Computer Science Review 50 (2023) 100575

Contents lists available at ScienceDirect

Computer Science Review

journal homepage: www.elsevier.com/locate/cosrev

Review article

Understanding blockchain: Definitions, architecture, design, and system comparison

Mohammad Hossein Tabatabaei [*], Roman Vitenberg, Narasimha Raghavan Veeraragavan

Department of Informatics, University of Oslo, Oslo, Norway

ARTICLE INFO

Article history:
Received 12 December 2022
Received in revised form 13 July 2023
Accepted 24 July 2023
Available online 16 August 2023

Keywords:
Blockchain system
Distributed ledger technology
Bitcoin
Ethereum
Hyperledger Fabric
IOTA

ABSTRACT

The explosive advent of the blockchain technology has led to hundreds of blockchain systems in the industry, thousands of academic papers published over the last few years, and an even larger number of new initiatives and projects. Despite the emerging consolidation efforts, the area remains highly turbulent without systematization, educational materials, or cross-system comparative analysis.

In this paper, we provide a systematic and comprehensive study of four popular yet widely different blockchain systems: Bitcoin, Ethereum, Hyperledger Fabric, and IOTA. The study is presented as a cross-system comparison, which is organized by clearly identified aspects: definitions, roles of the participants, entities, and the characteristics and design of each of the commonly used layers in the cross-system blockchain architecture. Our exploration goes deeper compared to what is currently available in academic surveys and tutorials. For example, we provide the first extensive coverage of the storage layer in Ethereum and the most comprehensive explanation of the consensus protocol in IOTA. The exposition is due to the consolidation of fragmented information gathered from white and yellow papers, academic publications, blogs, developer documentation, communication with the developers, as well as additional analysis gleaned from the source code. We hope that this survey will help the readers gain in-depth understanding of the design principles behind blockchain systems and contribute towards systematization of the area.

Contents

[*] Corresponding author.
E-mail addresses: mohammht@ifi.uio.no (M.H. Tabatabaei), romanvi@ifi.uio.no (R. Vitenberg), raghavan@ifi.uio.no (N.R. Veeraragavan).

https://doi.org/10.1016/j.cosrev.2023.100575
1574-0137/© 2023 Elsevier Inc. All rights reserved.

1. Introduction

Since Bitcoin inception and deployment back in 2009, it has been creating waves of discussions and debates about its success as the first widely popular cryptocurrency effort. Bitcoin did away with a broker entity and showed that it is fundamentally possible to have a working cooperative economic model in practice where the technology does all the mediation.

In 2014–2016, there occurred an explosive extension in the range of applications, with everybody talking about the *Blockchain Paradigm* underlying the technology that enables cooperative economic models and cooperative storage and management of agreements. According to [1], the annual revenue of blockchain-based enterprise applications worldwide will reach $19.9 billion by 2025, up from about $2.5 billion in 2016.

Declared intention of future use of blockchain in cooperative healthcare provision is making global news [2]. Cooperative energy markets based on microgrids are viewing blockchain as a major technological enabler [3]. Blockchain is becoming a common element in solutions for decentralized digital identity (DID) [4], certificate storage [5], land registries [6], and supply chains [7]. Microsoft declares "blockchain" as one of the key "must win" technologies for their Azure platform and business [8]. Similarly, IBM unveils a new ambitious blockchain service and strategy [9]. A long row of major enterprises such as Accenture, Cisco, Citibank, Facebook, Disney Studios, Goldman Sachs, and HSBC have publicly declared their investment in the technology with the intent of exploring the potential for exploitation, which has resulted in an establishment of industrial blockchain associations such as Diem [10]. Governments in North America, Europe, and Asia are advancing blockchain-related strategy and legislation, reflecting their significant interest towards the advent and utilization of the technology [11–16]. Blockchain has become one of the hottest topics at many societal, industrial, and academic conferences.

The above has resulted in hundreds of blockchain systems in the industry, thousands of academic papers published over the last few years, and an even larger number of new initiatives and projects. Following this explosive development, the area currently remains tumultuous, without commonly accepted terminology and with occasionally diverging concepts and ideas. For example, we list five different definitions of the term "blockchain" in Section 2.1, all being used in the literature and in description of actual systems.

Emerging surveys such as [17,18] among all have made an important first step towards orderly understanding of the area. However, most of the published surveys are focusing on one system (such as Bitcoin [19,20]), a particular class of blockchain (e.g., DAG-based [21]), or an individual aspect of blockchain (such as proof-of-X or defense mechanisms against security attacks of a specific type). In-depth comparisons across systems of different type are exceedingly rare, and so are standardization initiatives or attempts to establish general taxonomies. Thus, despite the consolidation efforts, the area remains highly turbulent without systematization. Accordingly, there are no textbooks or comprehensive educational materials, except for Bitcoin itself [22].

