supporting smart contracts), however performance (up to thousands tps) is not still comparable with the one offered by traditional databases (e.g. Multichain early versions [143]). Off-chains solutions are the best in terms of performance; raw-data are stored off-chain, while it is possible to handle meta-data or hashed-data on-chain as a complementary storage (e.g., Swarm [144] and Filecoin [145]). However, the ease of communication between the two technologies heavily depends on the type of blockchain and the corresponding off-chain solution chosen. The ideal off-chain storage is a private cloud attached to the corresponding blockchain, thus a full-permissioned structure (e.g., Microsoft Cryptlet Fabric [146]).

B. Blockchain as a platform
In general a blockchain-based system enables digital data-sharing, digital data-storing and virtual interactions among peers. The principal goal of a blockchain platform is to form P2P digital relationships favoring digital exchanges and business automatization.

4.B) Which is the platform primary purpose? The central question relies on the platform primary purpose between the following fundamental categories:

i) Asset digital exchange: Blockchain enables the sharing of any valuable data (i.e., asset) among parties without any geographical and timing constraint. Both the asset nature and the size of the data-flow impact on the choice of the blockchain nature and its architectural design.

4.B.i) Which is the asset nature? Assets could be *sensible data* that have to be managed restricting access to the record – full-permissioned blockchains. If no disclosure issue occurs, the quest of adopting or not permissions in writing rights merely depends on trade-offs: for better performance than that offered by Bitcoin-like blockchains one should pay the price of not guaranteeing full transparency (auditability) and equal rights of participation.

Trade-off 4.B.i) performance VS blockchain features: The choice whether to give priority to the basic blockchain features rather than to the performance is strictly linked to the nature of the exchanged assets in the network. To give the reader an idea, let us take the case of *tokens*. Blockchain became popular thanks to assets *tokenization*; the aim is to create a trading system of items that cannot be duplicated. Cryptocurrencies propose alternative payment methods through their tokens that represent a currency, i.e., a generic payment instrument. Other types of tokens such as *security tokens* – representing a participation, in terms of dividends, voting rights, interest rates and/or percentage of the issuing entity's profits – and *utility tokens* – representing only the right to purchase goods and services of the issuing entity – were created on blockchain in order to digitally participate in a business having easy access to digital services-goods [147]. In the case of currencies,

all blockchain properties (auditability in particular) are fundamental in the system, thus blockchain designers are forced to loose something in terms of performance since usually currencies are intended for the widest possible public. On the other hand, security and utility tokens are considered as an alternative investment method, therefore transparency is not essential in this case and one may adopt permissioned blockchains profiting from higher processing rate with respect to permissionless solutions.

ii) Business automatization: Blockchain platforms allow smart contract deployment and execution with the aim of letting any business to automate its functionalities. After questioning the sensitivity of the automatically managed data (as in question A.1), it is important to consider the ability to support world changing applications. There is no perfect blockchain for every use-case. However, what a selection of participants is affecting the most are: (*i*) the non functional properties of *security* and *robustness* in terms of failures resistance and, (*ii*) all the features related to blockchain applicability – that is, the *flexibility* to adapt the designed blockchain protocol to different business cases. Therefore, the choice is a matter of trade-off; more flexible architectures are usually less robust.

Trade-off 4.B.ii) flexibility VS robustness: Permissionless blockchains suffer from limitations in data-storing, computations, scalability and performance which does not make it applicable to many business situations. On the other hand, permissioned blockchains result more flexible for configuration since governed and hosted by a single central committee of trusted nodes; therefore, any type of change is made faster than in a fully open and trustless environment. A classic example is off-chain storage that results more intuitive in private networks that ease communications between the off-chain storage system and the blockchain [148]. Concerning security and robustness: is it better to adopt a permissionless blockchain architecture or to build a new structure on top of it. In fact, fully open-access distributed ledgers are quite robust against any type of failure as long as 50% of the system nodes are honest (see Appendix .2.4). In order to have robust but performing public blockchains, a possible solution is to use *side-chains* [149]. With side-chains one may move assets and functions from the principal blockchain (*main-chain*) to a second one. Thus, it is possible to have a private blockchain linked to a permissionless one. [142] gives a detailed report on the levels of performance and flexibility in permissioned, permissionless and open-permissioned blockchains. With regard to security and robustness in DLT we refer to the works of *Lin et al.* [49] and *Li et al.* [48].

3.4 How to use blockchain?

It will be important to assess *HOW* one can use blockchain, thus this section is meant to give suggestions for accelerating the implementation of the blockchain. Assuming that the reader has already decided to use blockchain and the type of blockchain to adopt, the next step would be the choice between developing her own solution, or using one of the existing platforms (Fig. 3.3).

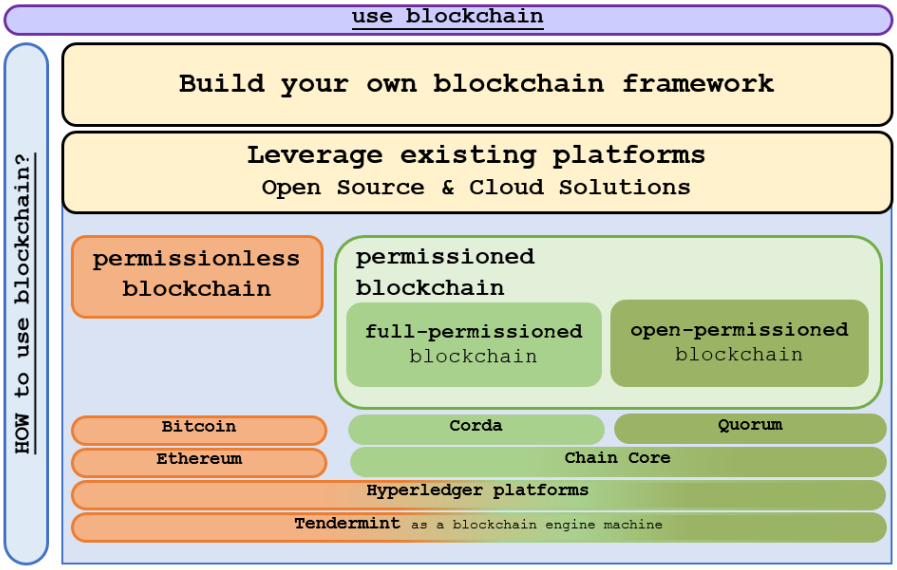

Figure 3.3: Blockchain adoption is possible by both (*i*) building an own framework or, (*ii*) leveraging existing platforms that can be open source and/or provided by cloud services. Here, the major blockchain platforms related to the three blockchain participation modes, are listed.

- Open-source platforms: Different blockchain frameworks can follow different visions in terms of application fields [150]. While ones have a versatile architecture that can be employed in various industries, from banking to supply chains, others are driven by very specific use-cases. Nevertheless, available major blockchain platforms can be easily classified into four groups as illustrated in Table 3.2 [151]. After having understood which of the *permissionless*, *open-permissioned* and *full-permissioned* blockchain implementations is the most suitable, one can exclude some solutions of those just presented with respect to their nature (Fig. 3.3). Considering that new blockchain frameworks regularly appear on a weekly basis, we survey in the following only the mostly used ones for proofs of concepts and prototypes. The reviewed blockchain frameworks are open source (Section 3.4.2).

Table 3.2: Classification of frameworks

(Group I)	Group (II)
Permissionless Transactions only (Bitcoin)	Permissionless With Smart Contracts (Ethereum)
Group (III)	Group (IV)
Permissioned Transactions only (Chain Core)	Permissioned With Smart Contracts (Hyperledger Fabric)

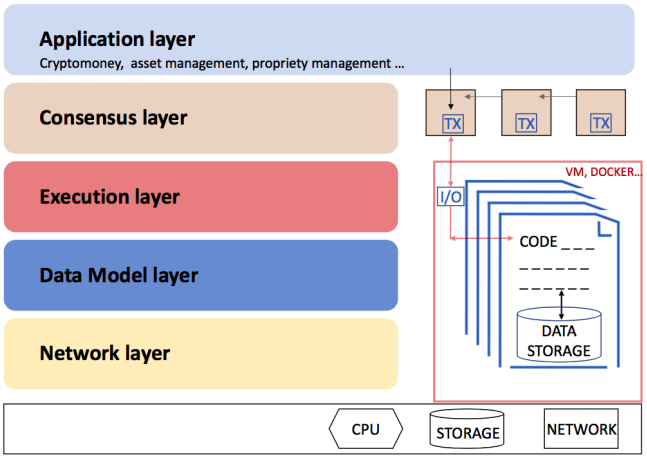

Figure 3.4: Abstraction of a blockchain framework as a multi-layer system.

- Blockchain on Cloud: Blockchain as a Service (BaaS) is an offering that allows customers to leverage cloud-based solutions to build and host their own blockchains: applications, smart contracts and different blockchain functions. It is indeed similar to the concept of Software As A Service (SaaS) model . External providers menage all tasks to keep the infrastructure operational (Section 3.4.5).

We compare the most prominent blockchain frameworks differentiating their layers, highlighting their architectural limitations and functional properties hence, providing all the information necessary for considering a platform solution. This tutorial part, where the characterizing aspects are listed and compared, may enable the reader to use one of the existing frameworks as a possible solution or as a guideline for developing their own framework. We discuss in the following pages along with architectural limitations, performance evaluation, and a review on Blockchain as a Service (BaaS) offer. Finally, we highlight the underlying vademecum logic of representative industrial blockchain applications.

3.4.1 *Multi-layer abstraction of a blockchain framework*

We depict in Fig. 3.4 the general abstraction of blockchain frameworks with a multi-layer view, marginally revised with respect to the layer division proposed by Croman et al. [13] and Dinh et al. [152], based on the recent advances in blockchain frameworks described hereafter. At the application level we find blockchain applications, such as a crypto-money wallets, in charge of communication within the blockchain network via transactions; it encompasses all APIs and application level communication protocols. At the consensus level we have the consensus algorithms in charge of ensuring a single valid chain of blocks in the system; it can be a static or a dynamic plug-and-play consensus system, and it directly determines a system adversary model and different nodes roles. At the execution level we have the smart contracts environments such as compilers, VMs, containers; it determines the transactions execution mode (CVM, EVM, TxVM) and the languages (Turing-complete or not) for smart contract development. Indeed, all blockchains have built-in smart contracts that implement their transaction logics. Bitcoin for instance first verifies transaction inputs by checking their signatures,

then it verifies that the balance of the output addresses matches that of the input ones. As the built-in part of Bitcoin protocol, these types of 'smart contract' are part of the framework code base. When we speak about smart contract languages, we only refer to smart contracts that can be defined by users. At the data model or storage level we have the data structure, contents and possible operations on data storage as well as ledger maintenance; it defines all the parts colored in blue in Fig. 3.4. At the network level we find the transaction forwarding and dissemination strategies as implemented by transport-layer and network-layer protocols. We present in the following diverse protocols and technologies from all levels adopted by different blockchain frameworks.

3.4.2 Major blockchain platforms available

Different blockchain frameworks can follow different visions in terms of fields of application [150]. While ones have an expendable architecture that can be employed in various industries, from banking to supply chains, others are driven by very specific use-cases. Nevertheless, available major blockchain platforms can be easily classified into four groups [151] as illustrated in Table 3.2. Considering that new blockchain frameworks regularly appear on a weekly basis, we survey in the following only those used the most for proofs of concepts and prototypes. Whenever appropriate, we recall aspects described in the previous sections (e.g., on consensus, journey of a transaction, block structure, etc). Note that, while many different cryptocurrencies exist today [38], cryptocurrencies frameworks comparison is out of the scope of this article, although it is an interesting subject. Moreover, minor or very young Blockchain platforms that we omit mostly follow the pattern of one of the major platforms described in the following. Fig. 3.3 positions the presented platforms with respect to the three blockchain natures (i.e., participation modes).

3.4.2.1 Bitcoin blockchain

Bitcoin is a public, permissioneless PoW-based blockchain network, giving an open access to its transaction logs. Besides its already well described primary lifecycle, the Bitcoin protocol actually does facilitate a weak version of smart contracts as well, using the UTXO model 2.2.1: UTXO in Bitcoin can be owned not just by a public key, but also by a script expressed in a simple stack-based programming language, requesting within a transaction attending to spend that UTXO, the data that satisfies the script. However, the scripting language as implemented is not Turing-complete.

As the first blockchain network publicly used, Bitcoin can be seen as a rigid predecessor of todays more enhanced frameworks that have overcame some of its limitations. With in mind possible re-use of the Bitcoin framework for other goals than the cryptocurrency one, it is important to notice that Bitcoin network was meant to serve as a public payment system without centralized determination and was designed accordingly, making it unsuitable for permissioned private systems. Participant nodes in a Bitcoin-like network can choose to be clients or miners. Clients (users) are capable of receiving and sending transactions while miners are in charge of mining toward PoW. In practice, four distinguish processes keep the network running: (*i*) Network Discovery process, (*ii*) Transaction Creation process, (*iii*) Block validation process and (*iv*) Mining process (software details can be found in [151]).

The P2P protocol is such that, in order to initiate a transaction, a sending peer transmits a signed transaction to its neighboring peers. Neighbors forward it in the overlay network only if they verify its validity; if a transaction is invalid, the propagation stops. Miners, as well as all nodes in the network, receive those new transactions through the P2P network. They verify and store them in an unverified transaction pool. In case the miner discover from the network that a given block was mined, it stops mining, it updating its pool of unverified transactions, and starts all over again. Once mined, a new block is transmitted over the P2P network. Every full node (those with a ledger) checks now the block validity before adds it to the ledger (block header, a hash, nonce, and all included transactions). This 'order-execute' architecture [65] requires all full nodes to execute sequentially every transaction, which cause low throughput performance. Two basic P2P network operations are used: an attachment strategy, which defines how clients establish connections to other peers, and a communication strategy, which defines how nodes communicate with their neighbor. Peer discovery in Bitcoin is performed by querying a hard-coded list of DNS seeds for bootstrapping. In case of previous connections, node can discover other peers also by requesting the IP list from neighbors; moreover, information based on own observations maintain a blacklist of misbehaving IP addresses. In addition, Bitcoin limits the number of connections per IP address range; this way nodes do not establish too many connections, enhancing their DoS resistance. The default number of connections is 8 (nevertheless, it was proposed to increase this number [77]).

The Bitcoin code is released under a MIT license [153].

3.4.2.2 Ethereum blockchain

Ethereum [8; 154] is an open platform designed to build and use decentralized applications that run *smart contracts*, i.e., a blockchain network of distributed applications that mechanically execute tasks when certain conditions are met. This can be done at a larger degree than what possible with the UTXO model in Bitcoin.

A smart contract is intended to enable a blockchain with a built-in fully-fledged Turing-complete programming language (named Solidity) to create contracts, allowing users to design own applications by writing up the logic in a few lines of code. This was an innovation when firstly proposed, but today, others platforms do also support smart contracts. As Bitcoin, Ethereum is also cryptocurrency-based, i.e., miners work to earn the crypto token called Ether, which is also used to pay transaction fees and services in the Ethereum network. To execute a smart contract, an Ethereum virtual machine (EVM) [154] must be hosted at every network node. Ethereum uses a PoW-based consensus algorithm, called Ethash, specifically created for Ethereum, despite there are recently efforts to switch to alternative PoS-based implementations. On an average, a block mining with PoW takes $15s$. The way Ethash provides a PoW is by emphasizing memory hardness, i.e., the fact that memory access can also be a bottleneck, besides the computing power. Ethash is designed to consume nearly the entire available memory access bandwidth; the PoW function is made to be sequential memory-hard, i.e., the nonce search requires a lot of memory and memory access bandwidth, so that the memory cannot be used in parallel to discover multiple nonces simultaneously [154]. Here smart contracts are visible to all users of the blockchain, hence also making security holes and bugs visible to everyone.

In terms of framework customizability, Ethereum cannot be seen as a modular framework: embedded functionalities cannot be changed on demand, even though there is no 'one fits all' solution. Moreover, Ethereum uses state-machine replication by implementing active-replication [155], where transactions are ordered at first and then broadcasted and executed sequentially on all nodes. The 'Order-execute' architecture explained in [65] requires all transactions to be deterministic: this type of architecture is largely adopted by blockchain frameworks, but it comes with overloads discussed in the Section 3.4.4. Used 'account based' data model enables actors in an Ethereum network to create transactions, create contracts, send messages and mine Ether. Network maintenance is done thorough four processes [151]:

i. Network discovery, enabling new nodes to join.

ii. Transaction creation, which allows users to create transaction or contracts and allows contracts to create transactions and messages.

iii. Block validation, done by every full node in the network before they add the new block to their blockchain.

iv. Mining, in charge of Ether mining and broadcasting a new block to the network.

As already anticipated, Ethereum supports three type of accounts: (*i*) Contract Account (CA) that can set up a transaction with address internally stored within a contract, or establish a transaction with another CA; (*ii*) EOA (Externally Owned Accounts) that initiate transaction to transfer ether to another EOA, or create a new contract, or call the function of an existing CA; (*iii*) Miners that can collect new unverified transactions and compute a valid state of a ledger, validate transactions, verify signatures and transaction fees, execute codes and checking they do not *run out of gas* (i.e., fail since the transaction fee paid out is not adequate for the transaction processing complexity). P2P communications in Ethereum rely on the Virtual P2P (Vp2p) wire protocol: nodes communicate using a cryptographic transport protocol coined RLPx [156]. RLPx uses a node discovery process with a 512-bit public key as node identifier, an encrypted handshake to establish connections; a node can connect to a known peer (a previously connected peer from which a corresponding session token is available for authenticating the requested connection), or to a new peer. Nodes find peers through the RLPx discovery protocol's distributed hash table (DHT). Peer's connections can also be initiated through a client-specific RPC API. A new node aiming to connect to the Ethereum network has to download the source code which comes with the IP address of a bootstrap node assumed always to be online and connected to other correct nodes. Following connections are established directly with other nodes via the DHT.

While the main Ethereum platform is a public blockchain network, the software is open-source and allows one to download and configure the network to be a local private network (participants are those that are granted permission only) using the proof-of-authority (PoA) consensus engine. We refer to Ethereum as the network in a public setup, used to transfer Ether between participants. Hence, Ethereum network achieves roughly 15-40 transactions per second (tps) with an estimated latency around $15s$ per block. In private setup Ethereum can achieve roughly thousand tps [137; 157]. The Ethereum code is open sourced under a GPL license [158].

3.4.2.3 Hyperledger

Hyperledger is an umbrella open-source project hosted by The Linux Foundation, created to favor cross-industry blockchain technologies [86]. At the moment of writing, Hyperledger consists of fourteen projects, six of which are distributed ledgers and the other eight projects are supporting modules [159]. There are more than 270 organizations in official the Hyperledger member community [136]. Considering that parties that join the network must be authenticated and authorized, Hyperledger frameworks are designed for permissioned blockchain applications (except the Sawtooth framework detailed hereafter). The Hyperledger Architecture Working Group identifies the following basic architectural components [159]:

i. *Consensus Layer*: responsible for verifying blocks of transactions and agreeing on their order.

ii. *Smart Contract Layer*: responsible for transactions processing [5] (proposal takeover, execution and validation).

iii. *Communication Layer*: responsible for P2P transport.

iv. *Data Store Abstraction*: responsible for different data-stores which can be used by other modules.

v. *Crypto Abstraction*: responsible for crypto algorithms.

vi. *Identity Service*: enables the establishment of a root of trust during setup of a blockchain instance, the enrollment and registration of identities during network operation, and authentication and authorization.

vii. *Policy Service*: responsible for policy management.

viii. *APIs* for interactions with applications.

ix. *Inter-operation Service*: in charge of supporting the inter-operation between different blockchain instances.

A trusted application distribution via smart contract bears a resemblance to well known state-machine replication techniques. However, there was a need for new designs considering that within blockchains many distributed application can run concurrently, deployed and run by anyone, potentially even maliciously written [65]. Hence, system performance with Hyperledger go significantly beyond the one of public and PoW-based blockchain frameworks; in fact, a computationally demanding PoW is not required [65; 160].

To the best to our knowledge, Hyperledger frameworks result from a first effort to make a modular blockchain platforms following the 'no one fits all' ideology. We detailed in the following the different Hyperledger frameworks.

a) *Fabric* [86] is the first proposal of hyperledger codebase, combining previous work done by Digital Asset Holdings, Blockstream's libconsensus, and IBM's OpenBlockchain. It provides a modular architecture, which allows components such as consensus and membership services to

[5] Among all the hyperledger frameworks, Indy is the only one which does not offer smart contracts.

be plug-and-play. An important feature introduced by Fabric is to allow nodes to confidentially transact on the same network of peers.

Fabric adopts the following terminology related to its work-flow; a blockchain 'application' handles the interface with the user and with the network. Smart contracts are called 'chaincodes' and are provided with a Node SDK, a Java SDK, and a command line interface – as development tools. Reading or writing the ledger is an operation referred to as a 'proposal'; it is built by an application via the SDK, and then sent to a blockchain peer, which processes it through a application-specific chaincode container. The chaincode then runs the transaction; if there are no issues, it endorses the transaction and sends it back to the application. An application, via the SDK, then sends the endorsed proposal to the ordering service, which packages many proposals from the whole network into a block that is then broadcast to the network peers. Finally, each peer validates the block and appends it to its ledger. The above described work-flow is referred to as an 'execute-order-validate' architecture [65] meant to go beyond more common 'execute-order' approaches; it is such that different groups of nodes have a different role in the network: clients which are submitting proposals, peers that validate transactions with a subset of them named 'endorsers' that execute all transaction proposals, and Ordering Service Nodes (or, simply, 'orderers').

Chaincode is written in Golang within Fabric v1.0, and is also available in Javascript in v1.1. Developers use chaincode to develop business contracts, asset definitions, and collectively-managed decentralized applications. Isolation between different chaincodes is guaranteed; assets created and updated by a specific chaincode cannot be accessed by a second one. Therefore, the chaincode needs to be installed on every peer endorsing a transaction. To develop smart contracts with Fabric, one can (*i*) code individual contracts into standalone instances of chaincode, or (*ii*) use chaincode to create decentralized applications that manage the life-cycle of one or multiple types of business contracts, letting the end users to instantiate instances of contracts within these applications. Interacting with the chaincode is done by using the gRPC [161], a client application that can directly call methods on a server application located in a different machine, as if it was a local object. A ledger is maintained using a local 'key-value' store (see Section 2.1.2) implemented by a LevelDB [162] (a key-value database implemented in Go) or Apache CouchDB [163].

Isolation between chaincodes is granted by *channels*: a channel can be seen as a completely separate instance of the Fabric; each channel is completely independent and never exchanges data with another channel, each of them having a different set of rules and policies. Fabric networks consist of peers incapable to communicate unless they are part of the same channel. Therefore, Fabric enables nodes of the same network to independently communicate with the predefined set of nodes in an isolated manner with respect to agreed policies.

In terms of latency, authors in [160] show that Fabric can achieve up to 10 000 tps and write a transaction irrevocably in the blockchain in around 0.5 s, even with peers spread in different continents.

b) *Iroha* [164] is contributed by several companies such as Soramitsu, Hitachi, NTT Data, and Colu. Its peculiarity is the emphasis on mobile application development, with client libraries for Android and iOS. Although inspired by Fabric, Iroha aims to complement other Hyperledger

projects, while providing a development environment for C++ along with the YAC consensus algorithm [165]. Written in C++, Iroha is build for high performance use-cases such as embedded systems.

c) *Sawtooth* [74] is mostly contributed by Intel. Unlike the others Hyperledger frameworks, it comes with support for both permissioned and permissionless deployments. It can use various consensus algorithms. By default, it uses a Proof of Elapsed Time (PoET) consensus (see Appendix .2.5.2), with the aim to provide the Bitcoin blockchain level of nodes scalability without its high energy footprint; PoET is suitable for permissionless blockchains and can be executed within the *Intel Software Guard Extensions (SGX)* [166] available in the most of newer Intel CPUs. For permissioned deployments, instead, BFT consensus is also made available (considering its already discussed advantages over PoET), and it does not rely on a single vendor hardware. Supporting deployments in which the blockchain network dynamically changes in size over time, Sawtooth was designed to enable on-the-fly change of the consensus protocol.

   Currently, any kind of EVM code can be compiled and run on Sawtooth. Along with the possibility to write smart contracts in Solidity, Sawtooth also provides a REST API and SDKs in several languages including Python, C++, Go, Java, JavaScript, and Rust for the development of applications which run on top of the Sawtooth platform. Sawtooth is licensed under an Apache License Version 2.0 software license [167].

d) *Indy* is still under incubation and not well documented to date. It is meant to support independent identity on distributed ledgers. The Indy code base, originally developed by Evernym, was donated to the Sovrin Foundation to establish a strong open source foundation for the Sovrin Network [168]. A goal is to create a global public utility for lifetime portable identity dedicated to any person, organization, or thing that does not depend on any centralized authority and can never be taken away. As already mentioned, it does not support smart contracts.

e) *Burrow* [169], formally known as eris-db, was released in December 2014. Currently under incubation, Burrow is a permissioned smart contract framework that provides a modular blockchain client with a permissioned smart contract interpreter built-in as part of the EVM specification. As of version 0.16, it has an Apache-licensed EVM implementation, initially licensed under GPLv3. It is functionally separated from the Ethereum protocol or any of the code bases implementing it. Any smart contract that is compiled by any EVM language compiler can be run in users' permissioned blockchain environments.

   The major components of Burrow are as follows:

   - *Gateway*: it provides interfaces for systems integration and user interfaces.
   - *Consensus Engine*: it serves to maintain the networking stack between the nodes and order transactions. The Tendermint consensus engine provides high transaction throughput over a set of known validators and prevents a blockchain from forking, hence it is currently used to implement consensus and p2p protocols. It is important to be aware that the Tendermint consensus engine comes from a separate blockchain framework detailed hereafter.
   - *Application Blockchain Interface (ABCI)*: it provides interface specification for the consen-

sus engine and smart contract application engine to connect.

- *Smart contract application engine*: it is the most important component, considering that it is in charge of transaction validation and of applying transactions to the application state according to an order provided by the consensus engine over ABCI.

Burrow is under active development and has currently released its version 0.27.0. The latest source code is licensed under Apache 2.0 license (available at [170]).

f) *Grid* [171] intends to provide reference implementations of supply chain-centric data types, data models, and smart contract logic based on industry best practices. Grid is an ecosystem of technologies, frameworks, and libraries that work together, letting users combine different components from the Hyperledger stack (the most appropriate according to their use-case) into a single solution.

The Hyperledger frameworks, examined so far, are used to build blockchains. Hyperledger open-source project also works on eight additional modules supporting these frameworks.

g) *Cello* [172] contributed by IBM, with sponsors from Soramitsu, Huawei, and Intel. It provides a toolkit that fulfills deployment of Blockchain-as-a-Service, allowing blockchains deployment to the cloud.

h) *Explorer* [173] contributed by DTCC, Intel, and IBM. It is a tool for visualizing blockchain operations. Designed to create a user-friendly web application, it can view, invoke, deploy, or query:

- Blocks.
- Transactions and associated data.
- Network information (name, status, list of nodes).
- Smart contracts (chain codes, transaction families).
- Other relevant information stored in the ledger.

The ability to visualize data helps to extract the value from it. Key components include a web server, a web UI, web sockets, a database, a security repository.

i) *Composer* [174] is contributed by Oxchains and IBM and is built in Javascript. It provides a set of tools for building blockchain networks enabling to:

- Model your business blockchain network.
- Generate REST APIs for interacting with your blockchain network.
- Generate a skeleton Angular application.

j) *Caliper* is a benchmark platform allowing users to measure the performance of a specific blockchain implementation with a set of predefined use-cases [175].

k) *Quilt* [176] is an implementation of the Interledger Protocol (ILP) [177] responsible for ledger systems interoperability by enabling transactions across ledgers.

l) *Aries* [178] extends the applicability of Indy technology beyond its current community components from the Hyperledger stack into a single solution. It provides a shared cryptographic wallet for blockchain clients rather than just an UI, and a communications protocol for their off-ledger interactions. It is not a blokchain but rather a shared infrastructure of tools that support peer-to-peer messaging and interactions among different DLTs. Note that the cryptographic support is provided by a separate Hyperledger project (Ursa [179]).

m) *Ursa* [179] is a shared cryptographic library. It has been created to allow all Hyperledger collaborators to work on the same cryptographic code. This enables many different projects to adopt the same code base for open-source, secure, and pluggable cryptographic implementations.

n) *Transact* [180], still in the incubation phase, aims to provide a standard interface for executing smart contracts that is separate from the distributed ledger implementation by way of a shared software library.

3.4.2.4 Corda

Corda [35] is a permissioned blockchain framework, created by the software company R3 that leads a consortium of two hundred global financial institutions. Unlike solutions we have examined so far that involve a global availability of data across the network and several validators, Corda only allows information access and validation functions to those parties actually involved in a given transaction. It enables consensus at the level of individual deals, instead of at the system level.

Nodes identities in Corda are attested by a X.509 certificate signed by a permissioning service called "Doorman" that each Corda network has. Unlike most of the other permissioned blockchain platforms, Corda does not order all transactions as one single virtual execution that forms the blockchain [87]. Instead, it defines states and transactions, where every transaction consumes (multiple) states and produces a new one. Only nodes affected by a transaction store the new state. Seen across all users, this transaction execution model produces a hashed directed acyclic graph or Hash-DAG [181]. Transactions must be valid – i.e., endorsed by the issuers and other affected nodes – and correct according to the underlying smart contract logic governing the state. Each state points to a notary responsible for ensuring transaction uniqueness, i.e., each state is consumed only once. The notary is a logical service that can be provided jointly by multiple nodes. The type of a state may designate an asset represented by the network, such as a token or an obligation, or anything else controlled by a smart contract.

A transaction in Corda consumes only states controlled by the same notary; hence, one notary by itself can atomically verify the transaction's validity and uniqueness to decide whether it is executed or not. To enable transactions that operate across states governed by different notaries, there is a specialized transaction that changes the notary. In view of the fact that each node stores only a part of the Hash-DAG, it is only aware of transactions and states that concern the node. This contrasts with most other blockchain frameworks and provides a means for partitioning the data among the nodes. As

is the case for other smart contract platforms, transactions refer to contracts that can be programmed in a universal general-purpose language.

A notary service in Corda orders and timestamps transactions that include states pointing to them. A notary service needs to cryptographically sign its statements of transaction uniqueness, such that other nodes in the network can rely on its assertions without directly talking to the notary. Currently, there is support for a notary service as a single node (centralized), for a distributed crash-tolerant implementation using RAFT [95], and for distributing it using the open-source BFT-SMaRt toolkit [182], an open-source Java-based library implementing robust BFT state machine replication. When using RAFT deployed on *n* trusted nodes, a Corda notary tolerates crashes of any $t < n/2$ of these nodes. With BFT-SMaRt running on n nodes, the notary is resilient to the subversion of $f < n/3$ nodes. Let us recall that RAFT consists in a crash tolerant consensus algorithm while BFT-SMaRt support also Byzantine failures. Corda runs in a JVM with the support for Oracle JDK 8 implementation, other are not actively supported. Applications on Corda called CorDapps can be written in any language targeting the JVM. However, Corda itself and most of the samples are written in Kotlin language, with recommendation to use IntelliJ IDEA, due to the strength of its Kotlin integration.

In Corda P2P network, each node is a JVM run-time environment hosting Corda's services and executing CorDapps. Communication between nodes via TLS-encrypted messages (sent over AMQP/1.0) enables the data sharing only on a need-to-know basis without global broadcasts. A network map service publishes the IP addresses through which every node on the network can be reached, along with the identity certificates of those nodes and the services they provide. The data model used in Corda is UTXO$^+$ (see Section 2.2.1.3).

From a transaction throughput perspective, experimentally it was reached thousands tps, using RAFT consensus, with 3 cluster members and Kafka distributed log [183], even if nominally it is meant to be around 120 tps.

The Corda code is licensed under Apache 2.0 [184].

3.4.2.5 Tendermint

Tendermint [185] is an application-oriented framework that consists of two components:

i. A blockchain consensus engine called Tendermint Core, which ensures that same transactions are recorded on every machine in the same order.

ii. A generic application interface called the Application BlockChain Interface (ABCI), which enables the transactions to be processed in any programming language.

Unlike other solutions, which usually come with built-in state machines, Tendermint can be used for BFT state machine replication of applications written in any programming language. Originally, Tendermint had a simple currency built-in, and to participate in consensus, users have to use the currency for a security deposit that can be revoked if they misbehave. Since then, Tendermint has evolved to be a general purpose blockchain consensus engine that can host arbitrary application states: it can be

used as a plug-and-play replacement for the consensus engines of other blockchain frameworks. An example of a cryptocurrency application built on Tendermint is the Cosmos network, a decentralized network of independent parallel blockchains; the first blockchain in the Cosmos network is the Cosmos hub using Tendermint as an underlying consensus engine [88].

Tendermint-core blockchains offer strong consistency (no forks) in an open system relying on two key ingredients: (*i*) a set of validators that generate blocks via a PBFT variant, and (*ii*) a rewarding mechanism that dynamically selects nodes to be validators for the next block via PoS [186].

In contrast to basic PBFT, where the client sends a new transaction directly to all nodes, the clients in Tendermint disseminate their transactions to the validating nodes using a gossip protocol. The biggest divergence is the technique first used by the Spinning protocol [187]: rotation of the leader after every block. The Tendermint Socket Protocol (TMSP) defines the core interface by which the consensus engine communicates with the application state machine: separation between consensus and its actual execution in the state-machine is achieved. First, consensus on the transactions order is reached, then the ordered transactions are executed, which improves the system's fault tolerance [188]. Indeed, while we still need a two-thirds majority for ordering $(3f+1)$, we only need a one-half majority for execution, to tolerate f Byzantine failures $(2f + 1)$. However, applications built using TMSP must be deterministic. A client connects to a Tendermint consensus network through a proxy, which may be hosted by provider or run locally. The proxy enables client transactions to be broadcasted to the network via the gossip layer. Note that Tenrdermint contains additional mechanisms that prevent a livelock bug [189], pertaining to locking and unlocking votes by validators.

As a peculiarity in terms of P2P communications, a Peer Exchange (PEX) protocol gossip is used to enable dynamic peer discovery. Each node broadcasts its current state every time it changes, optimizing the gossiping of messages to only needed information which they do not already have.

In terms of delay performance, Tendermint can achieve thousands tps on dozens of nodes distributed around the globe [189], with latencies of about one second, and performance degrading moderately in the face of adversarial attacks. Within a single local-area data-center deployment, Tendermint is capable of tens of thousands tps.

At the moment of writing Tendermint is still subject to various fixes. The source code is written in GO and licensed under Apache 2.0 [190].

3.4.2.6 Chain Core

Chain Core [63] is a permissioned blockchain framework, mostly focused on issuing and transferring financial assets within a consortium.

An asset is any type of value that can be issued on a blockchain. Units of an asset are fungible and can be transacted directly between parties without the involvement of the issuer. Each asset has a globally unique asset ID that is derived from an issuance program. In order to issue new units of an asset, the issuance program defines a set of possible signing keys and a threshold number of needed signatures; the rules for spending them must comply with the control program. Chain Core follows the

UTXO model. A program is written as a set of byte-code instructions for the Chain Virtual Machine (CVM). The CVM is a stack machine: each instruction performs operations on a data stack, usually working on the items on top of the stack. Cryptographic SHA256 and SHA3 instructions execute the corresponding hash functions. The CVM instruction set is Turing complete. In order to control the use of computational resources, the protocol allows networks to set a run limit that a program is not allowed to exceed [63]. Simple instructions consume less resources due to a lower cost, while processing-intensive instructions, such as signature checks, are more expensive.

Security against forks is enforced by the Federated Consensus [63]; it guarantees safety as long as at least $2m - n - 1$ block signers obey the protocol. The latter guarantees liveness as long as the block generator and at least m block signers follow the protocol. Due to the network need to evolve, the set of participants and the number of required block signatures can be configured differently for each blockchain network. The aim is to provide takeoffs between liveness, efficiency, and safety, giving the possibility to tune those parameters in respect to the current situation. The Federated Consensus protocol is executed by the n nodes configuring one of them as statical 'block generator'. It periodically selects a number of new, non-executed transactions, assembles them into blocks, and submits the block for approval to 'block signers'. Every signer validates the block proposed for a given block height, checking the signature of the generator, validating the transactions, and verifying some real-time constraints and then signs an endorsement for the block. Each signer endorses only one block at each height. Once a node receives q such endorsements for a block, the node appends the block to its chain.

Federated Consensus is a special case of a standard BFT protocol, operating with a fixed block generator. Indeed, under assumption that the blockchain generator operates correctly, Federated Consensus reduces to an ordinary Byzantine quorum system that tolerates f faulty signer nodes when $q = 2f + 1$ and $n = 3f + 1$. However, the protocol cannot prevent forks if the generator is malicious, e.g. by signing two different blocks for the same block height, making it single point of failure, which is not in scope with blockchain ideology. Even if the generator simply crashes, the protocol halts and requires manual intervention.

In the P2P overlay, in order to connect a user must know blockchain IP access addresses and must have been granted a network token from the Block Generator, which grants access if the token is provided. A node can then download the latest blockchain data from the Block Generator.

To the best of our knowledge, there is still a lack of performance analysis in literature about Chain Core, leaving a need for a formal and comprehensive evaluation.

Client libraries for Chain Core are available for the Java, Node.js and Ruby platforms. Chain Core Developer Edition is open source [191] and licensed under AGPL. A public testbed is made available for experimenters, operated by Chain, Microsoft and Cornell University [6].

It is worth noting that there is a new stack-based called TxVM (transaction virtual machine) [193]

[6]Nevertheless, according to GitHub repository, development and support have ended, encouraging a transition to Sequence, a new cloud blockchain infrastructure [192] where ledgers are managed as a service, therewith all transactions must be signed by the adequate keys controlled by the users (that have particular authority, disabling Sequence to access them). SDKs are available for Java, Node.js, and Ruby.

recently proposed as a new transaction model for Chain. It offers Turing-complete virtual machine to execute transaction programs. Each transaction is executed as a separated TxVM isolated from the blockchain state, providing as an output a deterministic log of proposed state changes. This approach avoids unexpected side effects in other contracts, even runs them in parallel. Its code is licensed under Apache 2.0.

3.4.2.7 Quorum

Quorum [89] is a permissioned implementation of Ethereum. It includes a minimalistic fork of the Go Ethereum client, leveraging the work done by Ethereum community including the account-based data model.

The Ethereum P2P layer was modified to allow connections only to a group of permissioned nodes. Thus, the platform is designed to support both, 'transaction-level privacy' and 'network-wide transparency' [45]. Although Ethereum Gas remains, its pricing was removed.

Within Quorum, smart contract execution is done with the EVM. Instead of a PoW-based mechanism, a voting-based consensus algorithms is implemented to facilitate a smart contract platform. Currently, it comes with two consensus choices: QuorumChain and a RAFT-based one. Data confidentiality is achieved within the network by allowing data visibility on a 'need-to-know' basis. There are two transaction's types. A 'public' one readable by all nodes, and a 'private' one, with transaction data encrypted by the public key of a receiver, i.e., readable only by nodes which participate in the transaction.

QuorumChain includes a group of 'voter' nodes – in charge of transaction execution in order to validate blocks – and, a certain number of 'maker' nodes (minimum is one). Only 'block-maker' nodes, whose identities are known to the whole nodes community, can propose a block. In order to avoid simultaneous block creations by several 'makers' at the same time, each maker node sets a random time slot and will propose a new block after it expires, sign it and send it to the rest of the network. 'Voters' validate transactions and send their votes in favor of blocks that they ensure are correct making an Ethereum transaction to 'BlockVoting' contracts distributed in all nodes. Hence, they are executing transactions in the blockchain network and hence facilitate consistency. Each block having more votes than a threshold is added locally in the chain at all nodes. Since votes for a given block are sent via standard Ethereum transactions, they can only be counted when the next block is created.

In terms of P2P dissemination, Quorum originally leverages on the Ethereum's gossip layer. A network set up with one 'maker' by default could lead to network inconsistencies (chain forks) unless the network is perfectly synchronized. This node can be seen as a single point of failure, and if this node crashes, the protocol halts. Byzantine fault in a 'maker' or a 'voter' node can result in inconsistencies and protocol's disruption. Additionally, the protocol relies on synchronized clocks for safety and liveness. Due to those facts, authors in [45] states that the protocol cannot ensure consensus in any realistic sense.

Eventually, QuorumChain was removed in Quorum 2.0, with an impact on dissemination. Another consensus choice was made available: a popular variant of Paxos [94], based on the RAFT protocol [95]. Available in many open-source tool-kits, Quorum uses the implementation in etcd [194]. RAFT is in charge of transactions replication to all participating nodes and to ensure that each node locally outputs the same sequence of transactions, tolerating any $t < n/2$ of the n nodes crash. Blocks are communicated over the HTTP transport layer built into etcd RAFT instead of the P2P protocol built-in to Ethereum. Quorum community argues they found by testing the default etcd HTTP transport to be more reliable than the P2P network (at least as implemented in geth) under high load. The maximum number of peers is configurable, with a default number set to be 25. One of the medium term goal is a pluggable consensus feature as stated in the project roadmap.

In terms of performances, there is definitely a gap to fill in the literature. To the best of our knowledge, there is only a vague estimation reported in the JPMorgan website stating that network can process dozens to hundreds tps, depending on how the network and smart contracts are configured, leaving a space for more precise performance analysis. Quorum is open sourced and maintained by JPM [195].

3.4.3 Frameworks discussion and related works

Platform	Bitcoin	Ethereum	Hyperledger platforms	Corda	Tendermint	Chain Core	Quorum
Common Features							
Data encription and hashing ⇒ data confidentiality and integrity Digitial signature ⇒ data authenticity and non-repudiation Auditability, immutability, open sourced code							
General Features							
Identity and membership	✗	✗	✓	✓	✓	✓	✓
Major usage	Public payment system	Generic blockchain platform	Modular blockchain platforms	Specialized distributed ledger platform for financial industry (digital assets)	blockchain consensus engine	multi-assets ledger designed for assets trading	general application platform
Cryptocurrency	Bitcoin	Ether cryptocurrency Tokens via smart contract	Currency and tokens possible via chaincode	✗	At first, but now ✗	✗	✗
Governance	N/A	Ethereum developers	Linux Foundation	R3	Tendermint developers	Chain, Microsoft, IC3	JPMorgan
Architectural Features							
Data model	UTXO	Account based	Key-value	UTXO$^+$	various	UTXO$^+$	Account based
Smart contracts execution	native	EVM	Fabric: docker, Sawtooth: native	JVM	various	Chain Virtual Machine (CVM), TxVM	EVM
Smart contract language	not Turing-complete	Solidity, Serpent, LLL	Fabric: GO & Javascript, Sawtooth: Java, Go, JavaScript, Rust or Solidity	Kotlin, Java	depends on software choice	written in bytecode instructions for the CVM	Go
Modularity	✗	✗	✓(consensus, membership services)	✓(consensus)	✗	✗	✗
Consensus protocol	Mining, PoW ledger level	PoW, POS transaction level	Various	RAFT (centralize), BFT via BFT-SMaRt toolkit	BFT	BFT - The Federated Concensus	QuorumChain, RAFT-based
Adversary model	50%	50%	BFT: 33%, PoET: Trusted Hardware	RAFT: 50%, BFT: 33%	33%	33%	RAFT based: 50%, Quorum chain 33%
Execution	sequentially on all peers	sequentially on all peers	parallel	sequentially on all peers	sequentially on all peers	sequentially on all peers	sequentially on all peers
Architecture	order-execute	order-execute	execute - order-validate	order-execute	order-execute	order-execute	order-execute
Node isolation	✗	✗	Fabric via channels	✗	✗	✗	✗
Dissemination	flooding	gossip (ĐΞVp2p)	gossip	gossip	gossip	gossip	gossip–v.1.x HTTPS–v.2.x
Throughput	7 tps	15-40 tps; in private setup ∼ thousand tps	dozen of thousands tps [65; 160]	120-1000 tps [183]	tens of thousands tps within single data-center [189]	N/A	dozens to hundreds of tps
Latency	600 sec	∼ 15 sec	< 1 sec	N/A	< 1 sec	N/A	N/A
Source Code	[153].	[158].	Sawtooth [167], Fabric [196], Indy [197], Iroha [198], Burrow [170].	[184].	[190].	[191].	[195].

Table 4.6 compares the presented frameworks according to (*i*) their features (analyzed in Sections 2.2 and 2.3) and, (*ii*) the characterizing aspects of the different abstraction levels (Fig. 3.4). We summarize in the following, these interesting aspects which may help the reader to choose the right platform to consider.

A. Cryptocurrency
Build-in cryptocurrency is the main ingredient within distributed public payment systems like Bitcoin and Ethereum. Even though permissioned blockchains do not require a build-in cryptocurrency, Hyperledger Fabric still ensures the possibility for a native currency or a digital token developed with 'chaincode'. Indeed, the common for all analyzed platform is that they ensure ledger's auditability and immutability.

B. Node roles
In different frameworks, nodes assume different roles and tasks in the process of reaching consensus. While in Ethereum and Bitcoin where roles and tasks of nodes participating in reaching consensus are identical, within Fabric, nodes are differentiated based on whether they are clients, peers or orderers. The motivation was to bypass architectural limitation with classical 'order-execute' architecture, considering that reaching consensus and state synchronization across all nodes do not require that all smart contracts are executed on all nodes. Instead, it is important to propagate the same state to all nodes.

C. Execution
The limitation raised from a sequential execution is a performance bottleneck. Indeed, authors in [65] show that Hyperledger Fabric, overcoming the stated limitation, achieves end-to-end throughput of more than 3500 tps in certain deployment configurations.

D. Performance
While blockchains may appear similar to legacy distributed storage systems, they provide some specific differences. They are typically implemented to support large scale data repository. Within the blockchain, the number of nodes increases the resilience of the system in terms of integrity and availability, however with a loss of performance. Such a trade-off can be complicated to assess even when all nodes have the same role in the system, and therefore can be even more difficult for those blockchains that further specialize the roles of nodes (e.g., Hyperledger platforms). To help precisely and consistently evaluate the unique performance attributes of blockchains, it is necessary to define relevant terms and metrics. In terms of performance comparison between platforms, the main difficulty is to find a way to fairly compare them given the fundamental differences touching to consensus, block structure, P2P behaviors, etc. Some trials in this direction exist. Authors in [157] describe the "BLOCKBENCH", an evaluation framework for analyzing private blockchains with Turing-complete smart contracts, releasing it as open source. BLOCKBENCH was used to conduct evaluation of the following blockchains: (*i*) Ethereum, (*ii*) Parity and, (*iii*) Hyperledger Fabric. They report the performance gaps attributing them to specific design choices at different layers of the blockchain's software stack. The results published in [137; 157] show that Hyperledger Fabric outperforms Ethereum in terms of evaluation metrics such as execution time, latency and throughput. Yet, pertinent metrics to measure performance of different blockchain projects are to be designed. The Hyperledger Performance and Scale Working Group (PSWG) published a white paper [199] with the goal to ensure that the performance and scalability of all blockchain projects are measured in a fair and equitable manner using metrics that are defined, gathered, and reported in a consistent way; it focuses on blockchain performance associated metrics, rather than on benchmarking. Indeed, benchmarking is more controlled than performance evaluation, thus [199] can be seen as a first step to guide development of any formal benchmarks.

E. Smart contract language

About programming language, Corda differentiates from the others in the semantic of a smart contract: besides the code, additionally legal prose can be found. The rationale behind this is to give the code legitimacy that is rooted in the associated legal prose. Such a construct is called a Ricardian Contract. Hence, meant to be used by highly regulated environment of the financial services industry, Corda was designed accordingly. Both, Fabric and Ethereum do not possess this feature as they rather aim to be a general purpose blockchain system.

F. Consensus
Permissioned blockchains mostly rely on asynchronous BFT replication protocols while their permsionless ancestors usually use PoX algorithms which are more suitable for an open-access mode. Most of the platforms come with a hard-coded consensus except the Hyperledger. This implies that in case of different fault models, one must switch on a different blockchain environment. Thus, plug-and-play consensus such as one employed by Hyperledger is particularly interesting. What further makes Fabric unique are the channels and related isolation; a consensus can only be reached at transaction level and not at ledger level as with the other platforms. Corda consensus is also reached at the transaction level, by involving only parties that participate in that transaction, employing also a 'pluggable' consensus, while nodes store only the transactions they participate to.

G. Security
With respect to physical security, some DLT systems are leveraging trusted hardware as a trade-off between cost of security and performance [152]. Most overheads of used algorithms arise from the assumption that nodes could have Byzantine manners. In particular, Endorsement key pair (EK) used for encryption, never visible outside trusted platform modules, is burnt into each device during manufacturing [200]. Nodes equipped with trusted hardware can be verified for certain properties, which makes it possible to use weaker trust model with the aim to improve performance. Security of those systems particularly depends on a trusted computing base that is running within specific hardware such as Intel SGX [166] and ARM TrustZone [201].

3.4.4 Architectural limitations

Public blockchain platforms have been criticized more than permissioned ones, in terms of architectural limitations. Table 3.3 summarizes the most evident limitations, differentiating between those shared between permissionless and permissioned systems and those specific for permissionless systems. In the following we focus on the former, as the latter were already covered by previous sections. Architectural limitations for permissioned systems are fully analyzed in [136] without considering, however, that some limitations also characterize open blockchains. In fact, those also apply to permissionless systems. Some of the presented platforms incorporate solutions for some architectural limitations (e.g., Hyperledger parallel execution). Hence, flexibility and limitations can be considered as selection criteria for a given blockchain framework.

A. Sequential execution
The active SMR, used in the majority of blockchain frameworks, requests an application to be ordered at first by the consensus, and then executed sequentially at all nodes. This can be addressed to both permissioned and permissioneless systems, such as Ethereum, a pioneer in this approach. One of its

Table 3.3: Architectural limitations of blockchain

Permisionless	Permissioned
Limited capacity	✓
Transaction cost	✓
Irrelevant data	✓
Mining risk	✓
Lack of privacy	✓
Non-deterministic execution	
Sequential execution on all nodes	
Trust model flexibility	
Hard-coded consensus	
Trusted hardware	

biggest limitation is the throughput upper bound, since the throughput and latency of execution are inversely proportional. Furthermore, smart contracts, designed to take a very long time to execute, can lead to a denial of service (DoS) attack on the network. Thus, cryptocurrency based blockchains had to introduce solutions such as *gas* or its own virtual machine like Ethereum, with the aim to control all execution steps. Intentional crypto-money fees and smart contract language limitation (due to the specific VM environment) hold back its wide adoption. Different approaches proposed by Ethereum, Hyperledger and Chain Core aim at overcoming the drawbacks related to sequential execution, such as multi-core computing, parallel execution and sharding[7], still under test.

B. Non-determinism
Smart contracts can revoke consensus hence leading non controllable side effects such as ledgers 'forks'. Adding determinism-oriented features in smart contract design is an unexplored research direction as of our knowledge.

C. Execution on all nodes
The process consumes computational resources that might be saved. In addition, many use-cases require that a transaction logic is revealed only to certain nodes. In order to reach consensus and synchronize the network, it is sufficient to propagate the same state to all nodes and execute smart contracts on a subset of them [136]. This lead to architectures such as Hyperledger Fabric, which executes a smart contract on a specified subgroup of nodes while ensuring propagation of the same state to all of them. Yet, such approaches open questions about nodes liability. How one can choose an adequate subset of trustful nodes? And how many of them? How to attribute roles to nodes? Such questions do not find clear answers in the literature, yet.

D. Trust model flexibility
Modern blockchain architectures should be designed to decouple application trust assumption from underlying consensus protocols. Adversary models such as 'f out of $3f + 1$' may not match the specific application trust model.

[7]Method to partition a database in small pieces (i.e., shards) that can be recomposed to regenerate the original database.

E. Hard-coded consensus
It is not an optimal solution, as there is no such consensus protocol that fits all scenarios. Changing the hard-coded consensus protocol is very difficult, so plug-and-play consensus engines seem to be an adequate solution. This can give developers different options to adopt due to specific needs. Nevertheless, the security consequences and related vulnerabilities due to automated consensus mechanism swap are unknown to date.

F. Trusted hardware
It represents one possible way to increase performances [202] while allowing a weaker trust model, typical of permissioned implementation. Nevertheless it may lead to specific vendor monopoly. This is a completely separate research space mixing computer science and electrical engineering disciplines.

3.4.5 Blockchain as a Service

The reviewed blockchain frameworks are open source. Nevertheless, commercial services make surface offering a blockchain platform, or Blockchain as a Service (BaaS). A BaaS is a service that allows customers to leverage cloud-based solutions to build, host and use their own blockchain apps, while the service provider is responsible to manage the infrastructure and keep it agile and operational. A BaaS is essentially a Software As A Service (SaaS) service, helping the blockchain adoption across businesses used to liability commercial chains. Table 3.4 surveys existing BaaS providers, related technology and corresponding references.

Table 3.4: Blockchain as a Service

Providers	Supported frameworks
Microsoft [146]	Hyperledger Fabric. Ethereum. Corda Quorum, Chain. BlockAps.
IBM [203]	Via Bluemix: Hyperledger Fabric.
SAP Cloud [204]	MultiChain, Hyperledger (Leonardo program).
HP [205]	Via HP Enteprise: Corda.
Oracle [206]	Hyperledger Fabric.
Amazon [207]	Via AWS: Ethereum; Hyperledger Fabric, Burow.
Huawei [208]	Hyperledger Fabric.
BitSe [209]	VeChain.
BLOCKO [210]	Coinstack.
Baidu [211]	Proprietary technology.

3.4.6 Use-case applications

After having explored the When and Which questions of the vademecum, let us present some existing blockchain applications in the recent state of the art, applying the proposed vademecum logic. We report two use-cases in (*i*) networking, and (*ii*) supply-chain respectively adopting (*i*) *permissionless*, and (*ii*) *open-permissioned* blockchain implementations.

3.4.6.1 Decentralized Internet storage

Despite its high potential for decentralized communications, the current Internet infrastructure management suffers from centralization of control and data operations. The data is often stored on big server farms usually controlled by a single entity. The availability in data access can not be guaranteed to be high due to various security, reliability and censorship issues. Indeed, there is a need for a decentralized shared storage in a trustless environment. Filecoin [145] is a blockchain-based file system, built on top of the InterPlanetery File System (IPFS) protocol [212] (a peer-to-peer protocol to share hypermedia), which goes in this direction. Let us develop the vademecum on the Filecoin use-case.

- *Q1: Do you need to store and share a ledger state?* Yes. Filecoin is meant to be a blockchain-based cooperative data storage and retrieval system thus, data needs to be stored in a shared ledger and updated.
- *Q2: Are there multiple potential writers?* Yes. Nodes in the network share files or proactively distribute them. Content based addressing and decentralization make data access resistant to censorship, failures, or attacks.
- *Q3: Who do you entrust with the ledger maintenance?* The ledger maintenance is entrusted to the entire public community. Filecoin is a fully open and decentralized system to which all network users have both access and permission, participating in the consensus procedures.

Therefore, according to the vademecum logic in Fig. 3.1, decentralized data storage not requiring data confidentiality (question 4.A) such as the one served by Filecoin is to be achieved via a permissionless blockchain. However, in Filecoin, the scalability and performance limitation of permissionless platforms (see Table 4.6) lead to the choice of recording in the blockchain the data hash only, with therefore a dedicated platform developed for Filecoin. IPFS protocol using content based addressing (i.e., one should know what to search) stores original data off-chain (in multiple 256 KB objects containing the links of each other). With data hash in the blockchain one can fetch the data content from IPFS.

3.4.6.2 Industrial IoT-based supply-chain

Supply chain management (SCM) is the process involving the transitions between the different actors characterizing the life cycle of a product, from producers to end-consumers. Often the communication between the different actors in supply-chain results inefficient [213], because actors in the process do not have access to products' information in its entirety. Moreover, detecting failures in the supply-chain proves to be difficult and expensive. Blockchain transparency can help to significantly improve

the SCM procedures while at the same time enabling actors (especially end-consumers) to monitor and trace the products transitions via IoT devices. Let us report the vademecum steps for the food supply chain traceability systems proposed in [131; 132; 214] (for agri-food products); the '*from-farm-to-fork*' logic becomes reality by leveraging blockchain and IoT's technologies such as RFID (Radio Frequency IDentification).

- *Q1: Do you need to store and share a ledger state?* Yes. Supply chains are characterized by input-output relationships among different actors transferring information on the product. The latter need to be registered (in a ledger) and communicated to the actors of the product life-cycle.
- *Q2: Are there multiple potential writers?* Yes. SCM is characterized by several interacting actors such as: providers, producers, processors, distributors, retailers, consumers, via IoT devices for some among them. All the actors that interact with the product and change its state record on the blockchain such a change.
- *Q3: Who do you entrust with the ledger maintenance?* A group of selected actors, i.e., SCM actors that maintain the ledger by recording information on it, automatically verifying and validating what has been declared by means of IoT devices.

  Q3.b) Do you need the ledger to be publicly verifiable? Public verifiability guarantees the authenticity, the integrity and the reliability of the shared information in a trustless environment where SCM agents can monitor, trace and manage the safety and the quality of the product.

When-Which part end-state consists in an open-permissioned blockchain implementation. According to the How vademecum guidelines, both Hyperledger and Quorum fit the SCM use-case for settings, data structure and, performance level. More precisely, the IBM Food chain [214] operates in *Sawtooth* while the other contributions [131; 132], still at a PoC level, consider both platforms.

3.5 Conclusion

For a new technology to realize its full potential, a lot of circumstances need to co-exist before network effects can be realized. In order for the technology to bring in systemic efficiencies, a critical mass needs to be attained. As an infrastructure technology, all major players in the market need to collaborate to define standards in a democratic manner. The blockchain community is indeed witnessing unprecedented levels of industry collaboration between players who are otherwise competitors in the space. Because of the cost of moving from one infrastructure technology to the next, an open source collaborative approach is the most promising way forward. This is the direction we insisted in this chapter, highlighting not only when and which blockchain technologies should be chosen, but also how they can be used and deployed.

From a societal perspective, while there has been an exponential increase in the interest around blockchain technologies, there is a huge lack of technical experts. Currently, blockchain engineers become one of the most payed and required jobs, yet there are no officially recognized courses to train engineers to fulfill the existing lack of blockchain experts. Both industry and academia started to think in this

direction providing some online courses, but it seems there is still a need for new and more comprehensive schooling and literatures. As an illustration of the current societal perspective on blockchain, due to ongoing innovation and development in the blockchain space, there is still not a consensus on a clear blockchain definition [215], despite we tried in this work to clarify key properties of blockchain, somehow giving an axiomatic view on possible different blockchain definitions. This chapter provides extensive details of our work presented in our publish work [52; 216].

Chapter 4

Virtual Machine Orchestrator Authenticator (VMOA)

Summary

The blockchain technology is gaining momentum because of its possible application to other systems than the cryptocurrency one. Indeed, blockchain, as a decentralized system based on a distributed digital ledger, can be utilized to securely manage any kind of assets, constructing a system that is independent of any authorization entity. In this chapter, we present the VMOA blockchain to secure virtual machine orchestration operations for cloud computing and network functions virtualization systems. Using tutorial examples, we describe our design choices and draw implementation plans. We

describe how transactions are identified and managed in B-VMOA. We further develop the vademecum logic applied to cloud orchestration and how it can lead to precise platform specifications. We capture the key system operations and complex interactions between them. To demonstrate its feasibility, we design and implement B-VMOA Proof of concept (PoC) with the aim of verifying its practical potential. This chapter provides extensive details of our work presented in patent application [217] and in our published work [128].

4.1 State of the art

IaaS (Infrastructure as a Service) is the most cloud service that leverages on the provisioning of virtual machines in a virtualized data-center environment, to which users typically access using the Internet. Users use IaaS services to remotely operate their IT infrastructure, taking profit from high-availability guarantees of cloud services, real estate saving, and competitive billing models. Standard cloud computing infrastructures are nowadays composed of servers with compute and storage capabilities, and they are being enhanced to also support virtual network management in the frame of the Network Functions Virtualization (NFV) evolution [218]. Furthermore, the computing infrastructure comprises network connections, Internet transit bandwidth, network layer and data-link layer addressing and routing protocols, virtual network controllers, control network, load balancers and other middle-boxes. In data-center virtualized environments, using full virtualization allows having a full decoupling between the operating system of the VMs and the one of the physical machine. In such cases, the physical machine a VM manager (VMM) process runs with kernel privileges to manage hardware sharing by enabling execution of multiple VM environments isolated from one another. Moreover, it provides the functionality needed to execute entire operating system (OS).

Virtualization servers run virtual machines in such a way that they can be dynamically migrated, duplicated, scaled in-out or up-down, turned on or off, accordingly to arbitrary orchestration policies, acting at one or multiple distinct computing resources, concurrently. The interconnection network resources between servers can also be made programmable to serve dynamic re-allocation tasks and the necessary abstraction and decoupling to IaaS communications (cloud access and inter-VM traffic) and computing facilities. A fully virtualized infrastructure allows a decoupling between the operating systems of the VMs and the operating system of the virtualization server physical machine. In such a case, the physical machine possesses a VM Manager (VMM), or hypervisor, running with kernel privileges to allow the management of multiple isolated VMs. The VMM offers specific functions to execute several distinct operating systems; the VMM receives instructions about how to create, destroy, modify, migrate VMs and reallocate resources either manually or automatically, from the command line interface or using orchestration protocols. When an external orchestrator is in charge of managing multiple physical machines acting as virtualization servers, there is the need to secure the communication between the orchestrator and the physical machine VMMs, in order to guarantee that a VM management command is authorized and legitimate. In fact, these architectures are very sensitive to attacks that can come from different horizons. For example, a virtual machine can be created by an attacker to run in a server and used to perform external DDOS attacks, and also internal attacks against data integrity and confidentiality. The goal of the Virtual Machine Orchestration Authentification (VMOA) method we

describe in this chapter is to protect and secure virtual machines, which in essence are software that is difficult to protect.
Usually, authentication is achieved through third party mediation. The third party must be trusted, yet it also represents a vector of attacks against the integrity of orchestration commands. Having a fully decentralized authentication logic is therefore a rising requirement for such environments. In this respect, blockchain is a technology meant to store, read and validate transactions in a decentralized fashion. Recently, various research efforts are carried out into this direction, in the communications networking field in particular as we discussed in [52]. We aim to present how blockchain can be used to secure cloud/NFV orchestration operations and in particular to enhance the authentification of orchestration commands in the lifecycle of cloud services.

4.2 Blockchain application to cloud orchestration

After having explored the When, Which and How questions of the vademecum in the previous chapter, let us present the blockchain applications to cloud orchestration, applying the proposed vademecum logic. We investigate how blockchain could be used as a way to secure the orchestration interface between cloud orchestrators and computing elements. The idea could easily be extended to SDN switches configuration from SDN controllers, knowing that a network switching instruction likely requires a lower latency than a virtual machine or container orchestration instruction. The idea is to translate cloud/network orchestration instructions sent from an orchestrator or a controller (i.e., virtual machine or switching rule instructions) to transactions that ought to be authentified in a decentralized way by a pool of agents integrated with compute, orchestration, or network elements. Applying the vademecum chart (Fig. 3.1) let us examine system we proposed in [128]:

- *Q1: Do you need to store and share a ledger state?* Yes. In the envisioned cloud environment, the orchestrator is an intermediate node in which one must have trust, thus it can be seen as a single point of failure or attack from a security standpoint. Orchestration instructions are translated into transactions to be recorded and authentified, hence need for a shared ledger state.
- *Q2: Are there multiple potential writers?* Yes. The architecture accounts for frequent transactions to be traded and shared by a pool of orchestrators, each possible in charge of one or overlapping domains and network elements, and network elements can also take part to the orchestration environment.
- *Q3: Who do you entrust with the ledger maintenance?* In a cloud infrastructure environment, network participants are whitelisted, thus a group of selected actors (orchestrators, compute nodes, network switches) is entrusted to maintain the ledger.

  Q3.b) Do you need the ledger to be publicly verifiable? The system consists of several validators known to the network, hence there is no need to have a system verificable by all the public community.

  T3.b): According to Fig. 3.1 one comes to a trade-off point. Despite the fact that legacy databases offer better performances in terms of scalability, throughput and latency, the proposed

systems aims to enable ledger replication across the network while benefiting from data immutability. To reinforce security, orchestration logic can be presented as a smart contracts logic, and therefore instantiated multiple times without the possibility to be rewritten. Since conventional databases do not offer simple solution for the tamper resistance [148], blockchain represents an adequate solution.

- *Q4: Which is the blockchain primary adoption?:* The proposed system principal goal is to use blockchain as a platform to support digital asset exchanges (where an asset is a computing resource) and related orchestration automatization.

  Q4.B) Which is the platform primary purpose? The goal of proposals is to leverage on blockchain to secure the orchestration interface by means of an abstraction making computing resources an asset, while the outcome is not the asset exchange itself rather the automatization of authentification that is related to its usage.

  Q4.B.ii) Is confidentiality required? Yes. Typically, one would assume it is required in privately operated cloud/network systems as in the common practice, hence full-permissioned blockchains seems to be a better fit as there is no need for a publicly available ledger.

4.3 VMOA

In this section we address a data-center configuration such that clusters of virtualization servers are managed by an orchestrator node, i.e., a logically disjoint node managing multiple virtualization servers in a same administrative cloud network domain. The VMM receives instructions from the orchestrator about how to create, destroy, modify, copy, migrate VM. The VMM generally consists of a software layer adapted to decouple the VMs from the host and dynamically allocate the computing resources to the different VMs as required. The orchestrator to virtualization server communication is typically secured using standard public-key systems, which however are not 100% safe. There is continuous pressure on cryptographic schemes that can become obsolete in front of very high computing power. More severe is the case when an attacker gets the public key of one of the end-points, exploiting various exploits being regularly discovered in operating systems. By hijacking this communication channel, it is possible to generate misleading orchestration instructions, to obtain, for instance, malicious creation, destruction, modification, copying, or migration of VMs. In the following, we present a trustful model aiming at going beyond legacy practices to secure the cloud orchestration channel. We refer to the orchestration command authentication function as Virtual Machine Orchestration Authenticator (VMOA). A VMOA node is a server that functionally has to be seen as an intermediate machine between an orchestration server and the virtualization server supporting the VM to be authenticated. We first present how to run a VMOA function as a centralized server, detailing the required signaling for such a case. Then, we show it can be reinterpreted under a blockchain system.