Today, the only way to gain in-depth technical understanding of the design behind almost any blockchain system is to read typically outdated white and yellow papers followed by perusing the continuously updated technical documentation, followed by scanning hundreds of blog posts by the developers and conflicting forum posts by the users and finally, by studying the source code.

1.1. Contributions

Our main contribution is the first systematic and comprehensive comparative study of blockchain design across different systems. While many existing works provide a focused cross-system comparison on a particular aspect, we aim to provide a comparative design insight that goes far beyond the information that is currently available in the literature. The comparison is performed across four blockchain systems representative of different blockchain strands (i.e., Bitcoin, Ethereum, Hyperledger Fabric, and IOTA) with divergence in the design priorities, architectural elements, performance, and even roles of the participants and basic definitions. While IOTA is not as popular as the other three, it is the most commonly used DAG-based system and it was ranked at the fourth place [23] by the market cap in 2017.

First, we present generic roles of the participants in blockchain along with the significance of each role and explain how the entities (i.e., computing devices of different categories) in a blockchain system of each type map to these roles. We introduce a generic layered architecture that applies to all blockchain systems regardless of the type. The study of the four systems is organized across these layers so that the design of each layer is considered separately from the rest. This methodology allows us to conduct a comprehensive cross-system comparison. The comparison is organized by clearly identified aspects: definitions, roles, entities, and the characteristics and design of each of the layers. We also contrast the performance of the four systems based on the previously published information and explain the reasons for the differences.

Our exploration goes deeper compared to what is currently available in academic surveys and tutorials. For example, we provide the first extensive coverage of the storage layer in Ethereum and the most comprehensive explanation of the consensus protocol in IOTA. The exposition is due to the consolidation of fragmented information about popular systems from blogs, developer documentation, and studying the source code.

Our main emphasis is on the education and pedagogical exposition that lends itself to courses and tutorials. While such descriptions exist for Bitcoin, no such materials are available for Ethereum and IOTA to the best of our knowledge. A more fine-grained summary of our contribution along with the comparison with existing surveys is available in Section 5.

Our study focuses on the design of the systems themselves rather than on application mechanisms developed atop them. In particular, we do not cover hybrid storage systems that combine on-chain and off-chain elements. Besides, the scope does not include functional features such as sharding that are still in development and that are not supported by the current versions of the blockchain systems.

1.2. Roadmap

In Section 2, we first contrast various blockchain definitions and reflect on the discrepancies in the commonly used terminology. Then, we introduce a list of roles of the participants and a layered blockchain architecture that are both applicable to all blockchain systems regardless of the type. In Section 3, we present an overview of the four systems, discuss the entities in each, and explain how they map to the generic roles. In Section 4, we provide a layer-by-layer comparison between the four systems while covering a variety of design aspects and characteristics. In Section 5, we contrast our work with other state-of-the-art surveys. Finally, we present our conclusions in Section 6.

2. Understanding blockchain: Definitions and concepts

In this section, we review and reflect upon central definitions and concepts of blockchain technologies.

2.1. Reflection on various blockchain definitions

The term of "blockchain" generally refers to a paradigm for maintaining information in a distributed system that is characterized by a number of properties. Since there is no specification or established standards in 2023 yet, different concretizations of this general definition have been adopted in the literature and existing popular blockchain-based systems.

Distributed Ledger Technologies (DLTs, in short) is a well-defined term: it refers to a system that records a ledger of transactions or a history of changes to the system state. The ledger is usually hard to tamper with, which is a boon for security, yet it also makes it hard to perform desirable changes, e.g., to prune the history or compact the ledger.

While people tend to equate blockchain with DLTs, both narrower and broader meanings of "blockchain" are in use. Literally, blockchain means "a chain of blocks", which implies a specific data structure for the ledger implementation. A chain of blocks precludes any parallelism between the transactions, however, which has a negative impact on the performance. Some ledger implementations use more a complex data structure such as braids [24] or a directed acyclic graph (DAG) in IOTA, which allow some degree of parallelism by retaining concurrently proposed competing blocks and merging them. Since the term of DLT does not imply any specific data structure, it covers such a generalization. On the other hand, the term of "blockchain" becomes a misnomer in that case. In absence of more refined terminology today, "blockchain" is used in the literature to refer to a chain of blocks or generalized DLTs.

To add to the confusion, some systems in this domain do not maintain a distributed ledger at all. For example, Corda [25] allows participating computing devices to agree upon and maintain shared knowledge in a non-trusted environment typical for blockchain. However, each piece of information is only shared within a subset of computing devices to which the information pertains. Yet, the term of "blockchain" is sometimes used to collectively refer to all systems in the domain including Corda.