4.3.1 Centralized VMOA

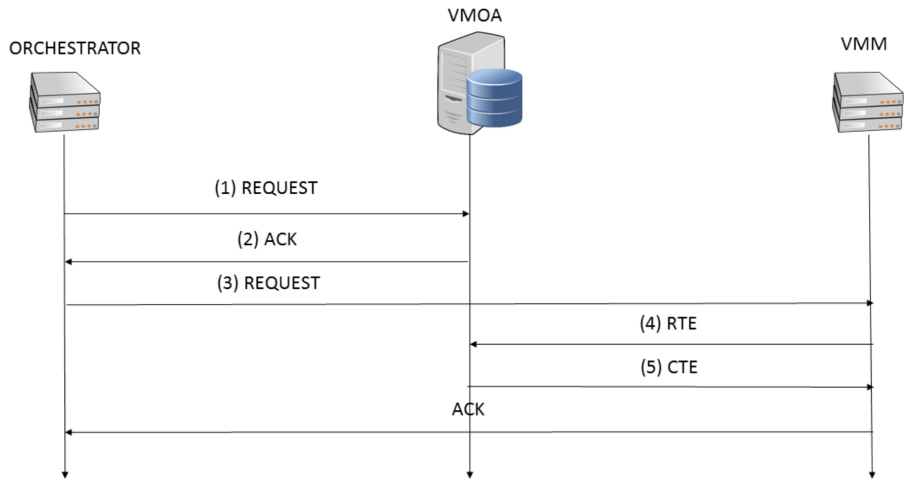

Figure 4.1: VMOA authentication protocol

Figure 4.1 illustrates the signaling message exchange between the orchestrator, VMOA and VMM nodes, for the case where the VMOA function is deployed on a single server. Often, authentication servers are realized as a single centralized entity, at least logically. In such a configuration, we can identify six main steps where authentication is run at both the orchestrator-VMOA and VMM-VMOA interfaces, as explained hereafter.

1) A cloud orchestrator issues a VM orchestration REQUEST instruction concerning, for instance, a VM create, copy, destroy, migrate or resize command, which is identified with its ID, the ID of the involved virtualization servers, the IDs of the involved VMs, and related assets, meant as the related amount of resource required by the VM for each involved network and computing resource.

2) The VMOA node verifies if the orchestrator has the authorization to issue orchestration commands, querying a local registry; if yes, the VMOA sends a positive ACK message to the VM orchestrator; if not, a negative acknowledgement is sent.

3) If the acknowledgement from the previous step is positive, the VM orchestrator sends the orchestration REQUEST command to the virtualization server concerned with the VM.

4) The VMM of the virtualization server, before actually running the command, queries VMOA sending a RTE (Ready to Execute) message in order to authenticate the orchestration command.

5) VMOA verifies whether the orchestration transaction is valid or not, by querying a local registry; if yes, it sends an acknowledgement via a CTE (Clear to Execute) message to the VMM; if not, the VMOA sends a negative acknowledgement. It is worth noting here that the way the validation is performed could rely on a distributed database system for the registry.

6) In case of positive acknowledgement at the previous step, the VMM authenticates the orchestration command request, and sends a positive acknowledgement (ACK) to the VM orchestrator. Otherwise, a negative acknowledgement can be propagated back to the VM orchestrator. If positive ACK, then the command is executed.

Centralized entities represent a single point of failure threat, that can disable nodes synchronization after system recovery. It is important to emphasize that there must be a trust in such centralized entities considering that it has the authority to change data and jeopardize its integrity and authenticity.

4.3.2 Blockchain VMOA

We propose in the following a way to run the VMOA function using a blockchain system shared between the virtualization server, the orchestrator and VMM agents. Indeed, the number of orchestrators and the number of virtualization servers may vary in time. The blockchain VMOA we envision is completely decentralized, with no hierarchy between orchestrators, and no hierarchy between virtualization servers. We refer in the following to blockchain VMOA as B-VMOA. Upon B-VMOA configuration, a VM orchestration command is issued to the virtualization server, which then by way of B-VMOA authenticates the orchestration command. In a case of negative confirmation, i.e., transaction rejection by VMOA peers, the orchestration command is not executed, while if it is appended, the command is carried out by the virtualization server VMM that in turn, if the command is successful, confirms the successful operation to the orchestrator by way of publishing it on B-VMOA. Figure 4.2 illustrates an abstract atomic view of the blockchain VMOA system. The system can be composed of multiple orchestrator servers, multiple host servers and multiple VMOA agents run locally on orchestrator and host servers, and on physically distinct servers. In a VMOA blockchain, the general steps are as follows. Orchestration request (create, destroy, resize, copy, migrate) issued by orchestrator to a virtualization server is done via corresponding transaction between corresponding peers on VMOA blockchain. As explained before, VMOA network consists of mutliple VMOA peers, in charge of an operational role to execute VMOA transactions and to distribute the transaction output to all network participants. Let us consider VMOA network in Fig. 4.2 with the corresponding transaction sequence diagram in Fig 4.3. We simplify the example using two VMOA peers. In reality, the size of the network may vary hence the lines initially used to help us understand transaction flow will become cumbersome.

1) Orchestrator sends a transaction proposal via B-VMOA client it run locally to the VMOA peers. This proposal represents specific orchestration request and consists of at least: the client id, payload, transaction id, and destination.

2) Each VMOA peer executes transaction against current ledger state on its local ledger. By invoking locally stored network policies, peer can verify sender authorizations, and produce the transaction output. It is important to notice that each VMOA peer must run transactions logic, i.e., a smart contract locally to be able to generate transaction read-write dataset by way of executing each transaction independently. Hence, B-VMOA peers have a functional role to endorse each transaction by way of verifying sender authorizations and calling smart contract functions.

3) Each B-VMOA peer signs the response in a form of <read,write> dataset and distribute it to all VMOA participants.

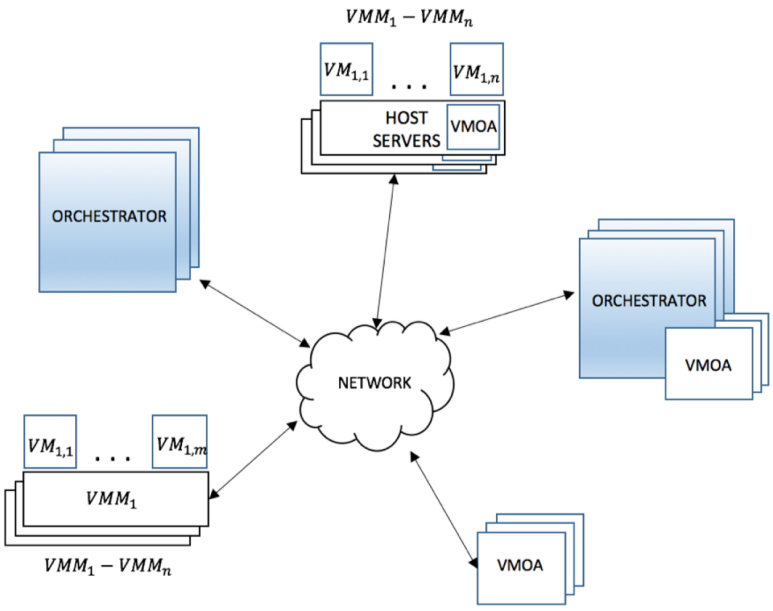

Figure 4.2: Distributed Virtual Machine Orchestrator Authenticator (VMOA).

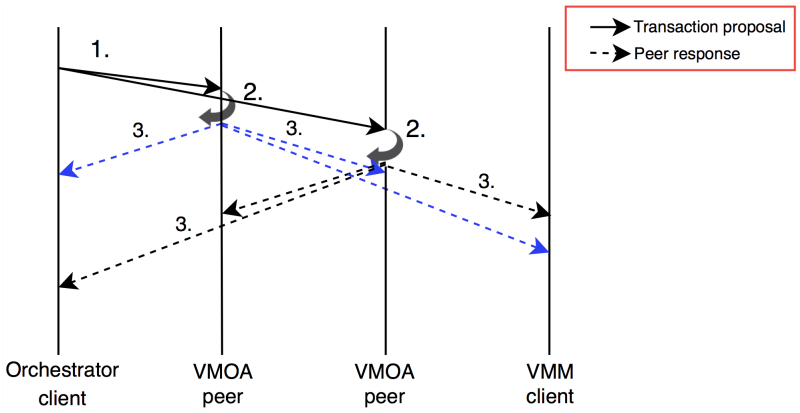

Figure 4.3: B-VMOA transaction sequence diagram.

Eventually, all VMOA nodes, including VMM node will receive transaction. Once it is validated, VMOA nodes will append new transaction to its blockchain ledger. When the transaction is to be considered valid directly depends on the following: (1) consensus, (2) p2p communication protocols and blockchain distributed architecture which can follow legacy sate-machine replication architecture or new advance architecture designs (see *execute-order-validate* architecture explained in previous chapter.) We later detail these steps once the proposed VMOA structure and transaction management is described. It is important to notice that there is no direct communication between cloud orchestrator and VMM as in the centralized example. Hence B-VMOA system design brings the following features:

- *Orchestration control plane decentralization*: By way of DLTs, B-VMOA enables P2P data sharing and storage without entrusting the ledger maintenance to any central authority. It does not mean completely cutting out intermediaries that validate transactions (*disintermediation*) like permissionless blockchains do, but rather decentralizing them along with their roles.
- *Immutability*: while legacy orchestration solutions allow data manipulation by a central authority, B-VMOA working with replicated information protect data from any sort of tampering and falsification; except in situations where the majority of the network's efforts are devoted to change the registry [31] (e.g, the Ethereum DAO fork [32]) or where the adversary thresholds are exceeded (see Appendix .2.4). Data immutability makes data accessible and manageable by different cloud entities that do not need to give the complete trust to orchestrator, but rather decentralize orchestration control plane.
- *Orchestrator - VMM communication integrity, authenticity and non-repudiation*: the data hashing grants that B-VMOA transactions are not modified during its transmission (i.e., integrity). Moreover, the origin of a transaction can be ascertained by the senders' public key dissemination, while the evidence of the sending action is represented by the data signing procedure involving the private key (i.e., authenticity and non-repudiation). Blockchain signing scheme combining asymmetries cryptography and data hashing is presented in Appendix .1.
- *Auditability*: All activities related to cloud orchestration in B-VMOA systems must be validated and verified thus, each activity should be visible to all B-VMOA participants in its entirety. In this way, we insure cloud orchestration traceability via audits. Indeed, recent implementations enable multiple ledger to be isolated and maintained within the same blockchain system via private channels. Nevertheless, ledgers data is visible to all channel participants, thus the auditability is satisfied at channel level.

The mix of the above features qualifies the conceived B-VMOA system at a quite high level of dependability, differentiating it from the legacy network orchestration solutions.

4.4 B-VMOA transaction management

We describe in this section how transactions are identified and managed in B-VMOA. A transaction is uniquely identified by the following information: issuer ID, command assets, receiver ID, VM ID and timestamp. The issuer could be any orchestrator in the network and the receiver could be any VMM. An orchestration alters the state of a VM (e.g., VM create, copy, destroy, migrate, resize). Orchestration command assets are the corresponding resource attributes (e.g., memory, processor, network resources). The VMOA function is operated with a set of VMOA agents that may be integrated to the orchestrator node, to the virtualization server node and/or to the independent server. The storage of the assets needs therefore to make use of a distributed database. In the case B-VMOA runs at the orchestrator node, steps 1-2 in Figure 4.1 are inter-process communications within the same orchestrator machine. In the case B-VMOA runs at the VMM node, steps 4-5 in Figure 2 are inter-process communications within the same virtualization server machine. In the case B-VMOA agents are integrated to the orchestrator nodes and to the virtualization server nodes, steps 3-6 in Figure 4.1 may be

omitted. The B-VMOA agent running locally informs the virtualization server that there is a command addressed to it. In a B-VMOA system, the database is implemented by a distributed ledger that is written and read by multiple distributed nodes concurrently. The way the distributed ledger is used supports decentralized validation of orchestration commands. The B-VMOA ledger consists of orchestration command transactions, a transaction being characterized by the transaction records already mentioned, and any other additional field that could reveal useful for the orchestration operation, such as for instance a blockchain transaction time-out. The ledger is stored at all B-VMOA peer nodes. The virtualization server, more precisely its VMM, is able to authenticate the orchestration command checking if there is a valid corresponding transaction stored in the VMOA ledger. An execution time limit can also be associated with the transaction. Orchestrators should be the only nodes able to issue commands. Every command corresponds to a particular transaction. Consequently, every action in a system (e.g., VM create, copy, move, resize) is associated with a transaction created by the orchestrator. Positive acknowledgement corresponds to command validation by the VMOA blockchain network. Alternatively, VMMs may autonomously recognize themselves in a new transaction in the VMOA blockchain ledger they run locally. Let us detail in detail how B-VMOA treats computing and network resources related to VM orchestration commands as blockchain assets. For each orchestration command, one or many transactions are executed with resource assets transfer from virtualization server to another virtualization server and/or to the orchestrator.

4.4.1 Block construction

Every orchestration transaction is available in all the B-VMOA ledger replicas, using a blockchain data structure. One orchestration transaction can involve multiple computing resource-level transactions, grouped in a same B-VMOA block. Starting a system for the first time, B-VMOA builds the genesis blocks, which is the first block in a system. All later transactions are stored in additional blocks chained together and building a chain of blocks (where blocks are linked by an identifier result of a hashing function on the block binary components). A 3-block VMOA blockchain example is illustrated in Figure 4.4. Each block contains timestamp, transactions, and the hash id of a previous block. B-VMOA nodes contain a smart contract that, as explained hereafter, is a code object to instruct operations to trigger upon block validation. All data in a block, including the hash id, is hashed and is used as link with the next block in a chain. This specific blockchain data structure ensures data security. In a blockchain network, a decentralized consensus protocol, involving all or a subset of the nodes, is needed to validate transactions. In B-VMOA, a private blockchain network is built. Nodes to be involved in the transaction validation can be the virtualization servers, the orchestrators, and/or dedicated hardware and/or dedicated virtual machines.

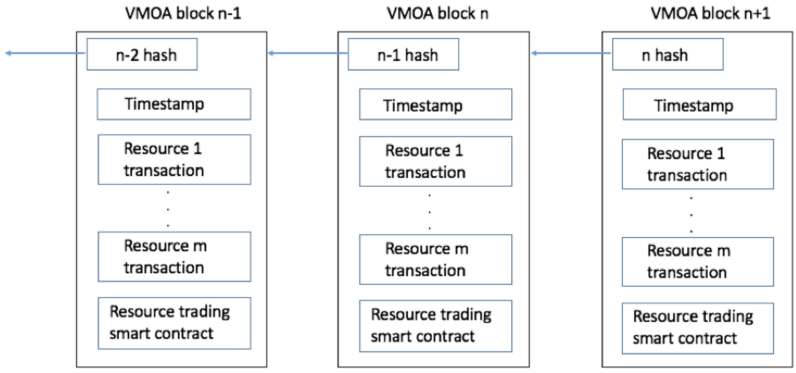

Figure 4.4: A 3-block VMOA blockchain structure example.

4.4.2 Blockchain operations

In B-VMOA, resources are treated as pseudo-currencies. For example, one currency for the number of CPUs, another one for the amount of RAM, another one for the access link capacity, another one for wide-area-network link resources, etc. As an example, suppose we have a single resource to manage and hence one single currency asset.

The data model of a B-VMOA transaction differs depending on the system implementation and its application. For instance, the Bitcoin protocol imposes the transfer of *Unspent Transaction Outputs* (UTXOs [55]), presented in Appendix 2.2.1.1. Post-Bitcoin data models have evolved in two different ways. First, blockchains moved to the adoption of an *account-based model*, making use of a completely new transaction syntax (Turing complete) [8] and resulting more 'smart contract friendly'; Ethereum is one of the so-called 'second generation' blockchains [56] adopting this record-keeping model. Subsequently, blockchains' intention was to maintain the original Bitcoin data-structure along with its improvement proposals [57] to which integrate the benefits of an account-based model. General blockchains, going beyond cryptocurrencies and digital assets, may adopt basic models supporting smart contract execution. Offering more and more general operations corresponds to a data model supporting more and more complex logic, hence overcoming both the account and the UTXO models. Blockchain-based systems of this type adopt a *key-value data model* (also called table-data model). We present in Appendix 2.2.1 these different models in more details along with its benefits and drawbacks. According to implemented data model, two asset management strategies are possible, at bootstrapping:

1) The digital asset corresponding to a server computing resource is owned by the given server, and orchestrators own 0 assets. This solution creates a micro trading system where brokers are different VMM and orchestrator clients, trading computing resources on B-VMOA network following the trading rules defined by way of smart contracts. This concept was design with respect to legacy *UTXO* and *account-based* model.

2) Recent blockchain generation adopt a *key-value data model*. The asset management strategy can offer more general operations and support for more complex transaction logic, hence overcoming

both the account and the UTXO models where B-VMOA peers register asset state as data-tuples that can be updated.

In the first case, at bootstrapping, a transaction corresponding to an operation freeing resources in a server (e.g., related to a VM destroy or migration orchestration command) is issued by the server VMM with as asset destination the orchestrator. As such an orchestration command is originally initiated by the orchestrator, and not by the VMM, there is the need to plan for a dual transaction from the orchestrator to the VMM, which triggers the actual transaction from the VMM to the orchestrator; this can be implemented by means of smart contract in the dual transaction block. Symmetrically, the same process linking the actual transaction to its dual transaction exists when computing resources have to be allocated to VMs: let us suppose that no resource is used at the virtualization server when an orchestration command requiring resource usage (e.g., VM creation); in order to perform such an operation, the VMM first must be triggered by an actual transaction from orchestrator, then followed by a dual transaction from the VMM.

4.4.3 The dual transaction abstraction

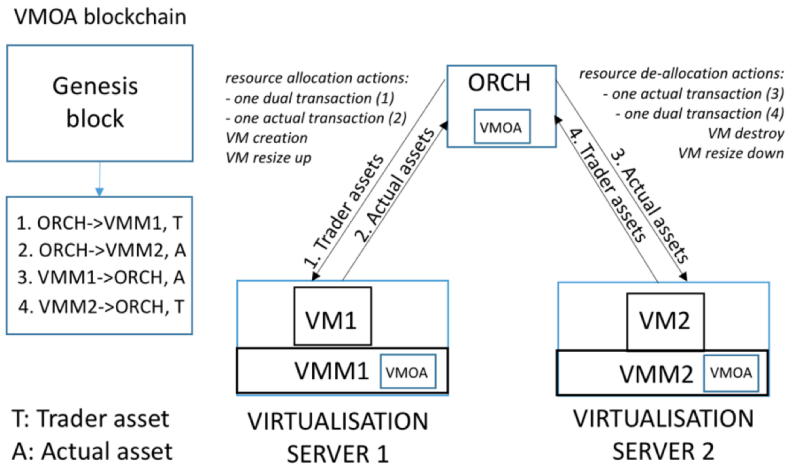

Figure 4.5: Representation of the transactions corresponding to allocation and deallocation orchestration commands.

Therefore, adopting the first strategy described above, we conceive an orchestration command as a trade of two assets: an actual one managed by an actual transaction and meant to represent the actual resource asset, and a dual artificial one managed by a dual transaction meant to represent a trader asset, where the dual transaction and the trader assets are artifacts needed to comply with transaction directionality constraints of blockchain system operations. Both assets (actual and trader ones) have the same absolute value, but each has a different functionality, i.e., the same amount of trader asset must be traded for a given amount of actual asset.
The process linking a dual transaction to an actual transaction is shown in Figure 4.5. The trader asset represents a pledge for the actual resource assets. In order to allocate actual resources (which can

correspond to a VM create or to a VM resize up orchestration command), i.e., to buy resource asset, as in the first two transactions illustrated in left part of Figure 4.5, the orchestrator must transfer the same amount of trader assets as a pledge to the virtualization server VMM where resources are instantiated, i.e., by whom resource assets are sold. Symmetrically, when a VMM frees occupied resources (which can correspond to a VM destroy or a VM resize down orchestration command), a trade of assets is performed in the opposite direction, i.e., the orchestrator buys trader assets back, while selling the same amount of actual resources to the VMM. The corresponding transactions are in the right part of Figure 4.5 (the number associated to transactions in the figure are there only to stress the relative order of transactions, i.e., transaction 3 does not need to be executed after transaction 2, but it is executed before transaction 4). For servers, the number of owned trader assets represents the amount of allocated computing resources and for an orchestrator, the amount of owned trader assets represents how many free resources it can allocate. In contrary, for an orchestrator, the amount of owned actual assets represents the amount of computing resources used by running VMs, and for servers, the amount of owned actual assets represents its available computing resources.
Orchestration command assets are the corresponding virtualization computing resources, e.g., at least memory, processor, and network resources. For simplicity, let us consider only one attribute, e.g., memory. The total number of trader assets in a system is computed as a total number of virtualization server memory. In the genesis block, one of the transactions should have an equivalent amount of trader assets assigned to the corresponding orchestrator.
The total number of actual assets per VMM corresponds to the real memory owned by each server. In the genesis block, one of the transactions should have an equivalent amount of actual assets dedicated to the corresponding VMM. The same applies to any computing resource involved in a virtualization stack. As a given orchestration action is expected to act on more than one computing resource, a VMOA block can contain as many transaction entries as computing resources involved.
Execution of commands directly corresponds to the different orchestration transaction entries, which obviously means that assets are redistributed after every orchestration command. Additionally, as transactions do not include any fees, the total number of assets must be the same in every moment. For each transaction to be considered as valid, consensus must be reached.

4.5 Numerical example

Let us describe an example with assets variation after that orchestration commands are executed in this order: (i) VM creation, (ii) VM resize, (iii) VM migration, (iv) VM destroy, in a network with one orchestrator and two servers. For simplicity, let us consider the case where 1 asset corresponds to 1 unit of resource, be it RAM, for instance. Table 4.1 illustrates assets distribution after every command.

- Step 0 - the first row shows assets distribution before any VM is created. The global difference between trader assets and actual assets is null. This is illustrated as a first, genesis block on the Figure 4.6.

- Step 1 - the second row corresponds to assets variation after VM creation command. The command creates one VM on the server, allocating 64 GB to it. The dual and actual transactions (second block in Figure 4.6) correspond- ing to these orchestration commands reallocate the

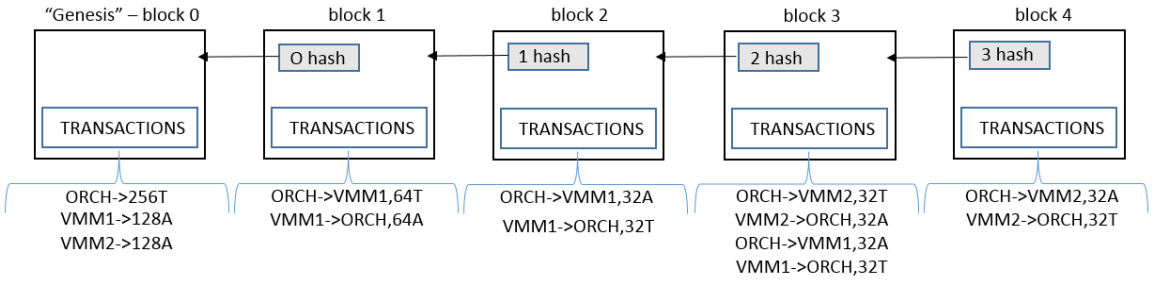

Figure 4.6: A 5-block VMOA blockchain example. 'A'indicates actual resources, while 'T 'indicates trader dual resources.

total system assets.

- Step 2 - the third row shows the assets distribution after the orchestrator resizes down VM on VMM1 by 32 GB. The corresponding transactions increase the total amount of Trader assets owned by the orchestrator, but it decreases its number of Actual assets for the same amount. On the other hand, VMM1 will gain the same amount of real assets and give away the corresponding number of Trader assets. Note that for the moment the distribution of assets owned by VMM2 remains the same (third block in Figure 4.6).
- Step 3 - the fourth row shows the assets variation after VM migration from VMM1 to VMM2. It is worth noting that in this peculiar case there should be two transactions between the orchestrator and the source server, and two transactions between the orchestrator and the new server. It corresponds in fact to a VM destroy in the left server and a VM creation to the right server (the fourth block on Figure 4.6).
- Step 4 - the fifth row shows the assets variation after the orchestrator issues a VM1 destroy command (by transferring 32 Actual assets) to VMM2 and VMM2 has freed resources (by trading back 32 Trader assets). This is illustrated as a final block in Figure 4.6.

This simple example is referred to the RAM resource, and a completely independent equivalent process could be run for the CPU resource, the access network link bandwidth related to the operation of VMs, the wide-area-network link bandwidth in case of geographically distributed virtualized network function overlays, etc. For instance, each VM can be instantiated with an amount of virtualization server access link bandwidth consumption. Indeed, the bottleneck is typically supposed to be the access link in cloud/NFV systems. Following the same scenario as in Table 4.1, the trader assets can represent a pledge for the CPU or network bandwidth resource.

Table 4.1: Memory resource asset management with B-VMOA – example.

	Orchestrator		VMM1		VMM2	
Assets owned	*Actual*	Trader	*Actual*	Trader	*Actual*	Trader
Step 0	0	256	128	0	128	0
Step 1 (VM creation)	64	192	64	64	128	0
Step 2 (VM resize)	32	224	96	32	128	0
Step 3 (VM migration)	32	224	128	0	96	32
Step 4 (VM destroy)	256	0	128	0	128	0

Future research work should focus on validating the model for a large number of B-VMOA peers. Since the community had move on more advance aforementioned data model, for our Proof-of-Concept we were able to adopt transaction management strategy which offers more general operations and support for more complex transaction logic, hence overcoming both the account and the UTXO models where B-VMOA peers register asset state as *key-values* pairs. Object oriented languages for smart contract adopted by the last platform generation allows us to keep the same B-VMOA functionality while facilitating the transaction management strategy.

4.6 VMOA Proof-of-Concept design

Let us further develop the vademecum logic applied to cloud orchestration and how it can lead to precise platform specifications. According to the HOW vademecum guidelines, the reader can put aside some platforms, ending up with one or more choices as preferable platforms based on the Fig. 3.3. Indeed, a chosen platform or set of platforms can overcome some of the limitations specific to another one, or may serve as reference and guideline to develop its own framework. We chose to design and develop B-VMOA POC leveraging Hyperledger Fabric platform as it is compliant with B-VMOA specifications we examined so far.

4.6.1 B-VMOA design on Hyperledger Fabric

Unlike frameworks that can be installed and used out-of-the box, Fabric rather provides a set of infrastructure and building blocks. Hence, at first we have to analyze specific blockchain based application requirements before we design the system infrastructure involving different HLF building blocks. Furthermore, in order to be able to develop B-VMOA functionality, we have to be familiar with the standard Fabric components, APIs, and its different functionality. At the moment of writing, HLF is the only platform that deviate from a traditional state-machine replication approach. In all others smart-contract based blockchain platforms, the consensus protocol is designed in such a way that all

transactions are ordered at first and distributed to all peers which will execute them sequentially. This so called *order-execute* architecture has several limitations as follows: (i) Hard-coded consensus; (ii) transaction execution on all peers; (iii) non-deterministic smart contract may take down entire network. The new architecture imposed by HLF separates main B-VMOA network functions which can be performed on separate peers by way of: (i) transactions execution on a subset of peer nodes, (ii) ordering service with modular consensus protocol, and (iii) transactions validation with respect to validation system policies. This results in better scalability, modular consensus module and better network performances.

4.6.1.1 Key concept

Let us detail system key concepts with respect to our B-VMOA design:

- B-VMOA assets; by way of asset we present the core value of a cloud data center, i.e., virtualisation servers that can allocate computing resources to VMs. We design asset model so that each server presents a object of value with different attributes that changes over the time in state database. This attributes are real computing resources like RAM, CPU etc.
- Chaincode; these state changes occur only via transactions coded in smart contract (SC) known as *chaincode*. Thus, chaincode defines the structure of the data center server (B-VMOA asset), along with attributes manipulation logic that can be executed against the asset state.
- Distributed ledger; ledger is a data structure that keeps track of all transactions. Furthermore, it records the asset state changes as a result of executed transactions on the network. Distributed here means that all B-VMOA peer nodes have its ledger replica.
- Membership service provider; let us describe in a high level how the identities are created and managed on B-VMOA network by way of membership service provider component (MSP) provided by HLF. A blockchain system has the process with known entities. In the context of HLF, a member refers to a separate or independent entity, managed by way of X509 certificates. When a participant identity is created the certificate is issued to the participant. Anytime a transaction is initiated by the participant, certificates private keys are used for signing the transaction, thus any component in the network can validate authenticity of the transaction using the participant public key. Besides the participants, the infrastructure components are assigned with an identity by way of certificates to prevent a scenario where malicious server is added to the network to manipulate the transactions. Certificates follow the typical process of issuance and the revocation by the certification authorities in the network. Identities are managed within the 'trust domain' so that the dependency on a single centralized certification authority is removed. As such, B-VMOA system have one MSP component per server as we defined trust domain on the server level.

4.6.1.2 B-VMOA Nodes

In general terms, a blockchain is an immutable transaction ledger, maintained within a distributed network of nodes. These nodes communicate with each other to maintain a copy of the ledger by

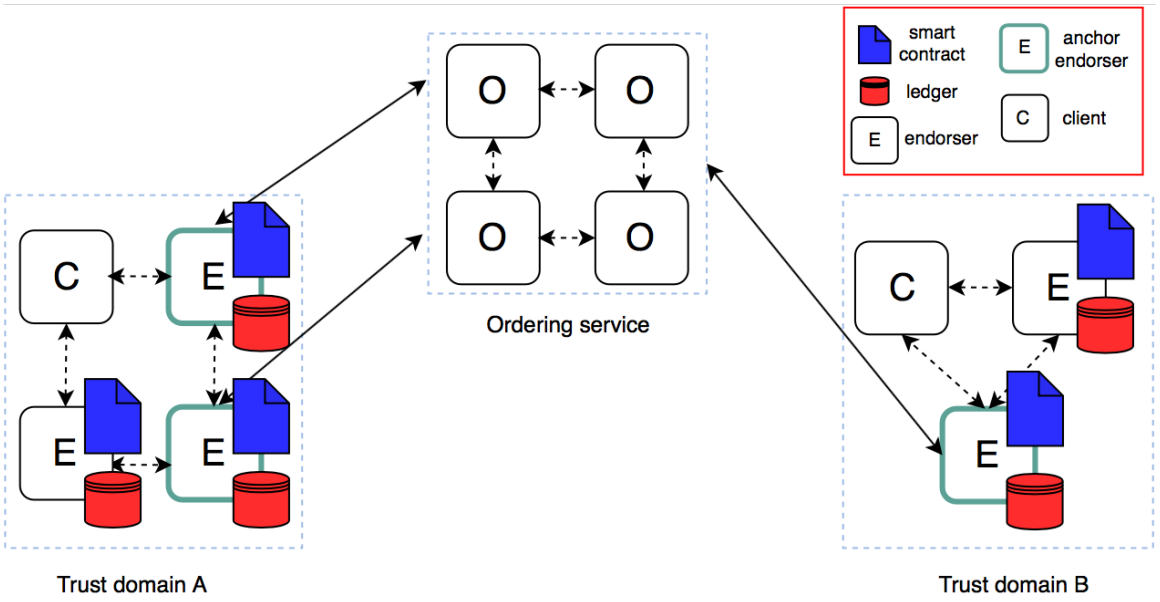

Figure 4.7: Hyperledger Fabric network example with two 'trust domains', A and B, and ordering service. Trust domain A consists of: (i) three endorsing peers where two of them are configured as anchor peer and (ii) one client peer. Trust domain B consists of: (i) two endorsing peers where one is configured as anchor peer and (ii) one client peer. Each endorsing peer is also a committing peer, hence has ledger replica. To be able to execute transactions independently, each endorsing peer has chaincode locally installed.

applying transactions that have been validated by a consensus protocol. As multiple nodes may run on the same hardware, they can be seen as a virtual entity, deployed on a physical machine, a virtual machine or a container. There are three different type of B-VMOA nodes as illustrated in Fig 4.7

1. B-VMOA Clients; a client acts as an intermediate between peers and ordering service. Client initiates the transaction proposal to peers, collect endorsed signatures, verifies if the transaction satisfy *endorsement policy* and than sends endorsed transaction to the ordering service. Endorsement policies are used by the client, the peer and the orderer to ensure that the transactions are valid before they get added to the ledger. Client knows who are endorsers from *endorsement policy* associated with a SC. The endorsement policy has two part: (i) a list of endorsers and (ii) endorsement criteria for a valid transaction (either the number of endorsements needed for checking the validity of the transaction or criteria specified as a percentage.) An example of endorsement policy for the network from Fig 4.7 may state that at least one peer from each organization must endorse the policy to have the transaction considered as valid. One may use logical expression with AND and OR operators to define expressions used for checking the validity of the transaction.

2. B-VMOA Peers; In a HLF, by default all peers are *committers*, thereby maintain the ledger. They receive ordered transactions within the blocks from ordering service, validates those transaction and commit state changes on the local ledger replica. Subset of the peers are in charge of

transaction execution, thereby called *endorsers*. They take up an additional role to simulate the transactions with respect to transaction logic defined in smart contracts (SC). Each transaction trigger SC that is locally instantiated on each *endorsing* peer. The endorser then rejects or accepts the transaction after it has carried out multiple validation check. Endorser then append its signature to the simulation result and send it back to the client. The SC is executed but the state of the SC is not updated in the ledger. The primary objective of this endorsement mechanism is to protect the network from intentional as well as unintentional attacks.

Within one 'trust domain' or 'organization' multiple peers may be deployed. Among them at least one has to be configured as *anchor* peer, responsible for block propagation to other peers within its organization via gossip protocol. To avoid single point of failure an organization can create a cluster of anchor peers. Hence, the anchor peer receives blocks from ordering service. Anchor peers are set up and defined as part of the channel configuration and the anchor peers are by default discoverable. It is important to notice that SC are deployed only on subset of peers called endorsers, while all network peers participate in ledger maintenance.

3. Orderers; Ordering service consist of a group of B-VMOA nodes responsible to establish consensus on the order of transactions and then broadcast them to peer nodes in the form of a block. Ordering service is provided by HLF and is explained in details hereafter.

4.6.1.3 Ledger

All peers in the network are *committing* peers, e.t., they have a ledger replica. Ledger is composed of: (i) transaction log and (ii) state database. Transaction log keeps track of all the transactions invoked against the current assets state, while the state data is the current state of the asset at any point in time.

By execution of SM, a transaction is created in the transaction log. With respect to the code in SM, they may be a change in the state data. Furthermore, the transaction log contains all transactions in the network whether they are validated or not, while state data changes upon valid and committed transaction. The transaction log is implemented using the levelDB, a lightweight library for building *key-value* data store [1]. As a fixed part of Fabric peer implementation, levelDB is part of the peer process. Furthermore, the state data consists of the versioned *key-value* pairs. It stores tuples in the form *(key, value, version)* where the current asset state is identified by the name of the key, and the value is represented by the way of arbitrary blobs or binary objects. Every time state gets to be updated a new version is created for the key value pair hence preserving the data loss. Both, transaction log and state database by default use levelDB which indeed supports such simple queries. To support more complex queries, the state data base is pluggable at the peer level. Beside levelDB, HLF enables a client-server model database Apache CouchDB [2]. The ledger forces a chain of ordered transactions hashes. Our B-VMOA system leverages Fabric to differs from other legacy blockchain systems in a way that block may contain a transaction which may not be valid. Thus, the validity of each transaction is addressed by the way of a bit mask, maintained in each ledger replica.

[1] levelDB - https://github.com/google/leveldb.
[2] CouchDB - http://couchdb.apache.org.

Furthermore, two types of transaction can happen on the network: (i) Invoke transaction which trigger specified smart contract functions thus may result not only in data-state reading but also in data-state change, while (ii) Query transaction executes specific queries which returns the current ledger state, hence no impact on the ledger state.

4.6.1.4 Transaction Flow

In the following we outline the 3-phase HLF transaction flow process as shown in Fig 4.8. We stress that B-VMOA design follows *execute-order-validate* architecture imposed by HLF.

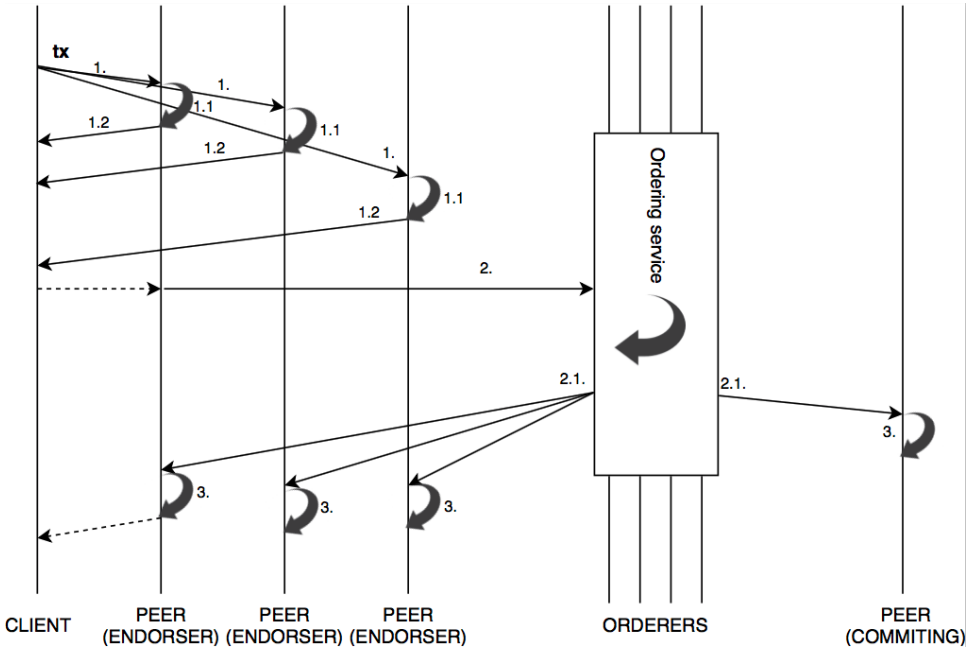

Figure 4.8: Hyperledger Fabric Transaction Flow.

First Phase: Transaction proposal and execution

1. To invoke a transaction, the client sends a PROPOSE message to a set of endorsing peers. The set of endorsing peers is made available to client via its peer, which knows the set of endorsing peers from endorsement policy.

1.1. Each of these endorsing peers then independently executes a chaincode to simulate the transaction using the transaction proposal from client as the inputs against the local *key-value* state. Further, each endorser generates a transaction proposal response: (i) *read-set* which represents the version of keys read by chaincode to be sure that all peers will perform modification on the same data state and (ii) *write-state* which represents a new generated key-value pairs as a result of a transaction simulation. At this moment, peers do not update their ledgers. All chaincodes are

isolated from peers within a container. Endorsers rather simply sings it and prepare for sending back the response.

1.2 All endorsers send to the client endorsement message: <*read-set, write-state, metadata, signature*> signed by its cryptographic signature. The first phase is complete once the client has received a sufficient [3] number of endorsement messages. It is important to notice that different peers can return different responses for the same transaction proposal considering that responses are generated at the different time, on peers with different ledger state, or due to the chaincode non-determinism. The first can be simply bypassed with the new request from client for a more up-to-date proposal response, while the later will result in the transaction rejection.

Second Phase: Transactions ordering

2. For transaction with a valid endorsement, the submitting client invokes ordering service using the broadcast service. If the client does not have capability of invoking ordering service directly, it may proxy its broadcast through some peer of its choice.

2.1 The ordering nodes receive transactions containing endorsed transaction proposal from different blockchain clients. They order them into blocks and wait until the block reached the block-size limit [4] or block timeout [5]. Sequence of transactions which are encapsulated in a block can differentiate from their arrival sequence, due to transaction propagation. Nevertheless, ordering nodes package all the transactions upon their arrival by dint of transaction timestamp without inspecting the transaction's content. They do not have a ledger replica, nor chaincode, hence they are responsible for distributing a strictly ordered blocks to the rest of the network (both, endorsing and committing peers).

Third Phase: Validation and commit

3. The final phase of the transaction work-flow involves subsequent validation of blocks after which they can be committed to the ledger. Specifically, at each peer, every transaction is validated to ensure that it has been consistently endorsed by all relevant endorsers before it is applied to the ledger. Only for valid transactions a peer performs multi-version concurrency control, i.e., serially verifies if there is a match between endorsement *read-set* and its current *key-value* state. Bitmask, stored within the block, indicates validity of each transaction. Failed transactions are retained for audit, but will not change data-state. At the end all 'valid' *write-sets* get to update local *key-value* states. B-VMOA submitting agent receive a notification about the result of transaction proposal.

It is important to stress that blockchain brings *consensus* on a whole different level. The entire aforementioned transaction workflow process is what the system consensus constitutes within B-VMOA system. Compared to legacy distributed system, all peers have to reach agreement not only on the order and content of transactions, but additionally on a way the state changes occur and on asset structure, in a process that is mediated by orderers. We discuss orderers in more detail in following section.

[3] This is literally what means to achieve consensus — every trust domain who matters must endorse the proposed ledger change before it will be accepted into any peer's ledger.

[4] BatchSize: predefined number of transaction per block

[5] BatchTimeout: maximum time to wait before send one block

4.6.1.5 Ordering service

Let us detail the concept of ordering, how orderers interact with peers and they role in the transaction flow. Furthermore, we detail currently available implementations of the ordering service. As a modular component, ordering service is logically decoupled from the network peer (see Fig 4.7). Hence, the blockchain platform can be tailored with respect to the application requirements.
Ordering nodes arbitrate interaction between B-VMOA client agents and peer nodes, hence they are crucial players in the transaction flow process. Permissionless blockchain systems allow any node to participate in a consensus thus such a systems must rely on **probabilistic** consensus algorithms which eventually guarantee ledger consistency to a high degree of probability. Nevertheless, probabilistic approach is yet subjected to phenomena also known as a ledger "fork", where different participants in the network have a different view of the ledger. Permissioned setting allows B-VMOA system to bypass this vulnerability, relying on **deterministic** consensus algorithms. Beside ordering role, the ordering service maintains the list of 'trust domains' that participate in a consensus. There can be one peer per 'trust domain' or many whereby at least one has to be anchor peer. Every node that interacts with blokchain network, including ordering peers requires their identity from digital certificate managed by peer MSP. Orderers do not participate in the first phase of transaction flow as shown in Fig. 4.8, but rather starts to play they role in the second phase. Orderers receive endorsed transactions from different B-VMOA agents simultaneously. BatchSize and BatchTimeout are configuration parameters related to the size and maximum elapsed duration for one block. Furthermore, blocks are distributed to all connected peers. The ordering service delivers blocks to the anchor peers using gossip protocol. In case of an outage, a peer can receive the blocks after connecting to an ordering service node, or by gossiping with another peer. All peers use this strict order of transaction. The same transaction can not be embedded into different blocks that compete to be accepted like in probabilistic-based blockchains. Hence, here *finality* means no ledger forks. It is important to notice that orderers neither execute nor validate transactions. They are in charge for providing message consistency and delivery guarantee. In fact, they are responsible for the 'order' phase in *execute-order-validate* architectures. At the moment of writing, HLF has support for (i) Solo and two CFT (crash fault-tolerant) ordering service implementations: (ii) the etcd library of the Raft protocol [6], and (iii) Kafka [7] (which uses Zookeeper internally). Let us detail some implementation aspects.

Solo
Solo orderer is a messaging middleware run as a single node without no clustering feature in it. As such it centralizes the message delivery and is not fault tolerant. Solo simplifies Fabric deployment and network set-up, thereby is convenient for system design testing and Proof-of-Concept. Nevertheless, CFT ordering service is intended to be used in production.

Kafka
Kafka is a messaging middleware that has high throughput and high scalability features, enabling crash fault-tolerant ordering by way of clustering. Apache Kafka relies on a "leader and follower" node configuration using a ZooKeeper ensemble for management purposes. It is available within HLF since v1.0. release.

[6] etcd - https://raft.github.io.
[7] Kafka - https://kafka.apache.org.

Raft
The latest implementation as of v1.4.1 release is Raft ordering service, a crash fault tolerant (CFT) ordering service based on an implementation of Raft protocol in etcd. Even though Raft follows the same leader design as Kafka, the major differences between them are as follows:
(i) Kafka and Zookeper are design to be run in a small group of host. Hence, even if the orderers are run by different 'trust domains', centralization is questionable since all the nodes have to belong to the same cluster that is controlled by single trust domain. Unlike Kafka, Raft enables different 'trust domains' to run their ordering nodes, hence completely decentralized ordering service.
(ii) Unlike Kafka, Raft is embedded into the ordering peer binaries. This facilitates set up process and improve system performance.

4.6.1.6 Membership Service Provider

The Membership Services Provider (MSP) provides the credentials to the nodes to participate in the blockchain network. The default MPS comes with Fabric binary, but alternate implementation of MSP may be plugged in without impacting the foundation components. The default MSP implementation is based on the public key infrastructure (PKI). Two main services are (i) authentication service and (ii) authorization service managed by way of certificates, hence all the certificates are issued, validated and revoked for every B-VMOA node. Certification Authority is explained in more details in appendix **??**.

4.6.2 B-VMOA model

Let us present B-VMOA model in more details. We conceived B-VMOA system design as follows: (i) multiple peers run B-VMOA chaincode along with endorsing policy, and (ii) multiple ordering nodes carry on massage ordering service. B-VMOA consists of at least three client nodes: one orchestrator client (OC) and two VMM clients (VMMC) responsible to initiate transactions on B-VMOA network. Nevertheless, we argue that the number of B-VMOA nodes depends also on endorsement policy. To have fully decentralized system, number of B-VMOA nodes should be the same as number of trust domains, hence at least one B-VMOA node per virtualisation server. VMM clients (VMMC) are hosted on different virtualisation servers within data center. Furthermore, orchestrator client (OC) can be run on any of two servers or on an independent server. In a real data center network, B-VMOA consists of multiple servers hosting multiple OC,VMMC and ordering nodes as illustrated on Fig 4.9. We stress that beside those virtual nodes, servers also run VM hypervisor or orchestrator.

Namely, B-VMOA system limits *trust domain* at the server level. Each server is considered as an independent trust domain, hence each server should run (i) one client node (to be able to propose transaction and receive events from the B-VMOA network), (ii) at least one peer node (to be able to participate in endorsement and ordering process). Furthermore, it may run one or more ordering nodes to participate in a consensus on finite sequence of transactions. In this way, we achieve decentralized and distributed B-VMOA system, where each server is able to validate and verify every orchestration command.

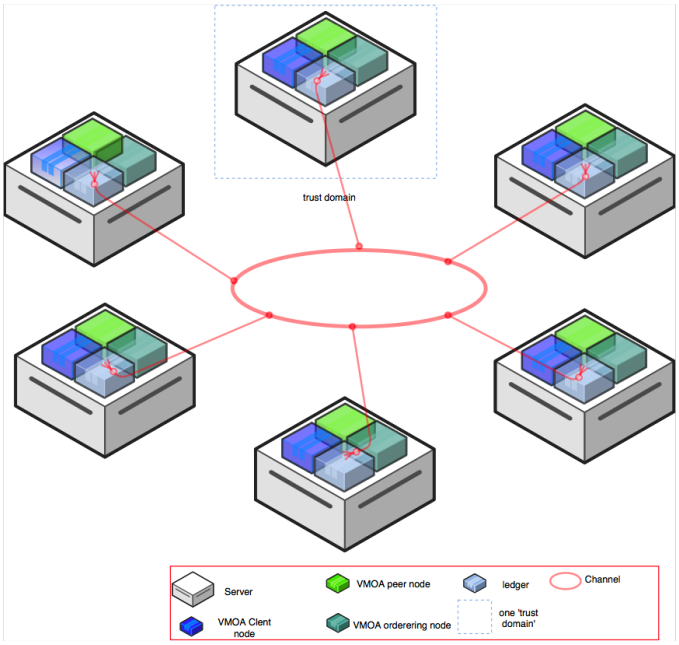

Figure 4.9: B-VMOA nodes.

Besides aforementioned advantages imposed by B-VMOA system, the most important originality is as follows. Orchestration management is no longer entrust to cloud orchestrator node but rather to a cluster of servers which additionally must agree on a way the asset state changes occur in a distributed process that is mediated by orderers. Indeed, legacy orchestration solutions may run physically distributed orchestrators nodes, nevertheless, term *centralized* in this context refers to vertically separate orchestration management from virtualisation servers only used to host VMs. B-VMOA breaks this separation by way of delegating to multiple stakeholders (virtualisation server) voting rights along with dispatching orchestration logic to server which is no longer just used as hosting machine.

4.7 Conclusion

In this chapter, we present our research work toward the design and implementation of a private blockchain system for managing the authentication of virtual machine orchestration commands in cloud computing and network function virtualization systems. We refer to it as B-VMOA (Bloclchain Virtual Machine Orchestrator Authenticator). We specified the basic primitives that are needed as well as conceptual boundaries. Our work targets the B-VMOA proof-of-concept implementation and system design to verify its practical potential. We design B-VMOA model using HLF as underlying technology. We conceived B-VMOA to break vertical separation between orchestration management and virtualization servers only used to host VMs. B-VMOA breaks this separation by way of delegating to multiple stakeholders (virtualisation server) voting rights along with dispatching orchestration

logic to servers which are no longer used just as hosting machines. By way of B-VMOA, we create a whole new ecosystem run within private data centers to secure virtual machine orchestration. Future research work should focus on exploring all B-VMOA potentials. For example, B-VMOA supports ledger and chaincode isolation between different participants by way of channels, hence could be used to create new ecosystem not only within a single private data center, but rather between multiple cloud providers. This could enable VM auditing and secure migration between multiple cloud stakeholders while creating a more competitive and user-oriented cloud market.

Chapter 5

Empirical Analysis for B-VMOA

Summary

In this chapter, we analyze B-VMOA performance as a function of various configurable parameters. We deploy B-VMOA using Hyperledger Fabric (HLF) platform in private cloud environment and collect data to validate our model. The HLF comprises of various components such as smart contracts, endorsers, commiters, validators, and orderers. As the performance of blockchain platform is a major concern for B-VMOA application, in this work we present an empirical study to characterize system performance and identify potential performance bottlenecks. Furthermore, we present the tools we used, network setup and discussion on empirical observations from the data collection.

5.1 Performance Metrics

Since blockchain is quite a new paradigm, it is essential to evaluate the performance of a system to ensure that it reaches desired capabilities. *Order-execute* blockchain-like systems are easier to evaluate since all transactions are ordered and executed on every peer. Complex transaction work-flow makes it more challenging to define the start and the end of the transaction. HLF constitutes various phases in processing a transaction such as endorsement phase, ordering phase, validation and commit phase.

Different phases scale differently due to various configurable parameters such as block size, endorsement policy, channels, state database. Hence, the main concern is to optimize network efficiency by finding the right set of these parameters. In this section, we present different B-VMOA performance metrics that may be applicable across different blockchain network classes. Due to our B-VMOA system design, this system-level performance metrics directly impact B-VMOA performance.
Let us detail different metrics we aim to evaluate:

1. *Transaction Latency* is the time elapsed between transaction request and when the transaction is confirmed by submitting client. It is the amount of all delays during the transaction flow due to the consensus algorithm. Some of the most prominent platforms examined in Section 3.4.2 are public systems, hence they rely on the probabilistic finality. Since these networks run in an open-access environment with anonymous miners, a transaction is considered final once enough [1] blocks have been appended to the chain. On the other hand, a voting-based consensus has immediate finality thus the state is guaranteed to be irrevocable. In our experiments, we consider a transaction complete when the client receives an event notification from its peer. In B-VMOA - HLF based system, we measure the latency by each transaction phase (see Section 4.6.1.4).

2. *Transaction Throughput*, expressed in transactions per second (TPS), represents the rate of appended transactions in a certain period. Only valid transactions should be considered. This is the measure across the whole B-VMOA network, thus the number (or %) of peers across it is observed also should be reported. Transaction throughput may be calculated as follows:

$$TransactionThroughput = \frac{TotalCommittedTransaction}{TotalTime}$$

   at % of observed peers.

5.2 Performance Evaluation Setup

Fig. 5.1 illustrates the performance evaluation setup. As we can see, it consists of two parts as following: (i) 'Network under Test' or 'System Under Test' (SUT), i.e. a collection of B-VMOA peers running the network and maintaining the ledger and (ii) The 'Test harness', i.e. the cluster of nodes in charge of performance evaluation. These nodes either generate workload on behalf of B-VMOA clients or analyze collected datasets to estimate performance metrics. Interface between Test Harness and SUT could be REST interface or more complex SDK.

5.3 B-VMOA system under test

In Chapter 4, we have demonstrated B-VMOA in a development environment. This section details our real cloud distributed environment setup.

[1] For Bitcoin enough means six, and for Ethereum twelve.

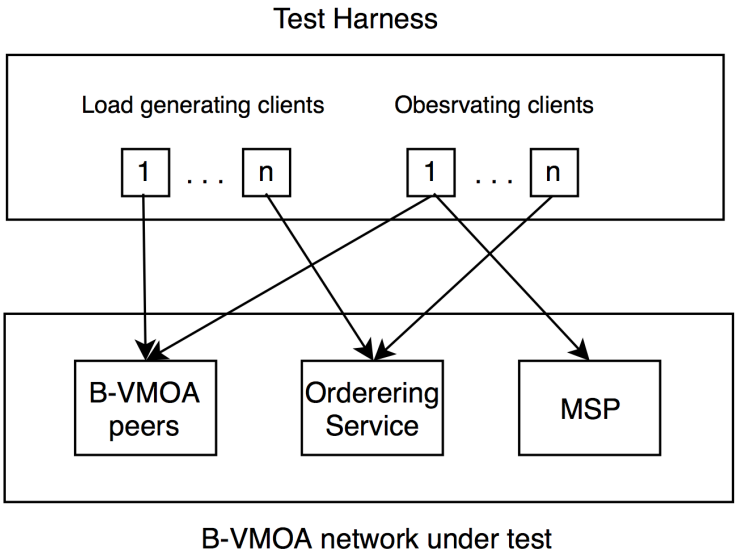

Figure 5.1: Performance evaluation setup.

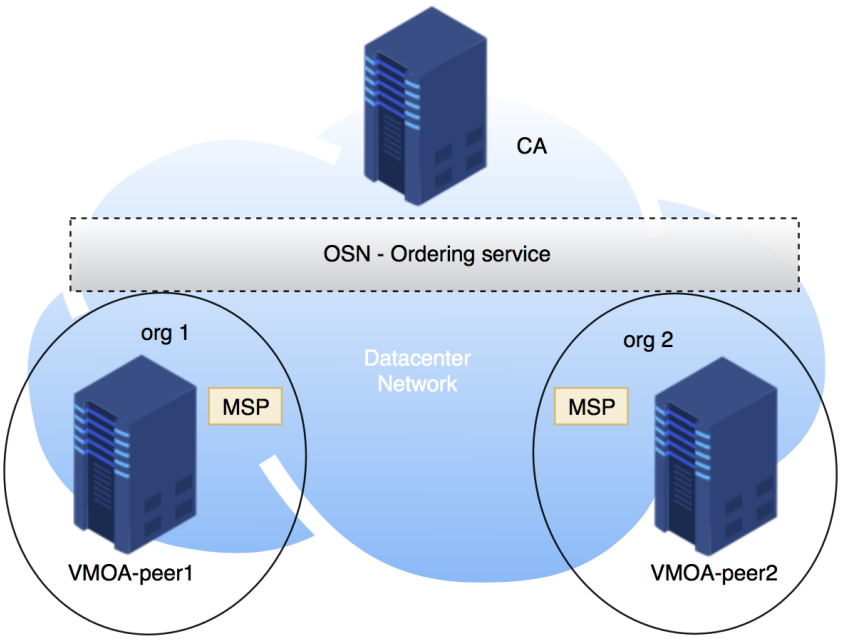

Figure 5.2: Experimental setup.

Fig. 5.2 shows B-VMOA experimental setup used in our benchmark experiments, i.e. a B-VMOA system under test. A consortium was setup with two trust domains representing two physical servers running B-VMOA agents. Hence, we run two HLF peers on each server (VMOA-peer1 and VMOA-

peer2). The ordering service was run on a separate node as a third neutral entity. To successfully commit on the B-VMOA network, the endorsement policy is set to include the signatures from at least one peer per trust domain, VMOA-peer1 or VMOA-peer2. To have a completely operative B-VMOA system, we have deployed CA server as a separate entity. We established communication by way of a single channel between the B-VMOA peers and Ordering service and instantiated a chaincode on it. Orderer was run in a Solo mode. All experimental configuration is summarized in Table 5.1.

Parameters	Values
Number of trust domain	2
Peer per domain	1
Block size	40 transaction per block
Block Timeout	$500ms$
Endorsement policy	peer1 OR peer2
Peer resources	(vCPU,GB) (1,1)(2,4)(4,16)(8,32)
StateDB Database	GoLevelDB
Ordering service	Solo

Table 5.1: Experimental configuration unless stated otherwise.

5.3.1 Private Cloud Setup

Fig. 5.3 shows our cloud setup. All nodes are run on a virtual private cloud (VPC). We used Amazon Virtual Private Cloud. Virtual private network closely resembles a traditional network within a data center, with the benefits of using the scalable infrastructure of AWS. A virtual private cloud (VPC) is logically isolated from other virtual networks in the same Cloud. We dedicated one VPC to our B-VMOA setup. Within VPC we run four VM instances that are dedicated to B-VMOA nodes: (i) Orderer, (ii) VMOA-peer1, (iii) VMOA-peer2 and (iv) CA server. Furthermore, we specified an IP address range for the VPC, added two subnets, associated security groups, and configured route tables. For external communication we used public subnet and a private subnet for B-VMOA nodes. B-VMOA peers were run on four machines, with resources allocated as follows: (i) 1 vCpu and 1 GB RAM, (ii) 2 vCpu and 4 GB RAM, (iii) 4 vCpu and 16 GB RAM and (iv) 8 vCpu and 32 GB RAM. Furthermore, to allow B-VMOA nodes to initiate outbound connections to external networks, but prevent unsolicited inbound connections, we used a network address translation (NAT) device for IPv4 traffic. NAT maps multiple private IPv4 addresses to a single public IPv4 address. A NAT device has fixed public IP address. It is connected to an external gateway and routes traffic from the nodes to external gateway, and routes any responses to the node. All nodes had the Ubuntu 14.04 LTS operating system.

In a multi-peer set up, the environment becomes more complex as it needs to have a propitiate crypto-material generated for users but also for each B-VMOA node. For example, to access the VMOA-peer1 node, one must have the appropriate certificate issued on local machine, but also a VMOA-peer1 host machine must have its certificate so it can interact with other nodes. Since different components are run

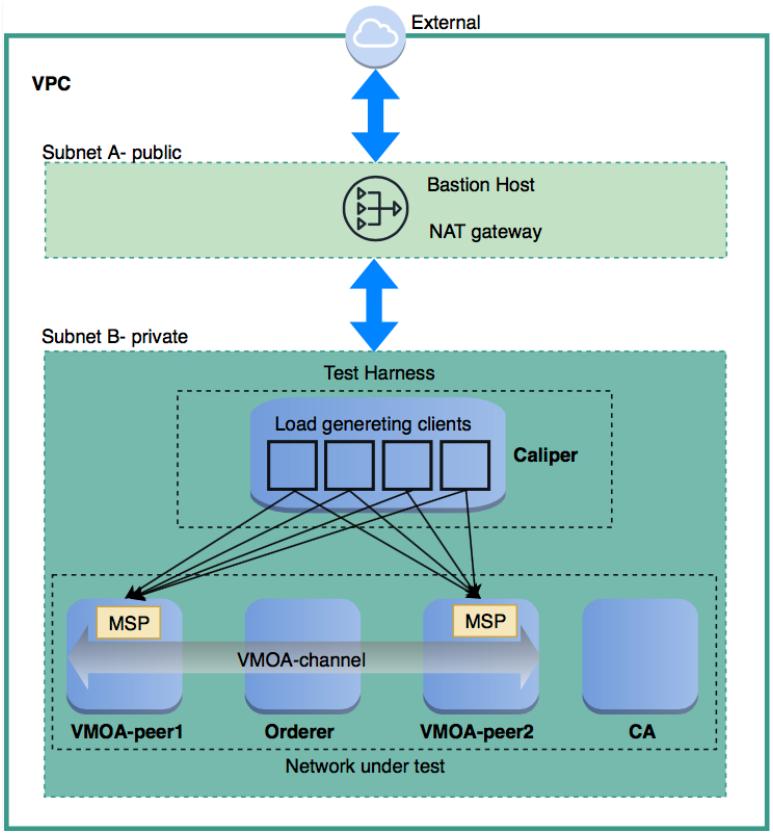

Figure 5.3: Experimental B-VMOA setup.

on different physical machines each node must have access to its local MSP on its own VM file-system. Furthermore, adequate artifacts have to be created: (i) Genesis block and (ii) Channel transaction file and transfer to the corresponding node along with adequate YAML files with network setup. Furthermore, all nodes have to run appropriate executable HLF binaries.

5.4 Test harness

Our test harness leverages Caliper benchmark tool [175]. One of fourteen projects under Hyperledger Green-House, Caliper is a blockchain performance benchmark framework, which facilitates testing blockchain solution with predefined use-cases. Using Caliper, one can send controlled transaction workloads to the *system under test* and measure the resulting transaction throughput and latency. In the lack of standard benchmarks for blockchain, we chose to build Caliper on client host as it is compliant with desired metrics definitions providing adequate interfaces for Fabric SDK. We set up Caliper to run on the client machines within the same cloud subnet as our VMOA nodes. Besides as a load generator, we use Caliper to check for transaction confirmations from the SUT by way of block events received from peers, assigning those transactions a completion timestamp required for performance evaluation.

Caliper is split into packages that are managed by Lerna [2], a tool for managing JavaScript projects with multiple packages, hence we had to first pull the required base dependencies, and then bootstrap the Caliper project. Caliper can be abstracted into two components: (i) Caliper core which consists of the Caliper engine and all modules, and (ii) Caliper adapters, implementations of the Caliper blockchain interfaces.
We setup four clients to send transactions to at least one peer. Each experiment varies transaction load within the range starting from 25 tps to 150 tps, which was the maximum capacity for client node used in our experiments. All clients send generated transaction load at a specified send rate, halts for 5 secs, and starts the next round. For each transaction arrival rate we generated 1000 txs, hence the SUT was subjected to more than 18000 transactions. At the end of the benchmark test, an average of the throughput and latency is calculated with respect to metrics defined in the previous section.

5.5 Experimental results

In this section, we examine the impact of various configurable parameters to conduct an in-depth study of core components and benchmark performance for common usage patterns. We present our observations with aim to derive some high-level guidelines and identify system bottlenecks.

A. *Impact of different transaction phases and resource allocation*

Transaction latency is amount of all the delays during the transaction flow: (i) Endorsment Phase, (ii) Ordering Phase, and (III) Validation phase. Those phases involve different system chaincodes. Unlike user chaincodes, the system chaincodes are built into the peer executable, yet they have the same programming model.

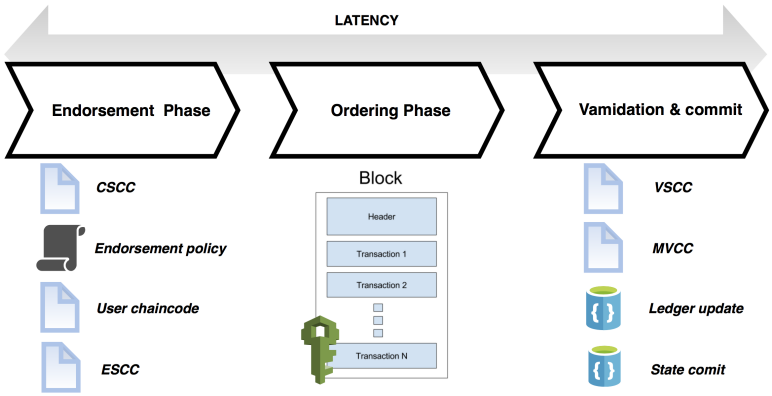

Figure 5.4: Impact of different transaction phase.

Fig. 5.4 illustrates the various system chaincodes and procedures run by peer and orderer nodes regarding transaction three-phase flow:

[2] www.github.com/lerna/lerna

I *Endorsement Phase.* A client invokes a user chaincode by way of transaction proposal (invoke transaction). Configuration System Chaincode (CSCC) is used to manage channel configurations. Client knows the list of endorsing peers to send transaction proposal from endorsement policy. Endorsers execute transaction running the user chaincode. To endorse the transaction, peer calls the Endorsement System Chaincode (ESCC) to sign the transaction response with peer identity and reply to the client.

II *Ordering Phase.* The ordering service does not inspect the transaction content. It is in charge to order transaction, create a block, sign it with its identity and deliver them to peers by leveraging gossip protocol.

III *Validation Phase.* All peers on the network validate all transactions within the block before they append it to its ledger. Validation System Chaincode (VSCC) evaluates transaction validation against the endorsement policy. Multi-Version Concurrency Control (MVCC) check ensures that the version of key read during endorsement phase match the key version in the local ledger state to perform read-write conflict check. Furthermore, the ledger is updated by way of appending the block to the local ledger. Unlike the ledger, state database only commits the write-sets of transactions marked as valid during MVCC check.

To optimize resource allocation for B-VMOA activities by minimizing the cost incurred by that allocation, we vary the number of CPU cores on peer nodes and study its effect. As part of system chaincodes, B-VMOA peers are submissive to CPU-intensive signature computation and verification routines. User chaincodes, i.e. a set of functions attempting to modify the data state against the ledger, add to this mix by way of transaction simulation during the endorsement phase.

Fig. 5.5, Fig. 5.6, and Fig. 5.7 show impact of peer resource allocation during the three-phase transaction flow. We note the following observation:
(i) As expected, average transaction latency decreases while the number of allocated vCPU increases. At ordering service, latency was approximately constant around 5 ms. This is because we varied the number of CPU cores on peer nodes which have an impact only on endorsement and validation phase. We notice that with an increase in the number of allocated vCPUs, endorsement latency decreased for 26%, from 14,2 ms to 10,5 ms, while validation latency decrease less, approximately by 15%, from 11,9 ms to 10,14 ms.
(ii) Query transaction latency decreases linearly with increase in vCPU allocated to node peers. The reason is that the read workload comprises of transactions that read the values for selected keys within the chaincode by way of performing lookups within its local database. Hence validation phase is omitted which indeed is the most time consuming phase (as shown in Fig. 5.5).
(iii) Each phase involves several system chaincodes and CPU-intensive routines. Fig. 5.7 plots impact of resource allocation on different routines performed by each phase according to Fig 5.4. While CSCC called during endorsement phase, and *state commit* during the validation phase are lightweight procedures only causing transaction latency of approximately 1 ms, the impact of transaction execution time and ledger update is the most important one. Namely, transaction execution time dropped by 20% from 4,78 ms to 3,8289 ms while the ledger update procedure showed to be the very system bottleneck decreasing its latency only by 11%, from 7,18 ms to 6,41 ms.

B. *Impact of the the block size and transaction arrival rate on performance*

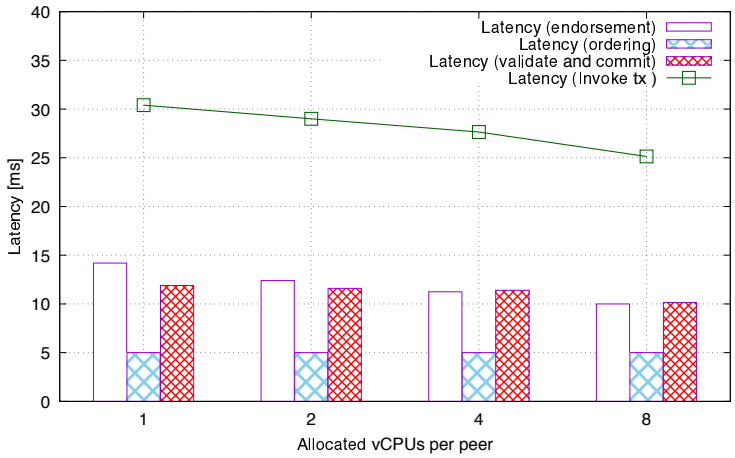

Figure 5.5: Resource allocation impact on three-phase transaction flow.

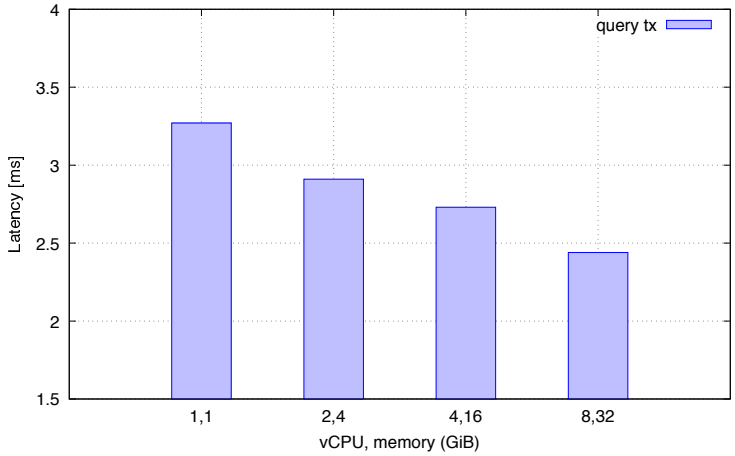

Figure 5.6: Resource allocation impact on the read workload.

Fig. 5.8 and Fig. 5.9 show the impact of the the block size and transaction arrival rate on network throughput and transaction latency respectively. Table 5.2 presents arrival rates and block sizes we tuned within various experiments.
Hereafter we note the following observations:
(iv) As expected, the throughput increased linearly with the increase of transaction arrival rate until it flattened out around 108.5 tps. We were able to reach the saturation point of 108.5 tps for the arrival rate of 125 tps. However, with further increase in arrival rate throughput remained flattened around the saturation point. Transaction latency was fluctuating around 200 ms until it increased significantly when the arrival rate was above the saturation point.
(v) For the arrival rate lower than 100 tps (the saturation point), the latency increases as the arrival rate

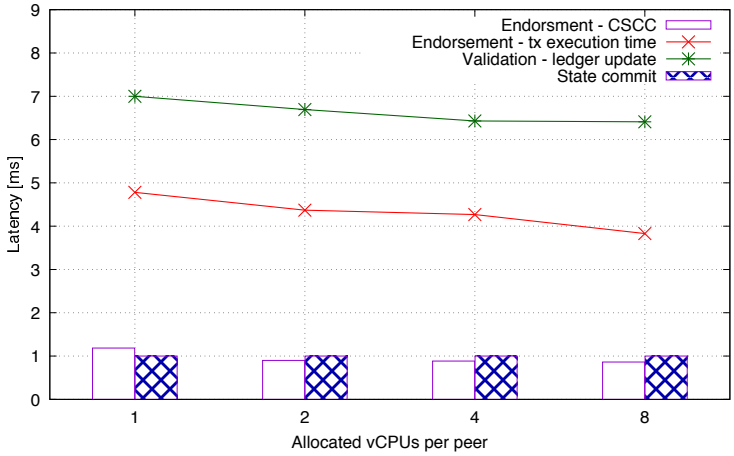

Figure 5.7: Resource allocation impact on CSCC, transaction execution, ledger update and state commit.

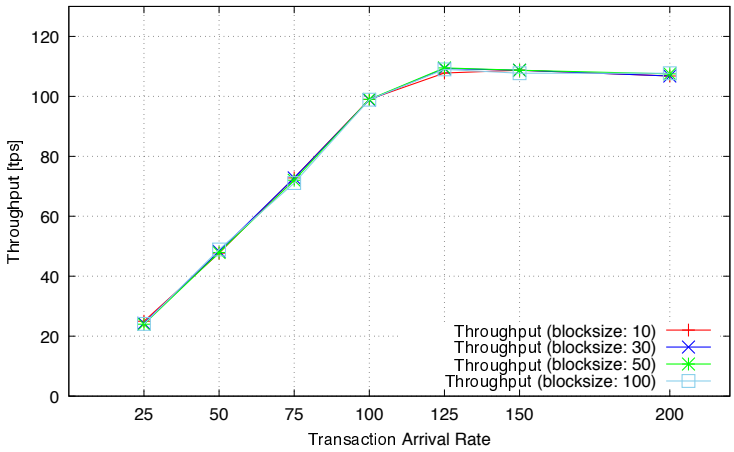

Figure 5.8: Impact of the block size and transaction arrival rate on throughput.

is lower than the block size while it decreases when the arrival rate is greater than the block size as the threshold. Blocks at higher arrival rate with lower block size were created faster rather than waiting for the block timeout, hence the waiting time at the orderer is reduced. Nevertheless, blocks with higher block size experienced higher latency with the increase of arrival rate. As the number of transactions in blocks increased, so did the time during validation and commit phase.
(vi) Latency decrease as block size increase for arrival rates greater than the saturation point. The time to validate and commit one block of size x is shorter than time needed to validate y blocks of size $\frac{x}{y}$.

C. *Ledger database impact on performance*

Among various configurable parameters, ledger database is actively involved in the transaction pro-

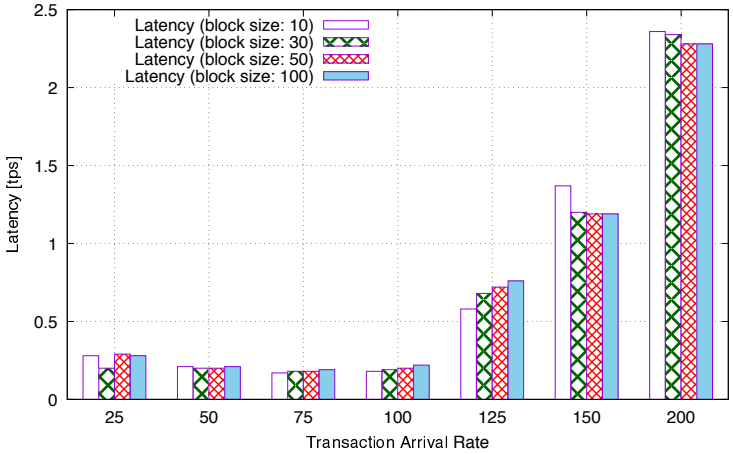

Figure 5.9: Impact of the block size and transaction arrival rate on transaction latency.

Parameters	Values
Transaction Arrival Rate [tps]	25, 50, 75, 100, 125, 150, 200
Block Size [tx]	10, 30, 50, 100
Load generating clients	4
StateDB Database	GoLevelDB
Resources	8 vCPU per peer

Table 5.2: Experiment configuration: The Impact of Block Size and Transaction Arrival Rate.

cessing, hence we study its impact on transaction latency and throughput. Fig. 5.10 and Fig. 5.11 plots the average throughput and latency for CouchDB and LevelDB ledger over different transaction loads. Workload comprised transactions with different complexities, differentiating workloads by the number of read-write operations:
a) *One read - one write*, with a simple function of adding new participants and assigning assets to them, corresponding to B-VMOA participant (VMM, orchestrator) creation transaction and resources linking to their account, for example, RAM. Before creating new participants, the function checks if there is already one with the same id, thus this transaction performs one read and one write operation.
b) *One read*, a simple query that reads the values for selected keys within peer local ledger.
c) *Two reads - two writes*, which checks the participants' current asset state and trade predefined amount of asset between them, thus it is compliant with B-VMOA trade transactions and involves two read and two write operations.

Hereafter we discuss our observations:
(vii) Transaction throughput significantly differs for transactions with high complexity (2r, 2w) with transaction throughput doubled with LevelDB compared to CouchDB. The maximum throughput for transactions with the complexity of (2r, 2w) with LevelDB was 85,1 tps, while we achieved the maximum of 42,8 tps with CouchDB. Furthermore, with an increase in the transaction complexity, the

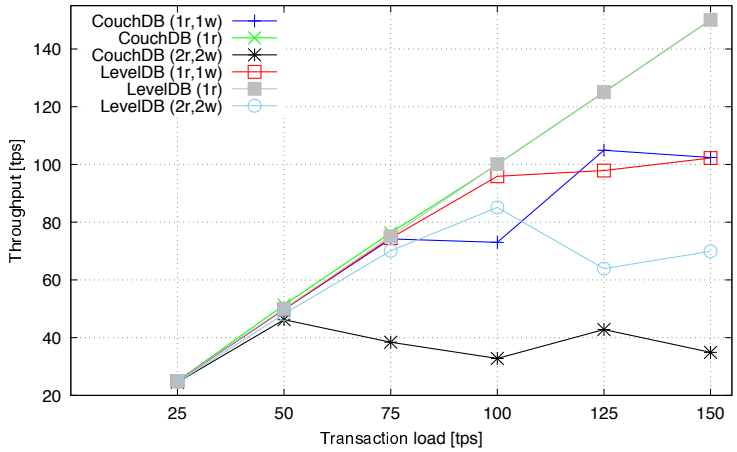

Figure 5.10: Ledger Database impact on open, query, and invoke transaction throughput.

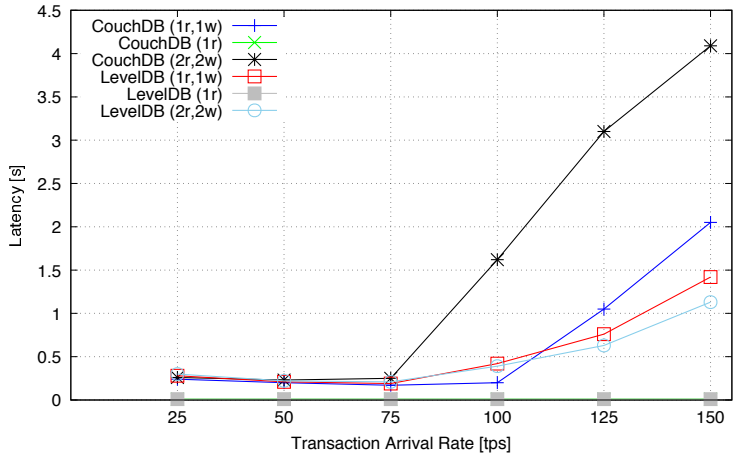

Figure 5.11: Ledger Database impact on open, query, and invoke transaction latency.

throughput with CouchDB dropped for 25% while LevelDB outperformed CouchDB with throughput decrease of 17,8%.

(viii) For transactions with higher complexity and arrival transaction rate up to 75 tps, both LevelDB and CouchDB ledger encounter transaction latency of the same order of magnitude. Overall, LevelDB outperformed CouchDB over all transaction rates with divergence more accentuated for arrival rates higher than 75 tps. With increase in transaction complexity and transaction load, the latency with CouchDB increased drastically from 0,26 s to 4,09 s, while with LevelDB we experienced increase in latency from 0.21 s up to 1,13 s.

(ix) With an increase in transaction arrival rate, query transactions did not experience throughput degradation nor an upsurge in latency, as shown in Fig. 5.10 and Fig. 5.11.

The reason for performance variation between LevelDB and CouchDB is that the later is accessed by peer nodes by way of REST API calls over a secure HTTP, while the former is embedded as part of

the peer process. With an increase in the transaction complexity, i.e. the number of write operations, the number of CouchDB REST API calls linearly increase trying to retrieve the endorsement policy for each transaction, hence low-throughput and high-latency performance compared to LevelDB.

D. *Raft and Solo ordering service*

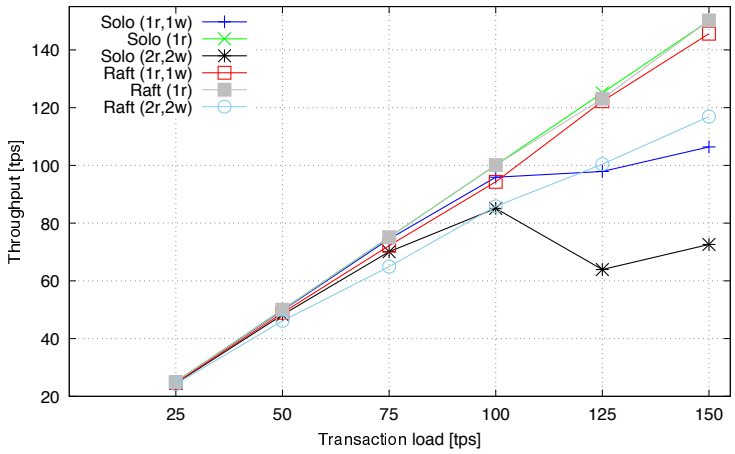

Figure 5.12: Transaction throughput with Solo and Raft ordering service.

Permissioned blockchain underlies B-VMOA system which unlike legacy public blockchains relies on deterministic consensus algorithms, hence any block a B-VMOA peer validates that is generated by the ordering service is guaranteed to be final and correct. Thus, ledgers cannot fork the way they do in many other distributed blockchains. While every ordering service available within current HLF v.1.4.1 release handles transactions and configuration updates in the same way, there are several different implementations for achieving consensus on the strict order of transactions between ordering service nodes.

The Solo implementation of the ordering service is not fault tolerant. Namely, it features a single ordering node, hence cannot be considered for production, but rather for testing applications and smart contracts or for building proofs of concept. Raft is a new consensus plugin introduced after Kafka and Solo based ordering system in the latest release 1.4.1. Raft brings additional capabilities to ordering services which makes it a production-ready system. As a distributed crash fault tolerance consensus algorithm, RAFT enables that in the event of a node failure, B-VMOA system is able to take a decision and process clients request.
We deployed experimental setup as shown in Fig. 5.2 within a single host to study the impact of production-like B-VMOA environment with distributed cluster of ordering nodes. Fig. 5.12 plots transaction throughput for networks with Solo and Raft as ordering service over several transaction arrival rates. Raft ordering service consisted of a cluster with three orderer nodes while solo implementation featured a single ordering node. Workload comprised transactions with different complexities, differentiating workloads by the number of read-write operations as in the previous experiment. We note the

following:
(x) As expected, solo and raft scale linearly with the increase in arrival rate when transactions contain one read operation. The reason is that a transaction with complexity (1r) carries a simple query that reads the values for selected keys within local ledger, hence ordering service does not have an impact on its performance.
(xi) For transaction loads lower than saturation point (100 tps) and transactions with higher complexity, i.e. (1r, 1w) and (2r, 2w) Raft throughput was in the same level of magnitude as Solo throughput. Yet raft impacts throughput to drop for approximately 7,4% due to orderers cluster distribution. For transaction loads higher than saturation point (100 tps) and raft ordering service, throughput continued to increase until it reached the saturation throughput of 116 tps, unlike with solo ordering service when throughput degraded for 17%.

5.6 Conclusion

In this chapter, we presented our study on how various configuration parameters impact the transaction throughput and latency. We vary various configurable parameters such as block size, transaction arrival rate, resource allocation, transaction complexity and ledger database to provide various guidelines on finding the right set of values for those parameters. As a result of our study, we provide the following guidelines on configuring those parameters:

- For transaction arrival rate lower than saturation point, lower block size result in lower transaction latency and throughput that matches the arrival rate.
- To achieve high throughput and low latency over arrival rates higher than saturation point, one should always use the higher block size.
- Separating the endorsement, ordering, and validation during transaction process gives B-VMOA advantages in performance and scalability, eliminating bottlenecks which can occur when execution and ordering are performed by the same nodes. Yet, ledger update and transaction execution procedures are indeed the performance bottleneck compared to the other system chaincode execution time.
- For a state database, GoLevelDB is a better choice in terms of performance. CouchDB has better rich-query support than LevelDB, hence it is a better choice for read-only transactions. Nevertheless, B-VMOA transaction load consists of several transactions with high-complexity, i.e. with more than one write operations, hence LevelDB is an adequate choice for B-VMOA state database.
- To avoid ordering service to be a single point of failure, B-VMOA system should have a cluster of Raft ordering nodes. With distributed ordering service transaction throughput is the same level of magnitude as in solo setup, while tolerating up to n faults within a system of 2n+1 nodes.

Chapter 6

Data Center Network topology impact on Scaling Up B-VMOA control plane

Summary

6.1 Introduction

Large-scale data center network (DCN) provide the core infrastructure to meet computing and storage requirements for both enterprises and cloud-based services. Nevertheless, B-VMOA is meant to be run within a data center, hence the B-VMOA hosts are actually servers running B-VMOA agents. Different data center interconnection topologies scale differently due to underlining protocol designs and

different infrastructure interconnections. Besides, Hyperledger Fabric optimizes B-VMOA network performance, security, and scalability by way of workload separation across: (i) transaction execution and validation peers, and (ii) transaction ordering nodes. It is essential to have a secure, reliable and scalable data dissemination protocol to ensure data integrity and consistency. To meet these requirements, a gossip data dissemination protocol is used. Overhead of B-VMOA control plane traffic is directly impacted by constant 'heartbeat' message unicasting between B-VMOA peers. Furthermore, to support the growing cloud computing needs, the number of servers in today's data centers is increasing exponentially, thus resulting in enormous challenges to efficient network design for interconnecting these servers so that the deployment and maintenance of the infrastructure is cost-effective. In this chapter we analyze the structural properties of several DCN topologies and also present some comparison among these network architectures with aim to reduce B-VMOA overhead costs. The analysis in this chapter is considering only private cloud intra-DC deployment of B-VMOA. As result, we suggest to adopt hypercube topology as a solution to address the performance bottleneck in the B-VMOA control plane.

6.2 Gossip protocol

B-VMOA peers leverage gossip protocol to disseminate ledger and channel data. Gossip messaging is continuous, and each peer on a channel is constantly receiving current and consistent ledger data from multiple peers. Each gossiped message is signed, thereby allowing Byzantine participants sending faked messages to be easily identified and the distribution of the message(s) to unwanted targets to be prevented. Peers affected by delays, network partitions, or other causes resulting in missed blocks will eventually be synced up to the current ledger state by contacting peers in possession of these missing blocks. The gossip-based data dissemination protocol performs three primary functions:

(i) Manage peer discovery and channel membership, by continually identifying available member peers, and eventually detecting peers that have gone offline.

(ii) Disseminate ledger data across all peers on a channel. Any peer with data that is out of sync with the rest of the channel identifies the missing blocks and syncs itself by copying the correct data.

(iii) Bring newly connected peers updated allowing peer-to-peer state transfer update of ledger data.

Gossip-based broadcasting operates by peers receiving messages from other peers on the channel, and then forwarding these messages to a number of randomly selected peers on the channel, where this number is a configurable constant. Peers can also exercise a pull mechanism rather than waiting for delivery of a message. This cycle repeats, with the result of channel membership, ledger and state information continually being kept current and in sync. For dissemination of new blocks, the leader peer on the channel pulls the data from the ordering service and initiates gossip dissemination to peers in its own organization. The leader election mechanism is used to elect one peer per trust domain which will maintain connection with the ordering service and initiate distribution of newly arrived blocks across the peers of its own organization. Leveraging leader election provides the system with the ability to

efficiently utilize the bandwidth of the ordering service. Static leader election allows to manually define one or more peers within an trust domain as leader peers. We stress, however, that having too many peers connect to the ordering service may result in inefficient use of bandwidth. Dynamic leader election enables peers within trust domain to elect one peer which will connect to the ordering service and pull out new blocks.

A dynamically elected leader sends heartbeat messages to the rest of the peers as an evidence of liveness. If one or more peers don't receive heartbeats updates during a set period of time, they will elect a new leader.

Online peers indicate their availability by continually broadcasting 'alive' (or heartbeat) messages with the public key infrastructure (PKI) ID and the signature of the sender over the message. Peers maintain channel membership by collecting these alive messages; if no peer receives an alive message from a specific peer, this peer is eventually purged from channel membership. Because 'alive' messages are cryptographically signed, malicious peers can never impersonate other peers, as they lack a signing key authorized by a root certificate authority (CA). This way of failure detection is simple and robust while remarkably efficient and fault tolerant. Nevertheless, this approach has certain inefficiency when it comes to network bandwidth consumption and has different scaling properties with respect to different DCN topologies and number of servers.

Specifically, for each network interface, a node sends a heartbeat message to all other nodes with interfaces on that network. In the common case where each node has an interface on each cluster network, there are $N(N-1)$ unicast heartbeats sent per network in an N-node cluster. The protocol algorithm assumes the ability to broadcast messages to all cluster nodes.

6.3 Data center network topologies

A data center networks (DCN) usually consist of computers, switches/routers, rack of servers, load balancers, cooling systems and other related equipment. The number of servers within data centers are becoming increasingly large due to an increase in the uptake of cloud computing. In general, DCN can be classified as following: (i) Tree-based topology, (ii) Recursive-defined topology, (iii) Hybrid network and (iv) Direct network [219]. In recent years, researchers have proposed several electrical/optical hybrid packet and circuit switched DCN architectures. Nevertheless, we chose to focus on packet switching DCN, while leaving the possibility to analyze Hybrid topologies in our future work. As we have mentioned above, we seek to minimize the overhead between multiple B-VMOA peers by properly choosing the most adequate interconnection topology. To ascertain our topology choice, we need to consider several aspects, among which the control overhead problem is the most important one. Namely, we need to consider what properties affect B-VMOA DCN performance and how to minimize the gossip overhead cost. Generally speaking, to choose the right topology for B-VMOA DCN, we need to maintain at least the following information:

1. The average number of heartbeats messages between B-VMOA peers;
2. Average path length between B-VMOA nodes;

3. Topology resilience;
4. Number of switches and other networking equipment needed for interconnection.

Let us discuss some structural properties of the following interconnection topologies: (i) Fat-tree topology (representative of tree-based topologies) (ii) DCell topology (one possible solution for recursive-defined topologies) and (iii) Hypercube topology (direct network topology).

6.3.1 Fat-tree topology

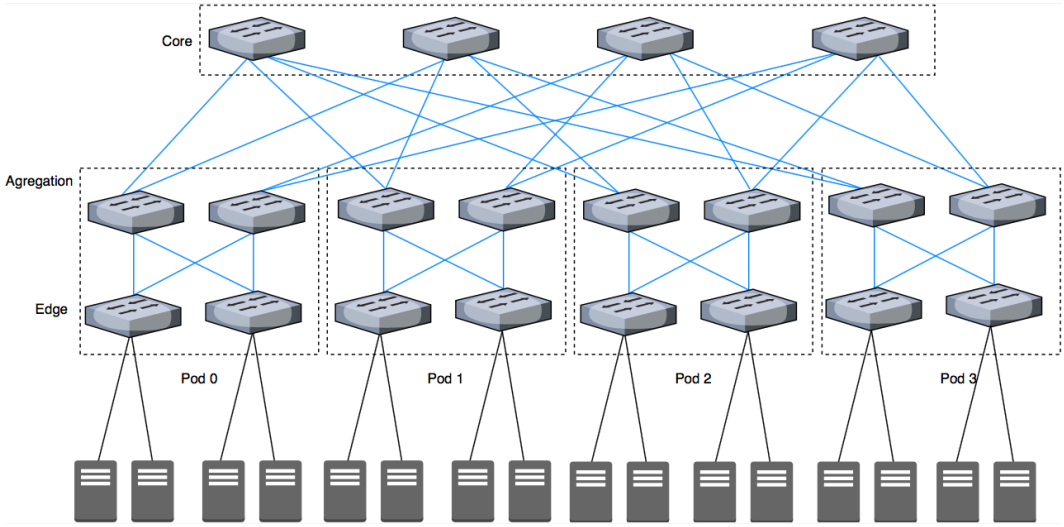

Figure 6.1: Fat-tree (Clos) architecture ($k = 4$).

The legacy three-tier DCN architecture follows a multi-rooted tree based network topology composed of three layers of network switches, namely access, aggregate, and core layers. One of the best-known interconnection topology to construct data centers has been proposed by Leiserson [220]. The authors proposed a special case of a Clos topology illustrated in Fig 6.1. Network is divided into *k* pods where each pod contains two layers of *k/2* switches. The lower layer is the edge layer where each *k*-port switch connects with the aggregation layer over the $k/2$ ports. Each switch in the aggregation layer uses the *k/2* ports to connect with the core layer and the remaining *k/2* ports to connect with the edge layer. The core layer contains $(k/2)^2$ k-port switches.

6.3.2 DCell topology

DCell topology shown in Fig 6.2 uses servers with multiple ports to built its recursively defined architecture [219]. A server in the DCell architecture is equipped with multiple Network Interface Cards (NICs). The DCell follows a recursively built hierarchy of cells. Basic unit is a Cell0 and building block of DCell topology which is arranged in multiple levels. Higher level cell contains multiple lower

layer cells. The Cell0 is building block of DCell topology and contains k nodes. One of them is used as commodity network switch to connect the rest of $n = k - 1$ servers within the same cell. A cell1 contains $k = n + 1$ Cell0 cells, and similarly a Cell2 contains $kn + 1$ Cell1 cells. A four level DCell with six servers in Cell0 can accommodate around 3.26 million servers, hence DCell is a highly scalable architecture. Furthermore, the DCell architecture depicts very high structural robustness.

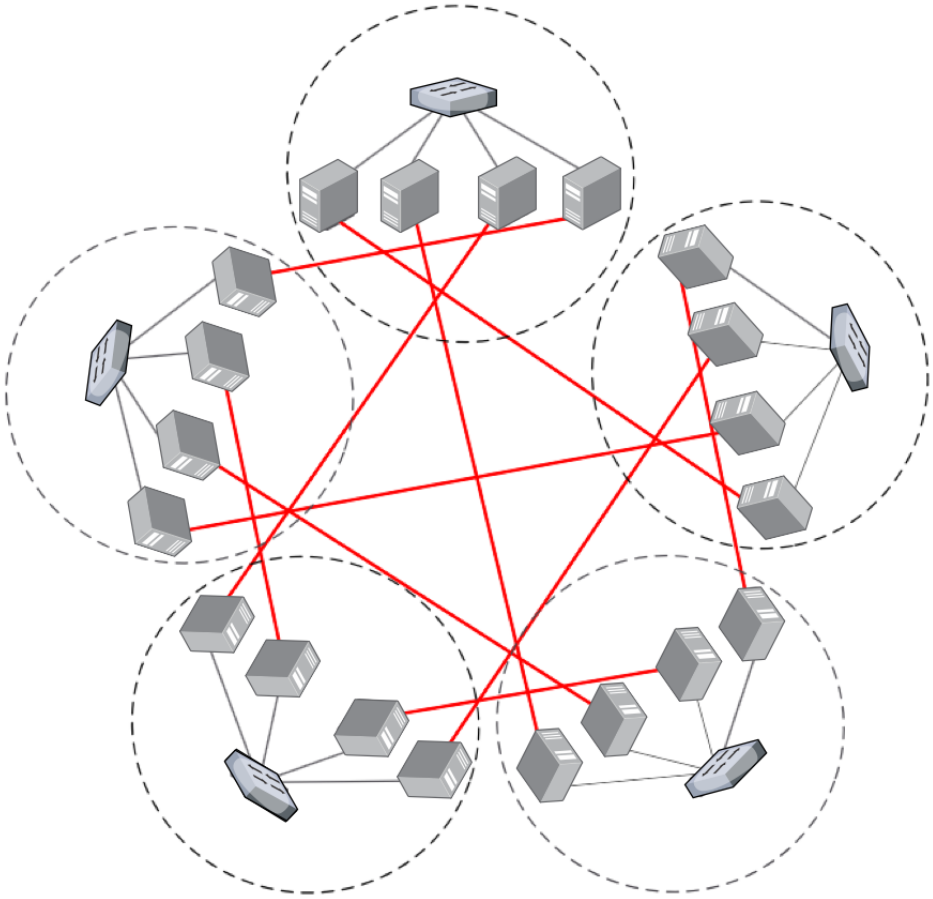

Figure 6.2: DCell topology with five cells

6.3.3 Hypercube topologies

The hypercube topologies have been widely used for processor interconnections [221]. Figure 6.3 shows an example of the hypercube topology with 16 switches where a regular cube has been replicated and the corresponding nodes have been directly connected. Moreover, the same replication principle can be applied in order to increase the dimension of the hypercube topology. The number of ports per switch needed to connect a switch with the rest of the network is a logarithmic function of the size of the network. However, this topology is a mesh topology which means that it allows any host in the network to communicate with any other host in the network. An advantage of the hypercube topology

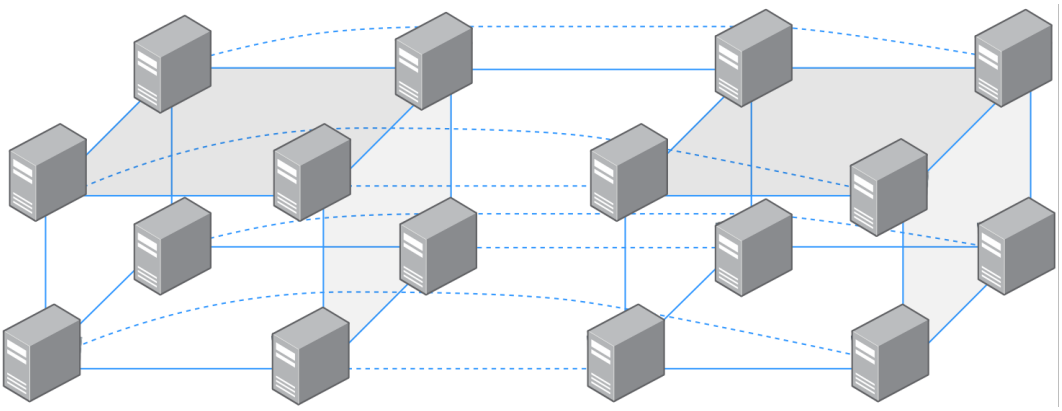

Figure 6.3: Hypercube topology.

is its high scalability as only one additional port per switch is needed in order to double the size of the whole network. We analyzed hypercube topology as a direct network topology, i.e., as a switchless network interconnection. Hence, the servers use multiple NIC ports to get involved in the network infrastructure and also be in charge of packet forwarding.

6.4 Comparison between topologies

In this subsection, some topology features are explored with aim to efficiently interconnect the servers within DCN to deliver peak performance for B-VMOA service. We conceived DCN where servers run VMOA nodes, interconnected in (i) Fat-tree, (ii) DCell and (iii) Hypercube topology. The analysis of structural properties of the aforementioned topologies are summarized in Table 6.1. All the derivations are detailed hereafter. All the B-VMOA peers leverage gossip protocol for data dissemination. B-VMOA agents are hosted on supported servers in DCN. In case of hypercube DCN, all server are directly connected without any switches, routers nor other network devices. Furthermore, servers perform the control and the data plane rules implemented on the node's network card (NIC).

6.4.1 Quantitative analysis of the structural properties

As discussed in Section 6.2, B-VMOA network leverages constant exchange of heartbeats messages between B-VMOA peer agents. However, it may eventually lead to network congestion due to the excessive overhead in control message exchange between B-VMOA peers. Considering that the three types of topology scale differently when involving different transmission paths, we need to study the amount of exchanged messages quantitatively. Let n designate the number of servers where each server run a B-VMOA agent. Let λ represent the heartbeat frame generation rate (per minute) that a single server sends to each server in the network cluster.

Topology feature	**Fat-tree**	**DCell - 2 layers**	**Hypercube**
Total number of links in the topology	$l_{FT}(n) = \frac{3k^3}{4}$	$l_{DCell}(n) = \frac{3}{2}k(k+1)$	$l_{HC}(n) = \frac{n}{2}\log_2 n$
Maximum number of B-VMOA hosts	$n = k^3/4$	$n = k(k+1)$	n
Average number of heartbeats frames generated per link	$n_{FT-live} = n\frac{1}{3}\lambda(n-1)$	$n_{Dcell-live} = \frac{2}{3}\lambda n(n-1)$	$n_{HC-live} = 2\lambda\frac{n-1}{\log_2 n}$
Average distance between B-VMOA hosts	$\mu_{FT}(n) = \frac{6n-\frac{1}{2}(4n)^{\frac{2}{3}}-(4n)^{\frac{1}{3}}-2}{n}$	$\mu_{DCell}(n) = \frac{5k^2-2k+2}{k^2+k+1}$	$\mu_{HC}(n) = \frac{1}{n-1}\sum_{k=1}^{log_2 n}\binom{log_2 n}{k}k$
Number of switches	$\frac{5}{4}k^2$	$(k+1)$	/

Table 6.1: The comparison between (i) Fat-tree (ii) DCell (iii) Hypercube network topologies where n is the number of B-VMOA hosts, and k denotes the number of layers in Fat-tree and DCell respectively.

6.4.1.1 Fat-tree topology

The number of links in the fat-tree topology can be calculated as a sum of the links above and below the aggregate layer. Number of links above the aggregate layer is equal to the number of nodes in the aggregate layer $k\frac{k}{2}$ multiplied by the number of links per node in direction of upper layer $\frac{k}{2}$. The same calculation is done for the edge layer which gives the total number of links in the fat-tree topology to be equal to:

$$l_{FT}(n) = k\frac{k}{2}\frac{k}{2} + k\frac{k}{2}\frac{k}{2} + k\frac{k}{2}\frac{k}{2} = \frac{3k^3}{4} = 3n \tag{6.4.1}$$

We stress that Fat-tree built with k-port switches has k pods, each of which contain two layers of $k/2$ switches. It consist of $(k/2)^2$ core switches, $k^2/2$ aggregation and edge switches respectively. The number of B-VMOA peer agents is equal to the total umber of supported servers $k^3/4$.

Hence, the total number of heartbeats or 'live' messages generated in the network is:

$$n_{FT-live}(n) = n(n-1)\lambda[min^{-1}] \tag{6.4.2}$$

The average number of heartbeats messages generated per link is:

$$n_{FT-live_{pl}}(n) = \frac{n_{FT-live}(n)}{l_{FT}(n)} = \frac{1}{3}\lambda(n-1) \tag{6.4.3}$$

In the fat-tree topology, we assume that the virtual machines are hosted under the edge level. The reason behind this is that, originally, in the fat-tree topology, the hosts are connected on the edge switches and the switches in the aggregation and the core layer were intended only to bring hierarchical interconnection of the edge switches. We follow this assumption in our case where the servers run B-VMOA agents interconnected with the corresponding switches in the fat-tree topology.

The average distance of the fat-tree topology can be calculated as the sum of distances from one B-VMOA peer node to other B-VMOA peer nodes under the same pad and the distances from the same node to all the B-VMOA nodes under all the other pads. Afterward, this sum is divided by the number of servers minus the one from which the distances are calculated:

$$\mu_{FT}(n) = [(\frac{k}{2}-1)2 + 4\frac{k}{2}(\frac{k}{2}-1) + 6(k-1)\frac{k}{2}\frac{k}{2}]\frac{1}{k\frac{k}{2}\frac{k}{2}-1} = \frac{6n - \frac{1}{2}(4n)^{\frac{2}{3}} - (4n)^{\frac{1}{3}} - 2}{n-1} \quad (6.4.4)$$

We note here that the average distance for the fat-tree topology has been calculated as the average distance between the servers assuming that B-VMOA agents are run on the hosts under the edge level. Otherwise, it would be illogical to use fat-tree topology if B-VMOA agents are hosted behind aggregate or core level as the initial hierarchical structure of this topology where the traffic is load-balanced over upper layers would be put in question.

6.4.1.2 DCell topology

Cell0 cell as a base unit for DCell topology has k servers and one node as a switch used to connect the rest of k servers in the same cell. Each of k server hosts in all $k+1$ Cell0-cells is connected with one of k server nodes from different cells. Besides, k server nodes in each of $k+1$ Cell0-cells are connected to the same k-port switch node. The total number of links can be calculated as following:

$$l_{DCell}(n) = \frac{1}{2}(k+1) + k(k+1) = \frac{3}{2}(k(k+1)) = \frac{3}{2}n, n = k(k+1) \quad (6.4.5)$$

The total number of heartbeats generated in the network is:

$$n_{DCell-live}(n) = n(n-1)\lambda[min^{-1}] \quad (6.4.6)$$

The average number of hearbeats generated per links is:

$$n_{DCell-live_{pl}}(n) = \frac{n_{DCell-live}(n)}{l_{DCell}(n)} = \frac{2}{3}\lambda n(n-1)[min^{-1}] \quad (6.4.7)$$

The average distance between $n = k(k+1)$ B-VMOA agent hosts can be calulated as a sum of distance between one server in one Cell0-cell and other $(k-1)$ servers within the same cell, plus the sum of distance between the same host to all server host in a cell directly connected, plus the sum of distance between the same host and all host servers within the rest of $(k-1)$ Cell0-cells. Than the sum is divided by the total number of B-VMOA agent hosts minus the one from which the distances are calculated:

$$\mu_{DCell}(n) = [2(k-1)+[1+3(k-1)]+(k-1)[3+5(k-1)]]\frac{1}{k(k+1)-1} = \frac{5k^2-2k+2}{k^2+k+1} \quad (6.4.8)$$

6.4.1.3 Hypercube topology

The number of links in the hypercube topology is equal to the number of nodes n multiplied by the number of links per node $log_2 n$ and divided by 2 as each of the links is calculated twice in this case:

$$l_{HC}(n) = \frac{n}{2} \log_2 n \tag{6.4.9}$$

The total number of hearbeats in the network generated per minute is:

$$n_{HC-live}(n) = n(n-1)\lambda[min^{-1}] \tag{6.4.10}$$

The number of heartbeat generated per link is:

$$n_{HC-live_{pl}}(n) = \frac{n_{HC-live}(n)}{l_{HC}(n)} = 2\lambda \frac{n-1}{\log_2 n}[min^{-1}] \tag{6.4.11}$$

The average distance of the hypercube topology can be calculated as follows. Let us index all the nodes of the hypercube by n bits. The neighboring nodes differ in only one single bit. Consequently, the distance between the two nodes is equal to the number of bits in which their indexes differ. This implicates that the number of neighbors of distance k is equal to $\binom{log_2 n}{k}$. As a result, the average distance from one node to all the other nodes in the network is:

$$\mu_{HC}(n) = \frac{1}{n-1} \sum_{k=1}^{log_2 n} \binom{log_2 n}{k} k \tag{6.4.12}$$

This also represents the average distance of the hypercube as hypercube is symmetrical and each of its nodes has the same average distance.

6.4.2 Discussion on topology choice

Let us discuss the results shown in the following figures. The results showed that all tree topologies scale different with respect to examined parameters. Furthermore, the result shows that DCN with hypercube interconnection topology scales better in the case where B-VMOA agents are hosted on DCN servers compared to fat-tree and DCell DCN architectures.

Fig 6.4 and Fig 6.5 show that DCell topology has appreciably less links for data centers up to approximately 500 servers. Hypercube and fat-tree scales similarly at first, while later Fat-tree outperforms hypercube giving the rise of its curve in Fig 6.5.

Nevertheless, heartbeats are constantly send over the network. This means sharing information among themselves in order to keep a global and consistent view of the network. The implication is that a lot of data is exchanged over the control plane. While this might not be an issue in a fat-tree and hypercube topologies, in DCell topology the issue of control overhead may become predominant, especially in

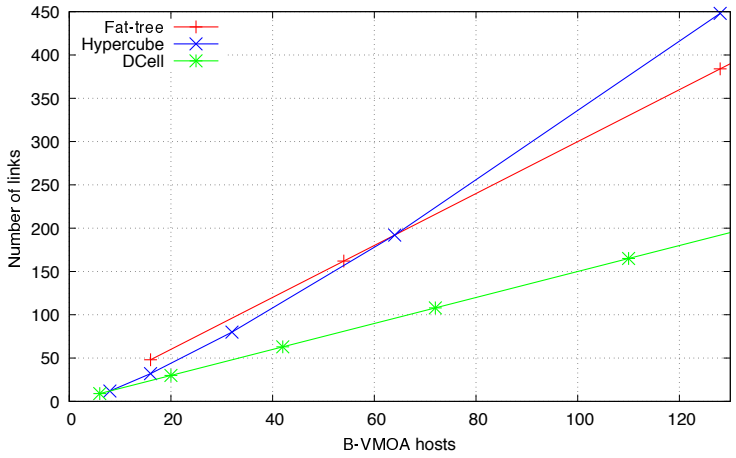

Figure 6.4: Total number of links.

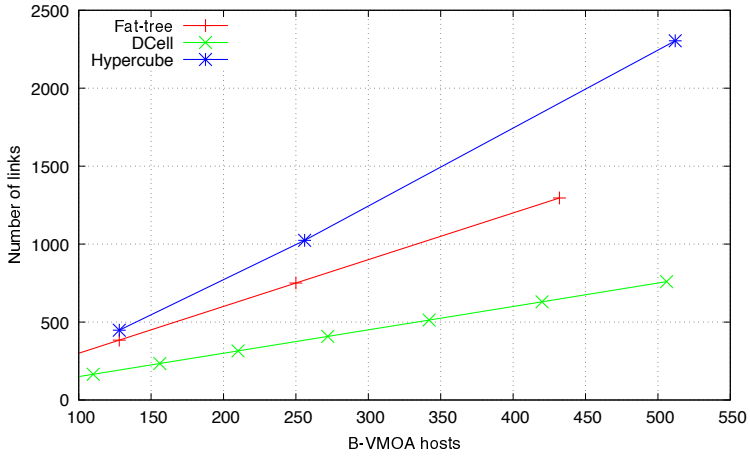

Figure 6.5: Total number of links.

networks with higher number of servers as shown in Fig 6.6 and Fig. 6.7. Furthermore, from Fig 6.7 we can also see that the hypercube topology generates lower number of heartbeats in comparison with fat-tree topology.

Fig 6.8 compares the topologies in terms of resiliency, i.e., the average number of links failures needed to cut-off one server. Hypercube topology significantly outperforms both, Fat-tree and DCell while Fat-tree behaves better when compared to DCell topology. Furthermore, the average distance between B-VMOA hosts differs significantly as can be seen in Fig 6.9. Hypercube has a considerably lower average distance, especially when the number of server is lower than 80. Fat-tree has approximately constant average distance. We note that the average distance for the fat-three topology is calculated as the average distance between the supported servers as we assume that only the servers host B-VMOA peer agents. From Fig. 6.10, we can see that with an increasing number of B-VMOA hosts, the

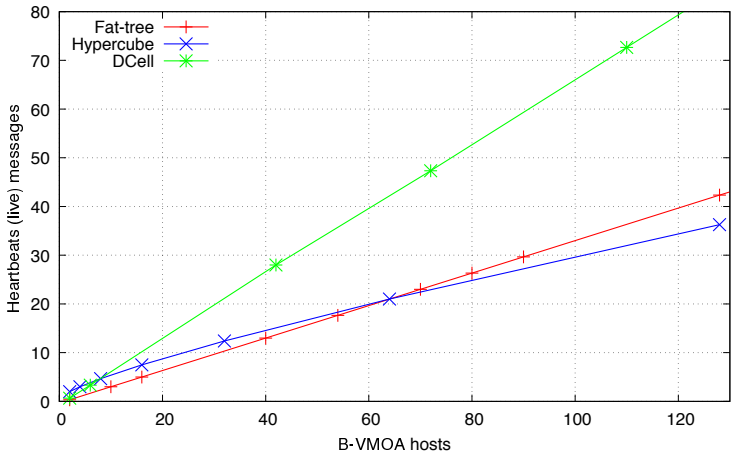

Figure 6.6: Number of heartbeats for data-centers up to 100 servers.

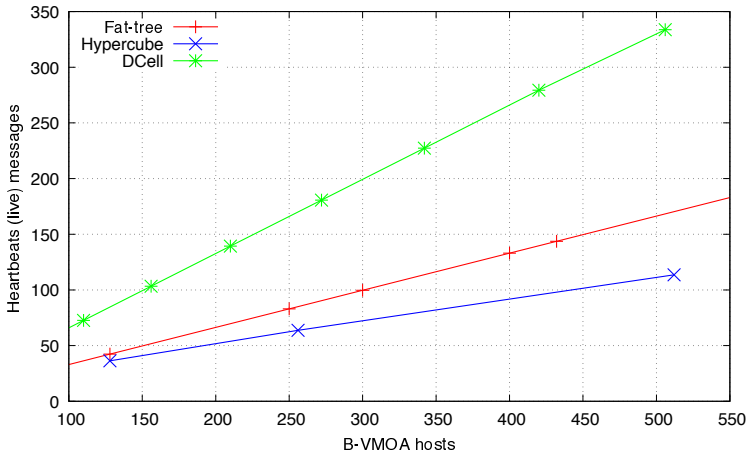

Figure 6.7: Number of heartbeats for data-centers with more than 100 servers.

number of switches used by DCell topology is far less than fat-tree, while hypercube leverages direct connection between servers.

6.5 Conclusion

In this chapter we provided an analytic comparison of different network topologies and their impact on the load induced by the B-VMOA control plane. More specifically, our analyses integrates the following features: first, it exploits gossip control plane overheads; second, it proposes a hypercube topology for B-VMOA DCN to reduce the transmission overhead. The evaluation results show that hypercube B-VMOA DCN can achieve higher performance than legacy fat-tree and DCell, with preserving servers' CPU and memory resources by way of packet processing on the smart NIC and saving

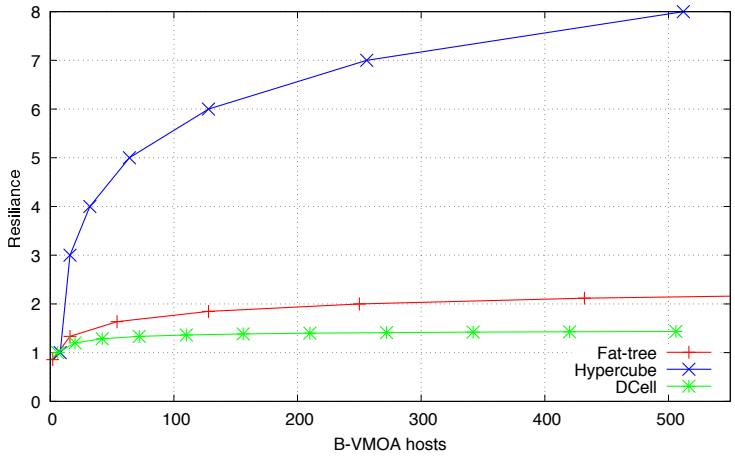

Figure 6.8: Topology resilience

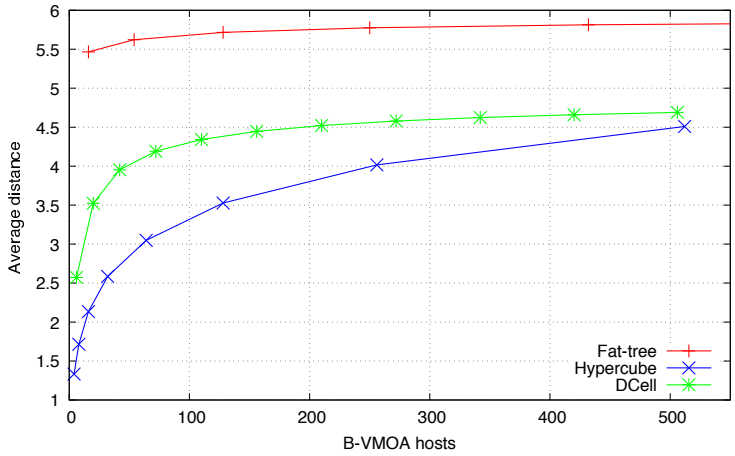

Figure 6.9: Average distance.

infrastructural cost due to its switchless structural design.

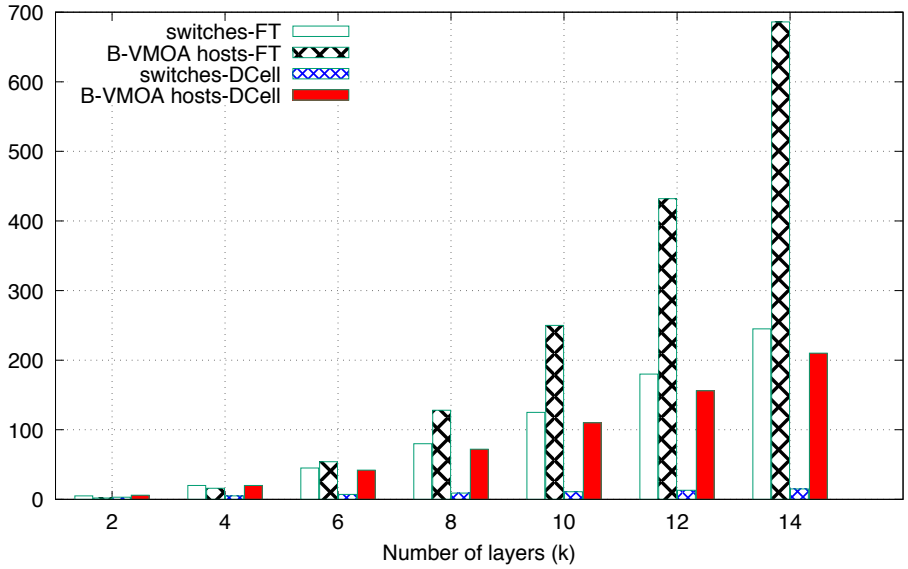

Figure 6.10: Number of switches and B-VMOA hosts.

Chapter 7

Conclusion

In this dissertation, we study blockchain technologies from different perspectives. Blockchain can be regarded as a quality leap from the distributed database technology. Generally, Distributed Ledger Technologies (DLTs) are designed to deal with black database in the form of data shared in a distributed manner, and blockchain represents one possible DLT to do it. Blockchain allows sharing a ledger of transactions that are read, validated and stored in a chain of blocks. Systems based on the blockchain technology work in a distributed manner, involving multiple agents or participants that ought to be independent of each other, and which can use peer-to-peer communications (P2P) to structure themselves into a network collectivity. In contrast to legacy client-server architectures, P2P network nodes do not always have specific roles, a fixed hierarchy; roles may not exist or may change over time depending on the actual operation behind a communication, i.e., a blockchain transaction. The adoption of P2P as communication paradigm adequately supports the goal that resources are shared and dispersed over a network which by construct forbids the existence of providers or servers centralizing tasks. The result is a decentralized ecosystem with no central authority. Blockchain can hence be used in diverse sectors with several applications. However, it is crucial for users to understand whether the technology fits the problems that they are aiming to solve or not. There may be cases where the price paid for decentralization results commercially unreasonable, and this is one of the reasons why regular databases are still widely used. Fundamental bricks in the design of a blockchain technology are as follows: (i) communications and transaction data storage are regulated by cryptographic security, network nodes have to agree on both the validity and the order in which transactions are listed in the blockchain, (ii) distributed consensus protocols solve these issues in a scenario where each node comes to vote. Leveraging on the important background presented in this dissertation, we design our blockchain vademecumc to give to the reader a comprehensive tutorial about when to use blockchain, which solution to use, and how to use it, based on use-case requirements. We aim at providing the community with such a vademecum, while giving a general presentation of blockchain that goes beyond its usage in Bitcoin and surveying a selection of the vast literature that emerged in the last few years. We draw the key requirements and their evolution when passing from permissionless to permissioned

blockchains, presenting the differences between proposed and experimented consensus mechanisms, and describing existing blockchain platforms. Blockchains have spawned a new era of systems development, and this thesis sets the foundation for future research in modeling and analysis in this area. Proposed vadamecum and analysis presented in this thesis are useful for both, the general community and blockchain system developers and the architects deploying blockchain systems in the field. Reader gain insights into the inner-workings of the system that provides guidance on which subsystems and functions to improve and its implications on the system performance.

For a new technology to realize its full potential, a lot of circumstances need to co-exist before network effects can be realized. In order for the technology to bring in systemic efficiencies, a critical mass needs to be attained. As an infrastructure technology, all major players in the market need to collaborate to define standards in a democratic manner. The blockchain community is indeed witness in unprecedented levels of industry collaboration between players who are otherwise competitors in the space. Because of the cost of moving from one infrastructure technology to the next, an open source collaborative approach is the most promising way forward. This is the direction we insisted in this disertation, highlighting not only when and which blockchain technologies should be chosen, but also how they can be used and deployed. From a societal perspective, while there has been an exponential increase in the interest around blockchain technologies, there is a huge lack of technical experts. As an illustration of the current societal perspective on blockchain due to ongoing innovation and development in the blockchain space, there is still not a consensus on a clear blockchain definition, despite we tried in this work to clarify key properties of blockchain, somehow giving an axiomatic view on possible different blockchain definitions.

As a second contribution, we propose new ecosystem based on our research work toward the design and implementation of a private blockchain system for managing the authentication of virtual machine orchestration commands in cloud computing and network function virtualization systems. Our main contribution is a B-VMOA model that capture critical steps performed by each data center stakeholders, i.e., VMMs and cloud orchestrators and the interactions between them. Proposed system aims to break vertical separation between orchestration management plane and virtualization servers only used to host VMs. We break this separation by way of delegating to multiple stakeholders (virtualisation servers) voting rights along with dispatching orchestration logic to servers which are no longer used just as hosting machines. Namely, we propose a new ecosystem run within private data centers to authenticate orchestration instructions and secure virtual machine orchestration. We also provide a detailed empirical analysis of model parameterization and validation using Hyperledger Fabric technology. Hyperledger Fabric is an open-source implementation of the distributed ledger platform for running smart contracts in a modular architecture. It is gaining popularity with more than 400 proof-of-concept and production implementations across different industries and use-cases. We deploy B-VMOA using HLF platform in private cloud environment to validate our system. The performance of blockchain system is a major concern for B-VMOA application, hence we conducted empirical study to characterize system performance and identify potential performance bottlenecks. Furthermore, we detailed the tools we used and network setup for reader to gain insights into the inner-workings of the system which would be valuable guidelines to developers and deployment engineers. We aim to derive several empirical observations and design guidelines which would be valuable to system architects and designers.

Our last contribution is an analytic comparison of different network topologies and their impact on the

load induced by the B-VMOA control plane. More specifically, our analyses integrates the following features: first, it exploits gossip control plane overheads; second, it propose the hypercube as the topology for B-VMOA DCN to reduce the transmission overhead. The evaluation results show that hypercube B-VMOA DCN can achieve higher performance than legacy fat-tree and DCell topologies, with preserving servers' CPU and memory resources by way of packet processing on the smart NIC and saving infrastructural cost due to its switchless structural design.

Future research directions

As a part of future work, we will study scalability and fault tolerant capability of B-VMOA using different blockchain topologies such as different number of peer nodes, trust domains, and number of peer per trust domains. It will be useful to validate the model for a large number of peer nodes and a wide range of parameters and system configurations. In addition, arrival rates in a real world production system would be following certain distribution, hence as a future work we plan to study the performance with different arrival rate distributions. Future research work should focus on exploring all B-VMOA potentials. For example, HLF supports ledger and chaincode isolation between different participants by way of channels, hence could be used to create new ecosystem not only within a single private data center, but rather between multiple cloud providers. This could enable VM auditing and secure migration between multiple cloud stakeholders while creating a more competitive and user-oriented cloud market. In such a case, just like permissioneless blockchain, permissioned blockchain are expected to run across a wide area network, hence the network latency would have a significant performance impact. Furthermore, for such a system, we would need to pay attention on how to chose a set of parameters and nodes to delegate validation process to omit a high propagation delays like in Bitcoin network.

Appendices

.1 Cryptography

Cryptography allows sending data trough trustless channels in a secure and verifiable way. Data hashing consists in a basic cryptographic operation that not only compresses data in a fixed-length format, but it does so irreversibly, which is crucial for ensuring the integrity of digital assets when transferred in the network. Asymmetric cryptography authenticates the data source and ensures its reception by the desired user. Blockchain combines asymmetric cryptography with hashing and *digital signature* schemes in order to provide fundamental security guarantees presented later on.

More precisely, a digital signature scheme consists of three phases as depicted in Fig. 1:

- *Key-pair generation phase*: each blockchain user generates a private key to sign a transaction with and a public key by which the receiver can verify the authenticity, integrity and provenance of the received data.
- *Signing phase*: the sender hashes the data and generates the digital signature with its private key; next, the signed hash is sent together with the encoded original data to the receiver. Data hashing not only makes the signature scheme more streamlined and efficient (data are compressed and have the same format), it also ensures the integrity of the transferred data (transactions contents are protected against being modified).
- *Verification phase*: the signed data is decoded with the sender's public key and compared with the re-computed hash value of the unsigned and uncompressed data.

Note that, in both the signing and verification phases the hashing function used must be the same (e.g., SHA256 for Bitcoin blockchain). Blockchain requires asymmetric algorithms – generating both public and private key – that allow for fast verification (the time taken for signing shall be the same as for the last phase). Digital signature algorithms in blockchains widely use elliptic curves (ECDSA [222; 223]).

.2 Digression on Consensus

.2.1 Consensus protocol properties

Consensus ensures nodes' agreement on a single request, or a sequence of requests also referred to as *atomic broadcast* [224]. Evidently, in any consensus protocol there are two events: the *proposal* and the *decision*. What nodes propose and decide is the interest they aim to agree upon, that in applications is most of the time a numerical value.

Fault-tolerant protocols are designed to deal with a limited number of faulty agents. According to [225; 226; 227], consensus reliability to halting failures is ensured by the following properties:

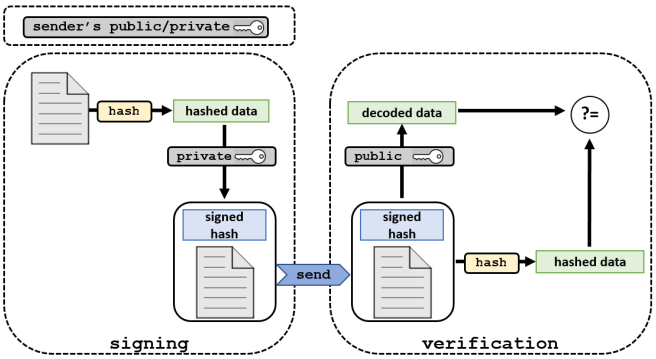

Figure 1: Phases of the digital signature protocol: (*i*) a public/private key pair is created – the public key can be recovered from the private one while the viceversa is not possible, (*ii*) data are signed – the signature is the result of encoding with the sender's private key the hashed data – and transferred. Once received (*iii*) the receiver decode data by the usage of the sender's public key and additionally verifies its authenticity.

- *Agreement*: every correct/honest node must agree on the same proposed value $\mathcal{V}$.
- *Validity*: if all nodes propose the same value $\mathcal{V}$, then all correct nodes decide $\mathcal{V}$.
- *Termination*: every correct node has to take a decision on a value $\mathcal{V}$.

Moreover, atomic broadcasts are reliable broadcasts satisfying the following property:

- *Total order*: if any correct node decides that value $\mathcal{V}_1$ comes before value $\mathcal{V}_2$, then every other correct node must order $\mathcal{V}_1$ and $\mathcal{V}_2$ at the same way.

Therefore atomic broadcasts are also known as *total order broadcasts* [228].

In [45; 229] authors grouped these properties in two classes: *liveness*, grouping validity and termination, and *safety* that incorporates the remaining properties. These properties are analyzed in [45] for atomic broadcasts characterized by a *broadcast* and a *deliver* event.

It is worth noting that blockchain applications may rise additional properties that can appear more important than those above to the designer. For instance, authors in [104] compare protocols in terms of *network identity management*, *energy consumption* and *adversary tolerated power*. Authors in [46] make comparisons in terms of security and performance; in particular, security is qualified in terms of *agreement* (i.e., the achievement of a consensus state) and the resistance to *transaction censorship* (i.e., the malicious behavior of suppressing transaction) and Denial of Service attacks [76]; and performance is qualified in terms of throughput (i.e., the transaction agreement rate), scalability (i.e., the system capability to respond adequately to a growth in the number of nodes) and latency (i.e., the time elapsing between proposal and decision phases during the consensus process). In [45] we find a comparison based on *liveness* and *safety*, while in [136] the comparison is limited to permissioned blockchains. A complete contribution on BFT protocols for replicated systems is provided in [230] where algorithm performances are evaluated in terms of cryptography costs, workloads, network conditions

and faults.

Eventually, in order to satisfy the desirable set of properties, a consensus protocol consists in a set of rules that each database transaction must respect. These rules, embedded in each blockchain node behavior implementation, are therefore application-dependent rules that can vary from system to system [231]. Therefore, consensus in blockchains is crucial since it characterizes the systems ensuring properties such as resilience and security that can be summarized by a desirable level of dependability [232; 233].

.2.2 Dealing with asynchronous communications

Networks can be *synchronous*, *asynchronous* or *partially synchronous* [234; 235]. Dealing with synchronous network does not mean dealing with networks where nodes' communications are not delayed in time; instead, it means considering message delays bounded by some value. In asynchronous networks, this upper bound does not exist or is flexible, as messages are supposed to be delayed arbitrarily. In partially synchronous networks, or *eventually synchronous* networks, asynchronous nodes present time windows where they behave synchronously. Partial synchrony offers a good adaptability to the real network behavior and, at the same time, simplifies network modeling. Both liveness and safety properties are guaranteed during synchronous periods. On the other hand, during periods of asynchrony liveness cannot be ensured as proven by the "impossibility theorem" [236] stating that deterministic protocols do not reach consensus in a fully asynchronous environment.

In order to overcome this limitation, fully synchronous networks opt for relaxing the deterministic constraint; they introduce randomness by requiring probabilistic termination (i.e., it is improbable for non-terminating executions to collectively occur) [237]. Authors in [238] proposed cryptographic solutions with *computational bounded adversary* (see Appendix .2.4) to overcome it. In partially synchronous networks, protocols correctly terminate during synchronous phases while they may stall during asynchronous ones, however termination is guaranteed under proper trust assumptions. More precisely, in order to preserve safety and liveness properties, this kind of protocols have to meet specific assumptions on the type and the number of faulty nodes in the network. In particular, fault-tolerant protocols typically work with a number n of nodes (replicas) exceeding twice the number of crashing nodes t and three times the number of Byzantine nodes b.

.2.3 Dealing with data consistency and consensus finality

An important impact on consensus has the "CAP" (Consistency, Availability, Partitioning) theorem [239; 240] stating that fault-tolerant distributed systems cannot guarantee at the same time full data consistency (i.e., the ability to have nodes storing the latest data version at the same time) and, complete failure independence (or high availability) in presence of a partition.

It is worth recalling that consensus implementation is a means for transaction validation and systems' resilience to failures. However, availability comes at the expense of consistency [50] whenever a network partition or failure happens. Thus, in general blockchain based systems aim at maintaining *eventual consistency*, i.e., consistency with time lags: all nodes get eventually a consistent view on the shared data, and in the convergence period upon each given change intermediate decisions may be taken, but eventually corrected based on the consistent store. Eventually consistent systems provide probabilistic consensus finality while consistent systems guarantee absolute finality.

.2.4 Integrating failure conditions

Summing up, each consensus protocol is characterized both by a *communication model* and a *failure model* which in turn is characterized by *trust assumptions*. Communications among nodes can be synchronous, asynchronous or can lie between the two cases. Failures may be of two types (crash and byzantine) and can characterize a certain number of nodes. Crash failures – where honest nodes may fail – must be distinguished from Byzantine failures – where nodes may act maliciously. Of the two types of failures, the Byzantine class involves several failure subtypes [241; 242; 243], which are far more disruptive than classical crash failures. More precisely, protocols in partially synchronous environments tolerate a number $t < n/2$ of crashing nodes and a number $b < n/3$ of byzantine nodes. Liveness and safety in synchronous or partially synchronous environments are guaranteed for those protocols working with $n \geq 3f + 1$ replicas, where f denotes the number of faulty nodes in general. In blockchains, properties and features result from a clever choice and implementation of a consensus protocol.

Consensus protocols, aiming at reaching an agreement state in the networks, satisfy their desired features and properties (such as liveness and safety) under some conditions. These are the so called trust assumptions characterizing the failure model of a protocol. These models are typically presenting bounds/threshold on the gap between two parameters referring to honest and malicious nodes respectively. Therefore, they are known in literature as "threshold adversary models" [229; 244]. The typical failure model foresees a threshold on the total number of nodes an adversary can control (f) with respect to the total number of nodes in the network (n). The threshold choice depends on the failure type and is between the half and a third (as previously met). However, this failure model presupposes knowledge of the number of parties involved in the network. Therefore, this classical adversary model works for permissioned networks where parties joining the system follows a specific membership protocol.

Bitcoin and other PoW-based cryptocurrencies consider an alternative failure model bounding no more the number of nodes but the work they may do. More precisely, the *computational threshold adversary model* limits the total amount of computational power that the adversary control (f_c) with respect to the total computational power (n_c). In order to guarantee double-

spending resilience Bitcoin selects a threshold of a minority $n_c > 2f_c$, namely the adversary can control a minority of computational power. Bounding computational power does not require knowledge on participating parties, therefore the model well adapts to PoW-based permissionless networks, where anyone can join the system.

Further adversary models can be found in literature; a new approach is the one of bounding the adversary *stake* (i.e., participation in a finite limited resource) [245], another option may be to adopt a game theoretical approach and therefore bounding adversary utility [246; 247].

.2.5 Proof-of-X Consensus

In the following we detail the PoW consensus, the PoS algorithm and, the PoS variations involving virtual mining.

.2.5.1 Proof-of-Work

The idea behind a PoW protocol is to make validation tasks difficult to perform, but trivial to verify. This idea was first presented as a solution to the email-spamming issue [248] and applied in a system called Hashcash [98]. The email sender should solve a cryptopuzzle finding the hash of a string, containing all the necessary information of the receiver, which has to meet a certain target. The usage of the Secure Hash Algorithm (SHA) [249], mapping data of arbitrary length to data of a fixed length in a non-invertible way, ensures a costly procedure to find a valid hash. B-money [247] suggested, in 1998, a PoW procedure where the computational effort can be easily quantified in terms of commodities baskets. At the same time, a PoW-based decentralized digital currency called Bit Gold was proposed [250] such that nodes should generate strings of bits using one-way functions with a cost expressed in number of compute cycles. The last Bitcoin's precursor, RPOW [251], incorporates the hashcash scheme creating Reusable PoW (RPoW) tokens. Bitcoin, as its precursors, uses a computational hard validation procedure to create rare and valuable goods. The real contribution brought by the system is the combination of decentralization, double-spending resistance, Sybil resistance and trustless node management with the "block-chain" architecture.

The PoW protocol consists in a race among nodes to be the winner and therefore gaining a reward of new minted tokens. The competition takes place among particular nodes, called *miners*, aiming at producing a valid PoW consisting in the hash value computation of a previous block header. In order to validate a block, the computed hash should meet a precise difficulty requirement. The nature of the problem relates the mining procedure to a lottery race where the validation process is completely aleatory and the probability of finding a valid hash is proportional to miners' computing power. Once the winner is found it acts as a leader node attaching to the blockchain its selected block of transactions. Its epoch expires with a new valid block, thus a new winner of the mining race. Bitcoin consensus provides for the

coincidence of both validator and leader roles in a single node. In general, PoW blockchains may separate the leader election (mining/transaction validation) form the transaction ordering procedure (i.e., Bitcoin-NG [21]).

Strong consistency would ensure a single chain of valid blocks published on the ledger. A PoW mechanism, however, guarantees consistency on a probabilistic form (forms of eventual consistency [2; 229; 252; 253]) since *forks* may occur. Whenever two blocks are validated approximately at the same time, or the network latency is delaying the transmission of a valid solution to the network, the result is the presence of two valid chains with the same block number. This inconsistent situation is solved with the validation of a new block through the *longest chain* rule: the chain with the most blocks is considered as the valid one, noting that the chain related to the greatest PoW effort may not be the longest chain [254]. The rule is proposed as a probabilistic solution to the Byzantine Generals problem [82]. Other variants of the longest chain rule were proposed in order to scale PoW blockchains: GHOST [255] proposed the *heaviest chain* rule that is confirming the block in the chain with the highest aggregate difficulty level, i.e., with the greatest computational load involved.

The economic incentives [256] resulting from the mining procedure induce miners to reduce the validation costs in order to maximize their earnings. Over the years the democratic idea pushed by Bitcoin of one-CPU-one-vote has left room for a centralizing trend in the validation process with a decreasing number of active solo miners and the formation of powerful coalitions of miners, *mining pool*, showing practical advantages but also motivating opportunistic pool-hopping behaviors [257]. Centralization in a permissionless environment results in increased vulnerability to double-spending attack. Decentralization is a characterizing feature for blockchain based cryptocurrency, one may argue that pool formation is nothing more than a converging trend to the original banking system [258]. An approach to face this monopoly trend is the inclusion of memory-access operations in the PoW computations accompanied by memory-bound functions. However, these schemes cannot make this centralization trend disappear since it requires specialized mining equipment and thus benefits from miners cooperation, as the original PoW (i.e., Litecoin [259; 260]).

Mining devices are constructed to compute hash values as fast as possible. The Bitcoin system was conceived for a CPU mining that was quickly replaced by a GPU (Graphic Processing Unit) mining. GPUs can perform hash computations in a more efficient way with respect to classical CPUs, therefore general Altcoins started adopting GPU mining at the end of 2010. This results in faster operations, due to operations parallelizing [261] and in energy savings [262]. When hardware based mining solutions took over the computing power dedicated in mining activities experienced, despite strong fluctuations, an exponential growth [263]. It worth nothing that alternative PoW-schemes try to compensate the incredible waste of energy with useful work at an academic level; *Primecoin* [264] searches for prime numbers chains (Cunningham chain [265]), *NooShare* [266] executes Monte-Carlo simulations, *Shoker* [267] proposes matrix-product problems to solve while in [268] authors propose to replace PoW hashing function with alternative one-way functions satisfying additional properties.

Pseudo-random leader elections based on PoW schemes [269] are generally prone to *grinding attacks*. The practice consists in testing several candidate blocks improving in this way the possibility of being a leader in the following round. Hence the need of unbiased unpredictable random elections as those adopted in [270; 271]. The need of alternative PoX schemes (*i*) motivating the proof of "useful" efforts and (*ii*) improving performance [272] in terms of security, scalability and eco-friendliness is evident.

.2.5.2 Proof-of-Stake and Virtual Mining Alternatives

The Proof-of-Stake (PoS) approach replaces the PoW leader election based on mining, with an alternative approach depending on users' investments in the blockchain, i.e., their stake: the amount of virtual tokens held by a user; in other words, the mining race costs are replaced by shares in the consensus. The probability of becoming a leader is proportional to one's stake; once a leader is selected among stake-holders, it has the right of validating the preferred block. As for PoW, consensus finality is not met and the "*richest chain*" rule breaks deadlock points – the valid chain is the one with the highest total amount of stake involved. Hence PoS could avoid the centralization trends observed with Bitcoin. PoS-type algorithms differ in the (*i*) estimate of users' holding and, in the (*ii*) adopted incentive mechanisms.

Users' stake can be estimated as an amount of coins stored in an account. However, security and fairness issues [186] arise when considering this consensus configuration: leader election components are quite predictable, and a selection based solely on the amount of tokens held by users is unfair ("rich-get-richer"). Hence, alternative solutions were proposed to elect the leader taking into account its stake.

One of the first PoS variations consists in weighting a coin stake by its "age" (i.e., the time elapsing between the last movement of the coin). In *PeerCoin* [273] the *coin age* has the same role of the computational power for the classical PoW scheme. However, the real difference is to give all participants the chance to be elected, thus solving monopoly-like situations. Despite stake-based coins (e.g., PeerCoin and *Nextcoin* [274]) prevent centralization trends, their underlying protocols encourage amassing coins and stay inactive in the network – that exposes the network to Sybil and DoS attacks [79]. Thus, the ideas to punish coins accumulation trends (*proof-of-stake-velocity* [275]) and to assign the reward for the validated blocks only to the active users (*proof-of-activity* [269]). Active peers are the ones that solve a crypto-puzzle with a difficulty target depending on the users' stake, thus hash computing improves network security. Leading stake-holders, responsible for block validation, are therefore picked in a pseudo-random fashion.

In both *Ouroboros* [271] and *Snow White* [270] participants use pseudo-random function to predict the block-generator however, while the former takes into account only the stake distribution in the network, the latter additionally relies on a pre-image (nonce) calculation. More precisely, Snow White is an "hybrid" protocol cleverly mixing PoW (computing only one hash

per round) and PoS (the hash should meet a target depending on user's stake). *Blackcoin* [276] and *Nova Coin* [277] are the first applications using this type of hybrid schemes (i.e. mixing different consensus mechanisms).

One of the latest variants of the PoS scheme was recently proposed by Ethereum. This is *Casper* [278] that is to be incorporated into the "Serenity" [279] version of the platform. Casper brings the PoS scheme closer to the traditional BFT model – more precisely, it combines the concepts of security deposits with voting in order to reach agreement. Peers have to make a security deposit in order to be elected as validating peers. The pseudo-random election takes into account the deposit entity made by the candidates and elect a set of validators. That is, Casper cannot be considered as an hybrid algorithm mixing PoS and BFT (see Section .2.7) since election and validation are not independent processes.

Concerning rewards distribution, PoS protocols originally distributed rewards among all peers regardless the elections results [270; 271] with the result of incentivizing the famous *nothing-at-stake* [100] attack. Today these naive implementations are overcame by valid alternatives: some [278; 280] asking validators to lock an amount of coins (*proof-of-deposit*), some [273] asking to destroy it (*proof-of-burn*), and some [281; 282; 283] asking to allocate a significant amount of memory/disk-space (*proof-of-capacity*) or to provide wireless network coverage (*proof-of-coverage*).

Efficient PoS alternatives based on virtual mining working for open-access blockchains with random leader election within untrusted nodes are the PoET (*proof-of-elapsed-time*) and the PoI (*proof-of-importance*) consensus schemes. The former adopts a trusted execution environment (TEE) in Intel SGX for the results verification [74] for guaranteeing both safety and randomness of the leader election. Peers make a request of *wait time* for processing the election procedure; the winner of the lottery is the validator with the shortest waiting. Correctness of the election can be publicly verified within the TEE: leaders generate a proof testifying they had the shortest wait time and additionally, they prove that the block broadcast happened right after the waiting expiration. The platform NEM (New Economic Movement [101]) proposes a blockchain based on a peculiar block validation process (i.e., *harvesting*) and a PoI [50] consensus algorithm determining the user that create and append transactions block (i.e., *harvester*). NEM works with an underlying cryptocurrency (i.e., XEM) that characterizing the balance of each account on the network that is split in a *vested* and an *unvested* part. Eligible validating peers are evaluated according to the amount of vested XEM and the support their accounts give to the network (i.e., number of transaction partners and number and size of transactions in the last 30 days). Contrary to previous mechanisms, PoI does not incentivize peers to save their coins/resources increasing their voting power. Harvester candidates are incentivezed to be 'active' in the network.

PoS enables both public and private leader election thus, the consensus protocol is applicable by both blockchain with and without permissions. Restricted elections result in DoS resilience since leader in the epoch become known to the stake-holder community at first and then to th public. Moreover, permissions on block validation may be assigned in order to improve

the efficiency of the system. That is, stakeholders privately delegate a representative set of validating peers (*delegated proof-of-stake* DPoS [102]). The list of witnesses is shuffled at the end of each round in such a way that each validator can produce block according to a certain rate. Witnesses are paid out for each produced block.

.2.6 BFT Algorithms

Traditional BFT protocols – resilient to both byzantine and crash failures – generally work under partial synchrony assumptions, bounded communication latency and a classical client-server architecture. Due to their nature (state machine replication protocols) properties of liveness and safety are guaranteed. Moreover, in BFT, both consensus *proposal* and consensus *decision* events are separated. The downside in these agreement protocol class is the communication complexity [284]. Hence, the necessity for closed-system adoption (i.e., permissioned blockchains).

The *Practical Byzantine Fault Tolerant* (PBFT) protocol [103] is a BFT variant that addresses the consensus problems for small systems, since agreement among n nodes is reached through the transmission of $O(n^2)$ messages; it does so relying on a three phase round division where in each round a block is validated passing through a *pre-prepared*, *prepared* and *commit* steps. Each peer proposal access to the next phase only with the $2/3$ network approval. Therefore, the algorithm requires at least $3f + 1$ honest replicas to tolerate f failing nodes. Recent PBFT variant *SIEVE* [285] introduce non-determinism in the chaincode execution handling transactions with occasionally different outputs. Moreover, an alternative PBFT-based consensus protocol recently proposed simplifies the traditional failure model for better efficiency levels. The idea behind *XFT* protocol [286] is to exploit the following assumption: *adversaries cannot control the majority of the nodes* $n > 2f$. In this way the crash fault tolerant protocol avoids considering byzantine failures.

With the arrival of consortium blockchains, the BFT protocol (popular in the financial sector) was amended to support open reading rights (public). *Stellar Consensus Protocol* (SCP [73]) is a BFT-variant based on permissions to choose a pool of known participants to trust. Participation to this pool (*quorum*) is open and global consensus is reached intersecting all the chosen quorums. In the same way, in *delegated BFT* protocols [287] only a class of representative peers comes to vote. The most popular BFT-open protocol adopting trusted subnetwork in the block validation process is *Ripple* [72]. It make use of *unique node lists* (UNLs) playing the same role as the Stellar quorums. The main characteristic of the protocol is that agreement is reached when the 80% of the nodes vote for the same candidate block, this result in low adversary power tolerance. The recent BFT variant, *proof-of-authority* (PoA) [288], relies on a set of trusted nodes (authorities) with a rotating leader. PoA algorithms [289; 290] ensure better performance with respect to PBFT consensus since working with less message exchanges (i.e., 1-2 message rounds to commit a block).

Classical BFT scalability drawbacks, regarding the number of nodes participating in the con-

sensus, have been solved with hybrid consensus protocols appropriately mixing PoX with BFT algorithms used in permissioned environments. This mix results in *committee* formation driving the consensus process replacing the original leader role. Hybrid models contemplate the usage of two different consensus procedures; one to form the committee and another one to establish consensus among the nodes inside the community. Note that, however, by "hybrid" we do not mean any committee-based consensus procedures (e.g., Hyperledger utilizes PBFT); hybrid algorithms are the ones mixing two different consensus schemes. In order to differentiate those hybrid schemes running classical BFT protocol – to the ones that make use only of PoX procedures – we denote them as *hybrid BFT-based* algorithms.

Nowadays, it is possible to find blockchains not requiring global consensus where each node has its own hash chain containing only the transactions where a user is involved. *Cong* proposed in [291] a system where agreement is established on special blocks representing a set of transactions. These systems can reach full *horizontal scalability* (i.e., scalability in the number of nodes) at the expense of robustness.

.2.7 Hybrid BFT-based Algorithms

Hybrid consensus mechanisms are born with the intent of preserving permissionless consensus but overcoming the trade-off between scalability and performance. Standard PoX consensus has to be improved by combining it with parts of BFT-based permissioned consensus mechanism. The idea of dividing the agreement process into different parts (see Section 2.2), initially proposed by private blockchains such as Hyperledger, is the key to built scalable permissionless protocols providing consensus finality. The assignment of tasks to the nodes is carried out by means of a committee-formation; consensus is driven by a community of nodes that build blocks at a first stage and then come to vote for their validity.

At first, the committee is formed, which then will agree on the validation of a block. Membership of the committee is open to all nodes in the blockchain; they acquire voting rights for the second phase through a PoX scheme. Existing hybrid algorithms involve PoW and PoS procedures to establish the leading nodes in the committee responsible for validating blocks. The idea of joining a committee through a PoW procedure is to assign voting power to each participant in proportion to their computational strength; this is the case of *ByzCoin* [292] and *PeerCensus* [293] where Bitcoin meet strong-consistency. Committee formation through PoX schemes is a dynamic process; participants receive a share of the committee through real or virtual mining. *Tendermint* [189] is the most popular protocol where Bitcoin PoW protocol is replaced with a PoS scheme that is, virtual mining. For Tendermint and other less known protocols [294; 295; 296] random committee selection is (can be) replaced by an assessment of the amount of tokens held by the blockchain nodes.

The right combination of PoX and BFT algorithms significantly improves the blockchain performance; however, scalability and throughput are not positively affected with a huge single-committee. Therefore, blockchains may adopt a consensus procedure based on multiple com-

mittee, also known as *sharding* [46]. In this way transactions can be processed in parallel by different *shards* (i.e., committees) of few nodes since their size is inversely proportional to the achieved performance level.

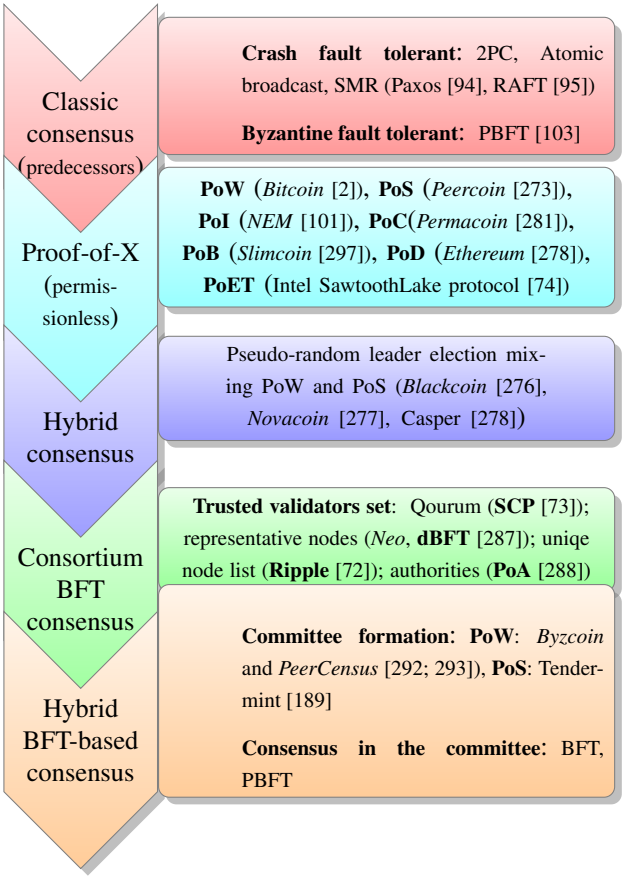

Figure 2: Evolutionary route of consensus protocols in five classes from pre-blockchain to post-blockchain protocols

.2.8 Summary of consensus mechanisms and their evolution

The diagram in Fig. 10 summarizes the evolution of the procedures to reach consensus in distributed systems, starting from the classic pre-blockchain algorithms - (*i*) Classic consensus - passing through the early blockchain consensus - (*ii*) Proof-of-X and (*iii*) Hybrid consensus - and, ending with the consortium solutions widely used today - (*iv*) Consortium BFT consensus and (*v*) Hybrid BFT-based. We have highlighted five main classes of consensus and characterized (where possible) the different variants. We consistently cite the main algorithms representing the consensus classes, encountered in the previous discussion.

ovo je appendix C

Bibliography

[1] D. Yaga *et al.*, "Blockchain technology overview," *Draft NISTIR*, vol. 8202, 2018.

[2] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," Oct. 2008, accessed: 2019-07-01. [online]: https://bitcoin.org/bitcoin.pdf.

[3] A. Baliga, "Understanding Blockchain Consensus Models," Persistent Systems – White paper, 2017, accessed: 2019-07-01. [online]: https://pdfs.semanticscholar.org/da8a/37b10bc1521a4d3de925d7ebc44bb606d740.pdf.

[4] Y. Guo and C. Liang, "Blockchain application and outlook in the banking industry," *Financial Innovation*, vol. 2, no. 1, p. 24, Dec. 2016. [Online]. Available: https://doi.org/10.1186/s40854-016-0034-9.

[5] K. Delmolino *et al.*, "Step by step towards creating a safe smart contract: Lessons and insights from a cryptocurrency lab," in *FC 2016*. Springer, pp. 79–94.

[6] J. Mendling *et al.*, "Blockchains for business process management-challenges and opportunities," *ACM Transactions on Management Information Systems (TMIS)*, vol. 9, no. 1, p. 4, 2018.

[7] "Introduction to smart contracts." accessed: 2019-07-01. [online]: http://solidity.readthedocs.io/en/latest/introduction-to-smart-contracts.html.

[8] V. Buterin, "Ethereum White-paper," accessed: 2019-07-01. [online]: https:github.com/ethereum/wiki/wiki/White-Paper.

[9] G. Greenspan, "Blockchains vs centralized databases," accessed: 2019-07-01. [online]: http://www.multichain.com/blog/2016/03/blockchains-vs-centralized-databases,.

[10] H. Sukhwani *et al.*, "Performance modeling of pbft consensus process for permissioned blockchain network (hyperledger fabric)," in *IEEE SRDS 2017*, Sept, pp. 253–255.

[11] "Hyperledger Architecture, Volume 1, Introduction to Hyperledger Business Blockchain Design Philosophy and Consensus," accessed: 2019-07-01. [online]: https://www.hyperledger.org/wp-content/uploads/2017/08/HyperLedger_Arch_WG_Paper_1_Consensus.pdf.

[12] "IBM Blockchain Platform," accessed: 2019-07-01. [online]: https://www.ibm.com/blogs/blockchain/2017/08/your-guide-to-the-ibm-blockchain-platform-announcement.

[13] K. Croman *et al.*, "On scaling decentralized blockchains," in *FC 2016*, pp. 106–125.

[14] R. Davenport, "Distributed database technology−a survey," *Computer Networks*, vol. 2, no. 3, pp. 155–167, 1978.

[15] R. Schollmeier, "A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications," in *P2P'01*, 2001.

[16] S. Goyal, "Centralized vs Decentralized vs Distributed," accessed: 2019-07-01. [online]: https://medium.com.

[17] "Codementor," accessed: 2019-07-01. [online]: https://www.codementor.io.

[18] Appinventive, "Blockchain App Development Cost," accessed: 2019-07-01. [online]: https://appinventiv.com/blockchain-app-development-cost.

[19] M. Conti *et al.*, "A Survey on Security and Privacy Issues of Bitcoin," *IEEE Communications Surveys Tutorials*, pp. 1–1, 2018.

[20] D. Bradbury, "The problem with Bitcoin," *Computer Fraud & Security*, vol. 2013, no. 11, pp. 5–8, 2013.

[21] I. Eyal *et al.*, "Bitcoin-NG: A Scalable Blockchain Protocol." in *USENIX NSDI 2016*.

[22] "What to Mine," accessed: 2019-07-01. [online]: https://whattomine.com/coins.

[23] A. Wisniewska, "Altcoins," institute of Economic Research, Working Paper. Accessed: 2019-07-01. [online]: https://ideas.repec.org/p/pes/wpaper/2016no14.html.

[24] M. Khalilov and A. Levi, "A Survey on Anonymity and Privacy in Bitcoin-like Digital Cash Systems," *IEEE Communications Surveys & Tutorials*, 2018.

[25] S. Popov, "The Tangle," iOTA White paper. Accessed: 2019-07-01. [online]: https://www.iota.org/research/academic-papers.

[26] "BigchainDB: a scalable blockchain database (White paper)," 2016, accessed: 2018-12-02. [online]: https://www.bigchaindb.com/whitepaper.

[27] K. Saur *et al.*, "Technology for secure partitioning and updating of a distributed digital ledger," uS Patent Application Publication, Intel Corporation, CA (USA), 2016-11-18, US20180145836A1.

[28] A. M. Antonopoulos, *Mastering Bitcoin, unlocking digital cryptocurrencies*. O'reilly Media, 2014.

[29] S. Haber and W. Stornetta, "How to time-stamp a digital document," in *Conference on the Theory and Application of Cryptography*. Springer, 1990, pp. 437–455.

[30] R. Merkle, "Digital signature system and method based on a conventional encryption function," Nov. 14, 1989, uS Patent 4,881,264.

[31] D. Conte de Leon *et al.*, "Blockchain: properties and misconceptions," *Asia Pacific Journal of Innovation and Entrepreneurship*, vol. 11, no. 3, pp. 286–300, 2017.

[32] Q. DuPont, "Experiments in algorithmic governance: A history and ethnography of "the dao", a failed decentralized autonomous organization," in *Bitcoin and Beyond*. Routledge, 2017, pp. 157–177.

[33] V. Buterin, "On public and private blockchains," accessed: 2019-07-01. [online]: https://blog.ethereum.org/2015/08/07/on-public-and-private-blockchains.

[34] K. Wüst and A. Gervais, "Do you need a blockchain?" *IACR Cryptology ePrint Archive*, vol. 2017, p. 375, 2017.

[35] R. Brown *et al.*, "Corda: An Introduction, White paper," accessed: 2019-07-01. [online]: https://docs.corda.net/head/_static/corda-introductory-whitepaper.pdf.

[36] "Hyperledger Architecture Vol.1, Introduction to Hyperledger Business Blockchain Design Philosophy and Consensus," accessed: 2019-07-01. [online]: https://www.hyperledger.org/wp-content/uploads/2017/08/Hyperledger_Arch_WG_Paper_1_Consensus.pdf.

[37] M. Walport, "Distributed ledger technology: beyond blockchain," UK Government Office for Science, Tech. Rep., 2016, accessed: 2019-07-01. [online]: https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/492972/gs-16-1-distributed-ledger-technology.pdf.

[38] F. Tschorsch and B. Scheuermann, "Bitcoin and Beyond: A Technical Survey on Decentralized Digital Currencies," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 3, pp. 2084–2123, 2016.

[39] T. Neudecker and H. Hartenstein, "Network layer aspects of permissionless blockchains," *IEEE Communications Surveys & Tutorials*, 2018.

[40] U. Mukhopadhyay *et al.*, "A brief survey of Cryptocurrency systems," in *PST 2016*.

[41] L. Sankar, M. Sindhu, and M. Sethumadhavan, "Survey of consensus protocols on blockchain applications," in *ICACCS 2017*.

[42] J. Garay, A. Kiayias, and N. Leonardos, "The bitcoin backbone protocol: Analysis and applications," in *EUROCRYPT 2015*.

[43] Z. Zheng *et al.*, "Blockchain challenges and opportunities: A survey," *Working Paper*, 2016.

[44] ——, "An Overview of Blockchain Technology: Architecture, Consensus, and Future Trends," in *IEEE BigData Congress*, 2017.

[45] C. Cachin and M. Vukolić, "Blockchains Consensus Protocols in the Wild," *arXiv preprint arXiv:1707.01873*, 2017.

[46] S. Bano *et al.*, "Consensus in the Age of Blockchains," *arXiv preprint arXiv:1711.03936*, 2017.

[47] W. Wang *et al.*, "A survey on consensus mechanisms and mining management in blockchain networks," *arXiv preprint arXiv:1805.02707*, 2018.

[48] X. Li *et al.*, "A survey on the security of blockchain systems," *Future Generation ComputerSystems*, 2017.

[49] I. Lin and T. Liao, "A Survey of Blockchain Security Issues and Challenges." *IJ Network Security*, vol. 19, no. 5, pp. 653–659, 2017.

[50] "Survey on Blockchain Technologies and Related Services," FY2015 Technical report – Nomura Research Institute, accessed: 2019-07-01. [online]: http://www.meti.go.jp/english/press/2016/pdf/0531_01f.pdf.

[51] M. A. Ferrag *et al.*, "Blockchain Technologies for the Internet of Things: Research Issues and Challenges," *arXiv preprint arXiv:1806.09099*, 2018.

[52] N. Bozic, G. Pujolle, and S. Secci, "A tutorial on blockchain and applications to secure network controlplanes," in *SCNS 2016*, pp. 1–8.

[53] W. Stallings, "A Blockchain Tutorial," *The Internet Protocol Journal*, vol. 20, no. 3, pp. 2–24, 2017.

[54] T. Koens and E. Poll, "What blockchain alternative do you need?" in *Data Privacy Management, Cryptocurrencies and Blockchain Technology*. Springer International Publishing, 2018, pp. 113–129.

[55] "Bitcoin Wiki. Transaction," accessed: 2019-07-01. [online]: https://en.bitcoin.it/wiki/Transaction.

[56] J. Willet, "The Second Bitcoin Whitepaper, V. 0.5," [online]: https://bravenewcoin.com/insights/the-second-bitcoin-whitepaper-vs--0-5.

[57] J. Bonneau *et al.*, "Perspectives on Bitcoin and second-generation cryptocurrencies," working Paper. Accessed: 2019-07-01. [online]: https://www.semanticscholar.org/paper/perspectives-on-Bitcoin-and-second-generation-Bonneau-Miller/3cf586a2bbdcb9bf9860b6ddf952e3a038d51811.

[58] D. MacGregor, D. Mothersole, and J. Zolnowsky, "Method and apparatus for a compare and swap instruction," Apr. 22, 1986, uS Patent number 4,584,640, NXP USA Inc.

[59] N. Atzei *et al.*, "SoK: unraveling Bitcoin smart contracts," in *POST 2018*. Springer.

[60] G. Andresen, "BIP 16: Pay to script hash," 2012, accessed: 2019-07-01. [online]: https://github.com/bitcoin/bips/blob/master/bip-0016.mediawiki.

[61] "Bitcoin Wiki. Nonce," accessed: 2019-07-01. [online]: https://en.bitcoin.it/wiki/Nonce.

[62] "Bitcoin Wiki. Script," accessed: 2019-07-01. [online]: https://en.bitcoin.it/wiki/Script.

[63] "Chain Protocol - White Paper," accessed: 2019-07-01. [online]: https://chain.com/docs/1.2/protocol/papers/whitepaper.

[64] P. Dai *et al.*, "Smart-Contract Value-Transfer Protocols on a Distributed Mobile Application Platform - White paper," 2017.

[65] E. Androulaki *et al.*, "Hyperledger fabric: A distributed operating system for permissioned blockchains," *arXiv preprint arXiv:1801.10228*, 2018.

[66] S. Popejoy, "The Pact Smart-Contract Language. White paper," accessed: 2019-07-01. [online]: http://kadena.io/docs/Kadena-PactWhitepaper.pdf.

[67] C. Lee, "Litecoin-open source p2p digital currency," accessed: 2019-07-01. [online]: https://github.com/coblee.

[68] D. Kuhnert, "The Dogecoin survival guide," accessed: 2019-07-01. [online]: https://imgur.com/a/Sgyox.

[69] M. Green and I. Miers, "Bolt: Anonymous payment channels for decentralized currencies," in *ACM CCS 2017*.

[70] G. Greenspan, "MultiChain Private Blockchain – White Paper," accessed: 2019-07-01. [online]: https://www.multichain.com/download/MultiChain-White-Paper.pdf.

[71] L. Goodman, "Tezos – A self-amending crypto-ledger, White paper," accessed: 2019-07-01. [online]: https://tezos.com/static/papers/white_paper.pdf.

[72] D. Schwartz *et al.*, "The Ripple protocol consensus algorithm – White Paper," accessed: 2019-07-01. [online]: https://ripple.com/files/ripple_consensus_whitepaper.pdf.

[73] D. Mazieres, "The stellar consensus protocol: A federated model for internet-level consensus - White paper," accessed: 2019-07-01. [online]: https://www.stellar.org/papers/stellar-consensus-protocol.pdf.

[74] K. Olson *et al.*, "Sawtooth: An Introduction, White paper," accessed: 2019-07-01. [online]: https://www.hyperledger.org/wp-content/uploads/2018/01/Hyperledger_Sawtooth_WhitePaper.pdf.

[75] A. Demers *et al.*, "Epidemic algorithms for replicated database maintenance," in *ACM PODC 1987*, pp. 1–12.

[76] B. Wiki, "Weaknesses-Denial of Service (DoS)," accessed: 2019-07-01. [online]: https://en.bitcoin.it/wiki/Weaknesses.

[77] E. Heilman *et al.*, "Eclipse Attacks on Bitcoin's Peer-to-Peer Network," in *USENIX Security Symposium*, 2015.

[78] G. Fanti and P. Viswanath, "Deanonymization in the Bitcoin P2P Network," in *NIPS 2017*, pp. 1364–1373. [Online]. Available: http://papers.nips.cc/paper/6735-deanonymization-in-the-bitcoin-p2p-network.pdf

[79] M. Babaioff *et al.*, "On bitcoin and red balloons," in *ACM EC 2012*.

[80] J. Göbel *et al.*, "Bitcoin blockchain dynamics: The selfish-mine strategy in the presence of propagation delay," *Performance Evaluation*, vol. 104, pp. 23–41, 2016.

[81] A. Gervais *et al.*, "On the security and performance of proof of work blockchains," in *ACM CCS 2016*, pp. 3–16.

[82] L. Lamport, R. Shostak, and M. Pease, "The byzantine generals problem," *ACM Transactions on Programming Languages and Systems*, vol. 4, no. 3, pp. 382–401, 1982.

[83] T. Swanson, "Consensus-as-a-service: a brief report on the emergence of permissioned, distributed ledger systems," R3 Technical Report, Apr. 2015, accessed: 2019-07-01. [online]: https://allquantor.at/blockchainbib/pdf/swanson2015consensus.pdf.

[84] S. Kiyomoto, M. Rahman, and A. Basu, "On blockchain-based anonymized dataset distribution platform," in *IEEE SERA 2017*, June, pp. 85–92.

[85] "A Next-Generation Smart Contract and Decentralized Application Platform, Ethereum white paper," accessed: 2019-07-01. [online]: https://github.com/ethereum/wiki/wiki/White-Paper.

[86] C. Cachin, "Architecture of the Hyperledger blockchain Fabric," in *DCCL 2016*.

[87] M. Hearn, "Corda: A distributed ledger," white Paper, Accessed: 2019-07-01. [online]: https://docs.corda.net/head/_static/corda-technical-whitepaper.pdf.

[88] "Cosmos: A Network of Distributed Ledgers," accessed: 2019-07-01. [online]: https://cosmos.network/cosmos-whitepaper.pdf.

[89] "Quorum White paper," accessed: 2019-07-01. [online]: https://github.com/jpmorganchase/quorum-docs/blob/master/Quorum%20Whitepaper%20v0.1.pdf.

[90] W. Ren, R. Beard, and E. Atkins, "A survey of consensus problems in multi-agent coordination," in *ACC 2005*, vol. 3, June, pp. 1859–1864.

[91] N. Lynch, *Distributed Algorithms*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1996.

[92] J. Gray, *Notes on data base operating systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1978, pp. 393–481.

[93] F. Schneider, "Implementing Fault-tolerant Services Using the State Machine Approach: A Tutorial," *ACM Comput. Surv.*, vol. 22, no. 4, pp. 299–319, 1990.

[94] L. Lamport *et al.*, "Paxos made simple," *ACM Sigact News*, vol. 32, no. 4, pp. 18–25, 2001.

[95] D. Ongaro and J. Ousterhout, "In search of an understandable consensus algorithm," in *USENIX Annual Technical Conference*.

[96] M. Pease, R. Shostak, and L. Lamport, "Reaching Agreement in the Presence of Faults," *J. ACM*, vol. 27, no. 2, pp. 228–234, 1980.

[97] R. Baldoni *et al.*, "Unconscious Eventual Consistency with Gossips," in *Symposium on Self-Stabilizing Systems*, 2006, pp. 65–81.

[98] B. Wiki, "Hashcash," accessed: 2019-07-01. [online]: https://en.bitcoin.it/wiki/Hashcash.

[99] J. Aspnes, C. Jackson, and A. Krishnamurthy, "Exposing computationally-challenged Byzantine impostors," TYALEU/DCS/TR-1332, Yale University, Tech. Rep., 2005, accessed: 2019-07-01. [online]: http://www.cs.yale.edu/homes/aspnes/papers/tr1332.pdf.

[100] W. Li *et al.*, "Securing proof-of-stake blockchain protocols," in *Data Privacy Management, Cryptocurrencies and Blockchain Technology*. Springer, 2017, pp. 297–315.

[101] "NEM - Technical Reference, NEM, Version 1.2.1," Tech. Rep., Feb 2018, accessed: 2019-07-01. [online]: https://nem.io/wp-content/themes/nem/files/NEM_techRef.pdf.

[102] D. Larimer, "Delegated proof-of-stake white paper," accessed: 2019-07-01. [online]: https://steemit.com/dpos/@dantheman/dpos-consensus-algorithm-this-missing-white-paper.

[103] M. Castro and B. Liskov, "Practical byzantine fault tolerance," in *OSDI 1999*.

[104] M. Vukolić, "The quest for scalable blockchain fabric: Proof-of-work vs. BFT replication," in *Int. Workshop on Open Problems in Network Security*, 2015.

[105] L. Bach, B. Mihaljevic, and M. Zagar, "Comparative analysis of blockchain consensus algorithms," in *IEEE MIPRO 2018*.

[106] K. Fanning and D. Centers, "Blockchain and its coming impact on financial services," *Journal of Corporate Accounting & Finance*, vol. 27, no. 5, pp. 53–57, 2016.

[107] B. Maurer, "Re-risking in realtime: on possible futures for finance after the blockchain," *BEHEMOTH – A Journal on Civilisation*, vol. 9, no. 2, pp. 82–96, 2016.

[108] O. Bussmann, "The future of finance: Fintech, tech disruption, and orchestrating innovation," in *Equity Markets in Transition*. Springer, 2017, pp. 473–486.

[109] A. Spielman, "Blockchain: digitally rebuilding the real estate industry," Ph.D. dissertation, Massachusetts Institute of Technology, 2016.

[110] G. Zyskind *et al.*, "Decentralizing privacy: Using blockchain to protect personal data," in *2015 IEEE – Security and Privacy Workshops*, pp. 180–184.

[111] J. Mattila *et al.*, "Industrial Blockchain Platforms: An Exercise in Use Case Development in the Energy Industry," *ETLA Working Papers*, no. 43, 2016.

[112] F. Imbault *et al.*, "The green blockchain: Managing decentralized energy production and consumption," in *IEEE EEEIC/ICPS 2017*, June, pp. 1–5.

[113] E. Münsing, J. Mather, and S. Moura, "Blockchains for decentralized optimization of energy resources in microgrid networks," in *IEEE CCTA 2017*, Aug, pp. 2164–2171.

[114] N. Witchey, "Healthcare transaction validation via blockchain proof-of-work, systems and methods," May 13, 2015, uS Patent App. 14/711,740.

[115] X. Yue *et al.*, "Healthcare Data Gateways: Found Healthcare Intelligence on Blockchain with Novel Privacy Risk Control," *Journal of Medical Systems*, vol. 40, no. 10, p. 218, Aug. 2016.

[116] L. Linn and M. Koo, "Blockchain for health data and its potential use in health it and health care related research," in *ONC/NIST Use of Blockchain for Healthcare and Research Workshop*, 2016.

[117] A. Ekblaw *et al.*, "A Case Study for Blockchain in Healthcare: "MedRec" prototype for electronic health records and medical research data," in *IEEE Open & Big Data Conference*, 2016.

[118] M. Mettler, "Blockchain technology in healthcare: The revolution starts here," in *IEEE Healthcom 2016*, pp. 1–3.

[119] C. Broderson *et al.*, "Blockchain: Securing a New Health Interoperability Experience," *Accenture, Working paper*, 2016.

[120] K. Peterson *et al.*, "A Blockchain-Based Approach to Health Information Exchange Networks," *Working paper*, 2016.

[121] U. Sharma, "Blockchain in healthcare: Patient benefits and more," accessed: 2019-07-01. [online]: https://www.ibm.com/blogs/blockchain/2017/10/blockchain-in-healthcare-patient-benefits-and-more.

[122] M. Orcutt, "Who Will Build the Health-Care Blockchain?" MIT Technology Review. Accessed: 2019-07-01. [online]: https://www.technologyreview.com/s/60882/who-will-build-the-health-care-blockchain.

[123] S. Huh *et al.*, "Managing IoT devices using blockchain platform," in *ICACT 2017*.

[124] M. Samaniego and R. Deters, "Blockchain as a service for iot," in *IEEE iThings/GreenCom/CPSCom/Smart Data 2016*.

[125] D. Kravitz and J. Cooper, "Securing user identity and transactions symbiotically: IoT meets blockchain," in *GIoTS 2017*, June, pp. 1–6.

[126] K. Özyilmaz and A. Yurdakul, "Work-in-progress: integrating low-power iot devices to a blockchain-based infrastructure," in *EMSOFT 2017*.

[127] A. Hari and T. Lakshman, "The Internet Blockchain: A Distributed, Tamper-Resistant Transaction Framework for the Internet," in *ACM HotNets*, 2016, pp. 204–210.

[128] N. Bozic, G. Pujolle, and S. Secci, "Securing virtual machine orchestration with blockchains," in *CSNet 2017*, Oct, pp. 1–8.

[129] I. D. Alvarenga, G. A. F. Rebello, and O. C. M. B. Duarte, "Securing configuration management and migration of virtual network functions using blockchain," in *NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium*, April 2018, pp. 1–9.

[130] D. Tse *et al.*, "Blockchain application in food supply information security," in *IEEE IEEM 2017*.

[131] F. Tian, "An agri-food supply chain traceability system for china based on rfid blockchain technology," *ICSSSM*, 2016.

[132] M. Caro *et al.*, "Blockchain-based traceability in agri-food supply chain management: A practical implementation," in *IoT Vertical and Topical Summit on Agriculture-Tuscany*. IEEE, 2018.

[133] Y. Yuan and F. Wang, "Towards blockchain-based intelligent transportation systems," in *IEEE ITSC 2016*.

[134] L. Li *et al.*, "Creditcoin: A privacy-preserving blockchain-based incentive announcement network for communications of smart vehicles," *IEEE Transactions on Intelligent Transportation Systems*, vol. 19, no. 7, pp. 2204–2220, 2018.

[135] K. Wüst and A. Gervais, "Do you need a blockchain?" in *2018 Crypto Valley Conference on Blockchain Technology (CVCBT)*. IEEE, 2018, pp. 45–54.

[136] M. Vukolić, "Rethinking Permissioned Blockchains," in *ACM Workshop on Blockchain, Cryptocurrencies and Contracts*, 2017, pp. 3–7.

[137] S. Pongnumkul *et al.*, "Performance Analysis of Private Blockchain Platforms in Varying Workloads," in *ICCCN 2017*, July, pp. 1–6.

[138] G. Greenspan, "Private blockchains are more than just shared databases," accessed: 2019-07-01. [online]: https://www.multichain.com/blog/2015/10/private-blockchains-shared-databases.

[139] S. Ray, "Blockchains versus traditional databases," accessed: 2019-07-01. [online]: https://hackernoon.com/blockchains-versus-traditional-databases-c1a728159f79.

[140] A. Narayanan, "Private blockchain is just a confusing name for a shared database," accessed: 2019-07-01. [online]: https://freedom-to-tinker.com/2015/09/18/private-blockchain-is-just-a-confusing-name-for-a-shared-database.

[141] C. Clack, V. Bakshi, and L. Braine, "Smart contract templates: foundations, design landscape and research directions," *arXiv preprint arXiv:1608.00771*, 2016.

[142] X. Xu *et al.*, "A taxonomy of blockchain-based systems for architecture design," in *IEEE ICSA 2017*, pp. 243–252.

[143] "MultiChain: open platform for building blockchains," accessed: 2019-07-01. [online]: https://www.multichain.com.

[144] "Swarm," accessed: 2019-07-01. [online]: https://swarm-guide.readthedocs.io/en/latest/introduction.html.

[145] P. Labs, "Filecoin: A Decentralized Storage Network (White paper)," 2016, accessed: 2019-01-22. [online]: https://filecoin.io/filecoin.pdf.

[146] "Microsoft BaaS," accessed: 2019-07-01. [online]: https://azure.microsoft.com/en/solutions/blockchain.

[147] "Types of tokens: the four mistakes beginner crypto-investors make," *ICO Scoring*, Accessed: 2019-07-01. [online]: https://medium.com/swlh/types-of-tokens-the-four-mistakes-beginner-crypto-investors-make-a76b53be5406.

[148] X. Xu *et al.*, "The blockchain as a software connector," in *IEEE/IFIP WICSA 2016*.

[149] A. Back *et al.*, "Enabling blockchain innovations with pegged sidechains," accessed: 2019-07-01. [online]: http://opensciencereview.com/papers/123/enablingblockchain-innovations-with-pegged-sidechains.

[150] M. Valenta and P. Sandner, "Comparison of Ethereum, Hyperledger Fabric and Corda," FSBC Working Paper, 2017, accessed: 2019-07-01. [online]: http://explore-ip.com/2017_Comparison-of-Ethereum-Hyperledger-Corda.pdf.

[151] A. Ellervee, R. Matulevicius, and N. Mayer, "A Comprehensive Reference Model for Blockchain-based Distributed Ledger Technology," in *ER Forum 2017*.

[152] T. Dinh *et al.*, "Untangling Blockchain: A Data Processing View of Blockchain Systems," *arXiv preprint arXiv:1708.05665*, 2017.

[153] "Bitcoin source code," accessed: 2019-07-01. [online]: https://github.com/bitcoin/bitcoin.

[154] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," Technical report, accessed: 2019-07-01. [online]: https://ethereum.github.io/yellowpaper/paper.pdf.

[155] B. Charron-Bost, F. Pedone, and A. Schiper, "Replication: Theory and Practice," Lecture Notes in Computer Science, 978-3-642-11294-2, Springer.

[156] "The RLPx Transport Protocol," accessed: 2019-07-01. [online]: https://github.com/ethereum/devp2p/blob/master/rlpx.md.

[157] T. Dinh *et al.*, "Blockbench: A framework for analyzing private blockchains," in *ACM SIGMODS/PODS 2017*.

[158] "Ethereum source code," accessed: 2019-07-01. [online]: https://github.com/ethereum.

[159] "Hyperledger Architecture, Vol. 2," accessed: 2019-07-01. [online]: https://www.hyperledger.org/wp-content/uploads/2018/04/Hyperledger_Arch_WG_Paper_2_SmartContracts.pdf.

[160] J. Sousa, A. Bessani, and M. Vukolić, "A byzantine fault-tolerant ordering service for the hyperledger fabric blockchain platform," *arXiv preprint arXiv:1709.06921*, 2017.

[161] "What is gRPC," accessed: 2019-07-01. [online]: https://grpc.io/docs/guides.

[162] "LevelDB key/value database in Go," accessed: 2019-07-01. [online]: https://github.com/syndtr/goleveldb.

[163] "Apache couchdb," accessed: 2019-07-01. [online]: http://couchdb.apache.org.

[164] V. Dhillon, D. Metcalf, and M. Hooper, "The Hyperledger Project," in *Blockchain Enabled Applications*. Springer, 2017, pp. 139–149.

[165] F. Muratov *et al.*, "YAC: BFT Consensus Algorithm for Blockchain," *arXiv preprint arXiv:1809.00554*, 2018.

[166] F. McKeen *et al.*, "Intel® software guard extensions (Intel® sgx) support for dynamic memory management inside an enclave," in *ACM HASP 2016*, p. 10.

[167] "Hyperledger Sawtooth source code," accessed: 2019-07-01. [online]: https://github.com/hyperledger/sawtooth-core.

[168] "Sovrin: A Protocol and Token for Self-Sovereign Identity and Decentralized Trust," White Paper, Sovrin Foundation, Version 1.0. Accessed: 2019-07-01. [online]: https:sovrin.org/library/sovrin-protocol-and-token-white-paper.

[169] "Hyperledger Improvement Proposal - Hyperledger Burrow," Accessed: 2019-07-01. [online]: https://www.hyperledger.org/wp-content/uploads/2017/06/HIP_Burrowv2.pdf.

[170] "Hyperledger Burrow source code," accessed: 2019-07-01. [online]: https://github.com/hyperledger/burrow.

[171] "Hyperledger Grid," accessed: 2019-07-01. [online]: https://www.hyperledger.org/projects/grid.

[172] "Welcome to Hyperledger Cello," accessed: 2019-07-01. [online]: http://hyperledger-cello.readthedocs.io/en/latest.

[173] "Hyperledger Explorer," accessed: 2019-07-01. [online]: https://www.hyperledger.org/projects/explorer.

[174] "Hyperledger Composer - An Overview," Accessed: 2019-07-01. [online]: https://www.hyperledger.org/wp-content/uploads/2017/05/Hyperledger-Composer-Overview.pdf.

[175] "Hyperledger CALIPER," accessed: 2019-07-01. [online]: https://www.hyperledger.org/projects/caliper..

[176] "Hyperledger QUILT," accessed: 2019-07-01. [online]: https://www.hyperledger.org/projects/quilt.

[177] "Interledger Protocol (ILP)," accessed: 2019-07-01. [online]: https://interledger.org/rfcs/0003-interledger-protocol.

[178] "Hyperledger Aries," accessed: 2019-07-01. [online]: https://www.hyperledger.org/projects/aries.

[179] "Hyperledger Ursa," accessed: 2019-07-01. [online]: https://www.hyperledger.org/projects/ursa.

[180] "Transact Hyperledger project," accessed: 2019-07-01. [online]: https://www.hyperledger.org/projects/transact.

[181] A. Kundu and E. Bertino, "On Hashing Graphs," *IACR Cryptology ePrint Archive*, vol. 2012, p. 352, 2012.

[182] A. Bessani, J. Sousa, and E. Alchieri, "State Machine Replication for the Masses with BFT-SMART," in *IEEE/IFIP DSN 2014*, June, pp. 355–362.

[183] J. Carlyle, "Corda Performance To infinity and beyond," Technical report, accessed: 2019-07-01. [online]: https://www.r3.com/wp-content/uploads/2018/04/Corda-Performance-ENG.pdf.

[184] "Corda source code," accessed: 2019-07-01. [online]: https://github.com/corda.

[185] J. Kwon, "Tendermint: Consensus without mining," techical report Accessed: 2019-07-01. [online]: https://cdn.relayto.com/media/files/LPgoWO18TCeMIggJVakt_tendermint.pdf.

[186] Y. Amoussou-Guenou *et al.*, "Correctness and Fairness of Tendermint-core Blockchains," *arXiv preprint arXiv:1805.08429*, 2018.

[187] G. Veronese *et al.*, "Spin one's wheels? Byzantine fault tolerance with a spinning primary," in *IEEE SRDS 2009*, pp. 135–144.

[188] J. Yin *et al.*, "Separating agreement from execution for byzantine fault tolerant services," *ACM SIGOPS Operating Systems Review*, vol. 37, no. 5, pp. 253–267, 2003.

[189] E. Buchman, "Tendermint: Byzantine fault tolerance in the age of blockchains," Ph.D. dissertation, University of Guelph, 2016.

[190] "Tendermint source code," accessed: 2019-07-01. [online]: https://github.com/tendermint.

[191] "Chain Core source code," accessed: 2019-07-01. [online]: https://github.com/chain/chain.

[192] "Chain news – Introducing Sequence," accessed: 2019-07-01. [online]: https://blog.chain.com/introducing-sequence-e14ff70b730.

[193] B. Glickstein *et al.*, *TxVM White paper. A New Design for Blockchain Transactions*, accessed: 2019-07-01. [online]: https://github.com/chain/txvm.

[194] "Raft etcd," accessed: 2019-07-01. [online]: https://github.com/coreos/etcd/tree/master/raft.

[195] "Quorum source code," accessed: 2019-07-01. [online]: https://github.com/jpmorganchase/quorum.

[196] "Hyperledger Fabric source code," accessed: 2019-07-01. [online]: https://github.com/hyperledger/fabric.

[197] "Hyperledger Indy source code," accessed: 2019-07-01. [online]: https://github.com/hyperledger/indy-sdk.

[198] "Hyperledger Iroha source code," accessed: 2019-07-01. [online]: https://github.com/hyperledger/iroha.

[199] "Hyperledger Blockchain Performance Metrics (White paper)," 2018, accessed: 2018-12-02. [online]: https://www.hyperledger.org/wp-content/uploads/2018/10/HL_Whitepaper_Metrics_PDF_V1.01.pdf.

[200] T. Dinh *et al.*, "M2R: Enabling Stronger Privacy in MapReduce Computation," in *USENIX Security Symposium*, 2015, pp. 447–462.

[201] J. Winter, "Trusted computing building blocks for embedded linux-based ARM trustzone platforms," in *ACM STC 2008*, pp. 21–30.

[202] H. Dang *et al.*, "Chain of Trust: Can Trusted Hardware Help Scaling Blockchains?" *arXiv preprint arXiv:1804.00399*, 2018.

[203] "IBM BaaS," accessed: 2019-07-01. [online]: https://www.ibm.com/blockchain.

[204] "SAP BaaS," accessed: 2019-07-01. [online]: https://www.sap.com/products/leonardo/blockchain.html.

[205] "HPE BaaS," accessed: 2019-07-01. [online]: https://www.hpe.com/us/en/solutions/blockchain.html.

[206] "Oracle Blockchain Cloud service," accessed: 2018-08-02. [online]: https://cloud.oracle.com/opc/paas/ebooks/Oracle_Blockchain_Cloud_Service.pdf.

[207] "Amazon BaaS," accessed: 2019-07-01. [online]: https://aws.amazon.com/partners/blockchain.

[208] "Huawei Blockchain Whitepaper," accessed: 2019-07-01. [online]: https://static.huaweicloud.com/upload/files/pdf/20180416/20180416142450_61761.pdf.

[209] "Vechain Whitepaper," accessed: 2019-07-01. [online]: https://cdn.vechain.com/vechainthor_development_plan_and_whitepaper_en_v1.0.pdf.

[210] "Blocko," accessed: 2019-07-01. [online]: https://www.blocko.io.

[211] "Baidu," accessed: 2019-07-01. [online]: https://chain.baidu.com.

[212] J. Benet, "IPFS - Content Addressed, Versioned, P2P File System (DRAFT 3)," accessed: 2019-07-01. [online]: https://ipfs.io.

[213] M. Deimel *et al.*, "Transparency in food supply chains: empirical results from german pig and dairy production," *Journal on Chain and Network Science*, 2008.

[214] "IBM Food Trust," accessed: 2019-07-01. [online]: https://www.ibm.com/blockchain/solutions/food-trust.

[215] S. Takagi *et al.*, "Blockchain-Based Digital Currencies for Community Building," discussion paper No.6 (17-004). Accessed: 2019-07-01. [online]: http://www.glocom.ac.jp/discussionpaper/dp06.

[216] M. Belotti, N. Božić, G. Pujolle, and S. Secci, "A vademecum on blockchain technologies: When, which and how," *IEEE Communications Surveys Tutorials*, pp. 1–1, 2019.

[217] N. Bozic, S. Secci, and G. Pujolle, "Systeme et procede d'authentification d'instructions de gestion de ressources informatiques d'un serveur de virtualisation," french Patent Application Publication, 2019-03-08, FR3070778.

[218] R. Mijumbi *et al.*, "Network function virtualization: State-of-the-art and research challenges," *IEEE Communications Surveys Tutorials*, vol. 18, pp. 236–262, 2016.

[219] T. Wang, Z. Su, Y. Xia, and M. Hamdi, "Rethinking the data center networking: Architecture, network protocols, and resource sharing," *IEEE Access*, vol. 2, pp. 1481–1496, 2014.

[220] C. E. Leiserson, "Fat-trees: Universal networks for hardware-efficient supercomputing," *IEEE Transactions on Computers*, vol. C-34, no. 10, pp. 892–901, Oct 1985.

[221] D. Agrawal and L. Bhuyan, "Generalized hypercube and hyperbus structures for a computer network," *IEEE Transactions on Computers*, vol. 33, no. 04, pp. 323–333, apr 1984.

[222] D. Johnson and A. Menezes, "Elliptic curve DSA (ECDSA): an enhanced DSA," in *USENIX Security Symposium*, vol. 7, 1998.

[223] D. Johnson, A. Menezes, and S. Vanstone, "The elliptic curve digital signature algorithm (ECDSA)," *Int. Journal of Information Security*, vol. 1, no. 1, pp. 36–63, 2001.

[224] S. Luan and V. Gligor, "A fault-tolerant protocol for atomic broadcast," *IEEE Transactions on Parallel and Distributed Systems*, vol. 1, no. 3, pp. 271–285, 1990.

[225] V. Hadzilacos and S. Toueg, "Fault-tolerant Broadcasts and Related Problems," in *Distributed Systems (2Nd Ed.)*. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 1993, pp. 97–145. [Online]. Available: http://dl.acm.org/citation.cfm?id=302430.302435

[226] T. Chandra and S. Toueg, "Unreliable Failure Detectors for Reliable Distributed Systems," *J. ACM*, vol. 43, no. 2, pp. 225–267, 1996.

[227] G. Coulouris, J. Dollimore, and T. Kindberg, *Distributed systems: concepts and design*. Pearson education, 2005.

[228] X. Défago, A. Schiper, and P. Urbán, "Total Order Broadcast and Multicast Algorithms: Taxonomy and Survey," *ACM Comput. Surv.*, vol. 36, no. 4, pp. 372–421, 2004.

[229] I. Abraham *et al.*, "The Blockchain Consensus Layer and BFT," *Bulletin of EATCS*, vol. 3, no. 123, 2017.

[230] A. Singh *et al.*, "BFT Protocols Under Fire," in *USENIX NSDI 2008*, vol. 8, pp. 189–204.

[231] K. Christidis and M. Devetsikiotis, "Blockchains and Smart Contracts for the Internet of Things," *IEEE Access*, vol. 4, pp. 2292–2303, 2016.

[232] D. Kreutz *et al.*, "Software-Defined Networking: A Comprehensive Survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2015.

[233] F. Cristian, "Understanding Fault-tolerant Distributed Systems," *Communications of the ACM*, vol. 34, no. 2, pp. 56–78, 1991.

[234] C. Dwork, N. Lynch, and L. Stockmeyer, "Consensus in the presence of partial synchrony (preliminary version)," in *ACM PODC 1984*.

[235] ——, "Consensus in the presence of partial synchrony," *Journal of the ACM (JACM)*, vol. 35, no. 2, pp. 288–323, 1988.

[236] M. Fischer, N. Lynch, and M. Paterson, "Impossibility of Distributed Consensus with One Faulty Process," *J. ACM*, vol. 32, no. 2, pp. 374–382, 1985.

[237] J. Aspnes, "Randomized protocols for asynchronous consensus," *Distributed Computing*, vol. 16, no. 2-3, pp. 165–175, 2003.

[238] C. Cachin, K. Kursawe, and V. Shoup, "Random Oracles in Constantipole: Practical Asynchronous Byzantine Agreement Using Cryptography," *Journal of Cryptology*, vol. 18, no. 3, pp. 219–246, 2005.

[239] E. Brewer, "Towards robust distributed systems," in *ACM PODC*, vol. 7, 2000.

[240] S. Gilbert and N. Lynch, "Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services," *SIGACT News*, vol. 33, no. 2, pp. 51–59, 2002.

[241] N. Lynch, M. Fischer, and R. Fowler, "A Simple and Efficient Byzantine Generals Algorithm," Georgia Inst of Tech school od information and computer science, Tech. Rep., 1982.

[242] M. Fischer and N. Lynch, "A lower bound for the time to assure interactive consistency," *Information processing letters*, vol. 14, no. 4, pp. 183–186, 1982.

[243] D. Dolev *et al.*, "An efficient algorithm for byzantine agreement without authentication," *Information and Control*, vol. 52, no. 3, pp. 257–274, 1982.

[244] I. Askoxylakis *et al.*, *Computer Security - ESORICS*. Springer, 2016.

[245] V. Buterin, "Ethereum News: On Stake," accessed: 2019-07-01. [online]: https://blog.ethereum.org/2014/07/05/stake.

[246] I. Abraham *et al.*, "Distributed computing meets game theory: robust mechanisms for rational secret sharing and multiparty computation," in *ACM PODC 2006*, pp. 53–62.

[247] W. Dai, "B-money (Blockchain)," [online:]: http://www.weidai.com/bmoney.txt .

[248] C. Dwork and M. Naor, "Pricing via processing or combatting junk mail," in *CRYPTO 1992*, pp. 139–147.

[249] D. Eastlake and P. Jones, "US secure hash algorithm 1 (SHA1)," *RFC 3174, DOI 10.17487/RFC3174*, 2001.

[250] N. Szabo, "Bit Gold, 2008," accessed: 2019-07-01. [online]: http://unenumerated.blogspot.de/2005/12/bit-gold.html.

[251] H. Finney, "RPOW - Reusable PoW," accessed: 2019-07-01. [online]: http://cryptome.org/rpow.htm.

[252] I. Eyal, "The miner's dilemma," in *IEEE SP 2015*, pp. 89–103.

[253] L. Luu *et al.*, "Demystifying incentives in the consensus computer," in *ACM CCS 2015*, pp. 706–719.

[254] B. Wiki, "Testnet," accessed: 2019-07-01. [online]: https://en.bitcoin.it/wiki/Testnet.

[255] Y. Sompolinsky and A. Zohar, "Accelerating Bitcoin's Transaction Processing Fast Money Grows on Trees, Not Chains," accessed: 2019-07-01. [online]: https://pdfs.semanticscholar.org/4016/80ef12c04c247c50737b9114c169c660aab9.pdf.

[256] H. Okada, S. Yamasaki, and V. Bracamonte, "Proposed classification of blockchains based on authority and incentive dimensions," in *ICACT 2017*, Feb, pp. 593–597.

[257] M. Belotti, S. Kirati, and S. Secci, "Bitcoin pool-hopping detection," in *IEEE RTSI 2018*.

[258] A. Gervais *et al.*, "Is Bitcoin a Decentralized Currency?" *IEEE Security Privacy*, vol. 12, no. 3, pp. 54–60, 2014.

[259] B. Wiki, "Scrypt proof of work," accessed: 2019-07-01. [online]: https://en.bitcoin.it/wiki/Scrypt_proof_of_work.

[260] C. Percival, "Stronger key derivation via sequential memory-hard functions," *Self-published*, 2009, [Online]: http://www.bsdcan.org/2009/schedule/attachments/87_scrypt.pdf.

[261] J. Zhou, K. Yu, and B. Wu, "Parallel frequent patterns mining algorithm on GPU," in *IEEE ICSMC 2010*, pp. 435–440.

[262] G. Pinto, F. Castor, and Y. Liu, "Mining questions about software energy consumption," in *ACM MSR 2014*, pp. 22–31.

[263] M. B. Taylor, "The Evolution of Bitcoin Hardware," *Computer*, vol. 50, no. 9, pp. 58–66, 2017.

[264] S. King, "Primecoin: Cryptocurrency with prime number proof-of-work," *Working paper*, 2013, accessed: 2019-07-01. [online]: http://primecoin.io/bin/primecoin-paper.pdf.

[265] J. Andersen and E. Weisstein, "Cunningham chain. from mathworld–a wolfram web resource," 2005.

[266] A. Coventry, "NooShare: A decentralized ledger of shared computational resources," Technical report, Apr. 2012, accessed: 2019-07-01. [online]: http://web.mit.edu/alex_c/www/noosharepdf.

[267] A. Shoker, "Sustainable blockchain through proof of exercise," in *IEEE NCA 2017*, pp. 1–9.

[268] B. Marshall *et al.*, "Proofs of Work from Worst-Case Assumptions," Cryptology ePrint Archive, Report 2018/559, 2018, accessed: 2019-07-01. [online]: https://eprint.iacr.org/2018/559.

[269] I. Bentov *et al.*, "Proof of Activity: Extending Bitcoin's Proof of Work via Proof of Stake," Cryptology ePrint Archive, Report 2014/452, 2014, accessed: 2019-07-01. [online]: https://eprint.iacr.org/2014/452.

[270] I. Bentov, R. Pass, and E. Shi, "Snow white: Provably secure proofs of stake." *IACR Cryptology ePrint Archive*, vol. 2016, p. 919, 2016.

[271] A. Kiayias *et al.*, "Ouroboros: A provably secure proof-of-stake blockchain protocol," in *CRYPTO 2017*, pp. 357–388.

[272] P. Singh *et al.*, "Performance Comparison of Executing Fast Transactions in Bitcoin Network Using Verifiable Code Execution," in *ADCONS 2013*, Dec.

[273] S. King and S. Nadal, "Peercoin–secure & sustainable cryptocoin," accessed: 2019-07-01. [online]: https://peercoin.net/whitepaper.

[274] A. Penzl *et al.*, "SNAPSHOT-Nxt unsurpassable blockchain solutions," accessed: 2019-07-01. [online]: https://www.nxter.org/snapshot-nxt-unsurpassable-blockchain-solutions.

[275] L. Ren, "Proof of stake velocity: Building the social currency of the digital age," Technical report, 2014, accessed: 2019-07-01. [online]: https://www.reddcoin.com/papers/PoSV.pdf.

[276] P. Vasin, "Blackcoin's proof-of-stake protocol v2," accessed: 2019-07-01. [online]: https://blackcoin.co/blackcoin-pos-protocolv2-whitepaper.pdf.

[277] "Novacoin," accessed: 2019-07-01. [online]: https//:altcoinwiki.org/en/Novacoin.

[278] V. Buterin and V. Griffith, "Casper the friendly finality gadget," *arXiv preprint arXiv:1710.09437*, 2017.

[279] V. Buterin, "Understanding Serenity, part I: Abstraction," accessed: 2019-07-01. [online]: https://blog.ethereum.org/2015/12/24/understanding-serenity-part-i-abstraction.

[280] "Tendermint Byzantine-fault tolerant state machine replication," accessed: 2019-07-01. [online]: http://tendermint.com.

[281] A. Miller *et al.*, "Permacoin: Repurposing Bitcoin Work for Data Preservation," in *IEEE SP 2014*, pp. 475–490.

[282] S. P. *et al.*, "SpaceMint: A Cryptocurrency Based on Proofs of Space," Cryptology ePrint Archive – Report, 2015/528, accessed: 2019-07-01. [online]: https://eprint.iacr.org/2015/528.

[283] A. Haleem *et al.*, "Helium: A Decentralized Machine Network," white Paper, Accessed: 2019-07-01. [online]: http://whitepaper.helium.com/.

[284] C. Cachin, "Yet another visit to Paxos," *IBM Research, Zurich, Switzerland, Tech. Rep. RZ3754*, 2009.

[285] C. Cachin, S. Schubert, and M. Vukolić, "Non-determinism in byzantine fault-tolerant replication," *arXiv preprint arXiv:1603.07351*, 2016.

[286] S. Liu *et al.*, "XFT: Practical Fault Tolerance beyond Crashes," in *OSDI 2016*.

[287] "NEO White Paper," accessed: 2019-07-01. [online]: http://docs.neo.org/en-us.

[288] "Proof of Authority," accessed: 2019-07-01. [online]: https://wiki.parity.io/Proof-of-Authority-Chains.

[289] "Aura-Authority Round," accessed: 2019-07-01. [online]: https://wiki.parity.io/Aura.html.

[290] "Clique PoA protocol," accessed: 2019-07-01. [online]: https://github.com/ethereum/EIPs/issues/225.

[291] K. Cong, "A Blockchain Consensus Protocol With Horizontal Scalability," *Master Thesis, Delft University of Technology*, 2017, accessed: 2019-07-01. [online]: https://infoscience.epfl.ch/record/232895.

[292] E. Kogias *et al.*, "Enhancing bitcoin security and performance with strong consistency via collective signing," in *USENIX Security 2016*, pp. 279–296.

[293] C. Decker, J. Seidel, and R. Wattenhofer, "Bitcoin meets strong consistency," in *ACM ICDCN 2016*, p. 13.

[294] I. Abraham *et al.*, "Solidus: An incentive-compatible cryptocurrency based on permissionless byzantine consensus," *arXiv preprint arXiv:1612.02916*, 2016.

[295] E. Kokoris *et al.*, "OmniLedger: A Secure, Scale-Out, Decentralized Ledger via Sharding," Cryptology ePrint Archive, Technical Report 2017/406, accessed: 2019-07-01. [online]: //https:eprint.iacr.//org/2017/406.

[296] Y. Gilad *et al.*, "Algorand: Scaling byzantine agreements for cryptocurrencies," in *ACM SOSP 2017*, pp. 51–68.

[297] "Slimcoin: A Peer-to-Peer Crypto-Currency with Proof-of-Burn," Technical report, 2014, accessed: 2019-07-01. [online]: http://www.doc.ic.ac.uk/~ids/realdotdot/crypto_papers_etc_worth_reading/proof_of_burn/slimcoin_whitepaper.pdf.