We have been able to identify the following definitions in the literature:

Definition 1. Blockchain is a system that uses the data structure of Bitcoin but extends the functionality. This definition is used by, e.g., Bitcoin spin-offs that were created either due to hard forks or as an extension of the limited scripting functionality of Bitcoin. This definition is not limited, however, just to cryptocurrency systems; it can be utilized for a large spectrum of business logic by customizing the blockchain modules and protocols.

Definition 2. Blockchain is a system that maintains a chain of blocks. This definition allows for generalization of Definition 1: it allows data structures other than those used in Bitcoin. For example, Ethereum and Hyperledger match this definition.

Definition 3. Blockchain is a system that maintains a ledger of all transactions. The ledger does not need to be stored as a chain of blocks, however. IOTA is an example of a system that follows this definition.

Definition 4. Blockchain is a system with distributed non-trusting parties collaborating without a trusted intermediary. This definition rather refers to the main beneficial property of the paradigm. It was originally advocated by Corda [25].

Definition 5. Blockchain is a system that provides support for smart contracts. Many blogs and popular science articles (such as [26]) regard blockchain as a way of replacing paper-based contracts and human intermediaries with smart contracts, without considering how such contracts are implemented.

The first three definitions above are sorted by generality, from the most concrete to the most general. While they refer to the way the system is built, the last two definitions are about the way the system is used.

It is important to distinguish between definition of blockchain and its characterization. While the definition has not been universally agreed upon, the fundamental properties have been extensively explored in the prior literature. For example, blockchain data is immutable: new data can be added but already included data cannot be deleted or modified. Blockchain additionally provides tamper-resilience, i.e., protection of blockchain data against any unwanted modification. Since immutability and tamper-resilience are explained and discussed in a large body of literature [27], we do not cover them in this paper.

In the absence of proper definition, a blockchain is sometimes compared to a distributed tamper-resilient database with immutable data. It is important to observe that a blockchain differs from such a database in two fundamental ways. First, a database is an organized collection of data representing the current system state. The main functionality of a database is to allow efficient data retrieval, fusion, and aggregation triggered by user queries.

In contrast, most blockchain implementations represent a ledger in which a history of transactions (or, more generally, of changes to the system state) is recorded. For example, there is simply no concept of a user balance in Bitcoin! While Ethereum keeps track of a contract state, it only provides limited means to retrieve and process state data, as explicitly defined by the contract. It does not support abstractions of a flexible query language, data view, schema, join, etc. Besides, Ethereum records a history of all changes to the state, which results in blockchain space being more expensive and the storage less efficient compared to a database. As a result, only specific data elements (such as short transactions or indices) are stored on a blockchain. Many blockchain systems combine blockchain with offchain storage (databases or dedicated file systems).

Secondly, the trust model is radically different as observed in [25]. The database servers typically trust each other, even in federated databases, in the sense that they do not expect attacks from within the system. The main security focus is on making it difficult to compromise a server in the first place. To this end, database systems defend against malicious clients by using firewalls, strict access control, and many other methods. The situation is fundamentally different in the blockchain environment: the interests of participating computing devices are inherently misaligned so that they need to verify information received from each other and run a consensus to agree on changes to the data. While being able to agree on changes and progress in absence of a trusted administrator is a powerful abstraction, it bears a cost tag in terms of performance. If there are no misaligned interests between the participants and attacks from within the system are unlikely, there is little point in using blockchain technologies.

2.2. Blockchain types

The two most important characterizations are the constraints on which participants are allowed to propose updates to the blockchain[1] and which participants are allowed to read blockchain data. In Bitcoin, any computing device may propose updates to the ledger or read it. This is also the case for the public deployments of Ethereum and IOTA. In Hyperledger Fabric, only authorized computing devices are allowed either. However, mixed models are also possible: In Ripple, every computing device has read access while only authorized computing devices can propose updates. We call these characterizations *update-access-restricted* and *query-access-restricted*, respectively. We refer to a system that is update-access-restricted or query-access-restricted as *access-restricted*. Obviously, systems that are access-restricted require computing device authorization.

Usually, the identities are also handled differently in systems that are access-restricted. Namely, such systems typically use real computing device identities while systems that are not access-restricted commonly use public keys for identification. In principle, however, systems that are not access-restricted can still use real identities, if such identities can be verified and if the transparency of participation is more important than privacy. It is also not unimaginable for access-restricted systems to employ public keys or pseudonyms, though it may require the authorization or authentication component to operate under different trust and security assumptions compared to the blockchain system itself.

There exists another related dimension for classifying blockchain systems, namely *decentralization*. Large-scale decentralized blockchain systems, such as Bitcoin, Ethereum, and IOTA, may have up to hundreds of thousands participating computing devices [28,29] without any coordinated management of the system structure, organization of computing devices, or network connections. On the other hand, consortium blockchain systems such as Hyperledger Fabric are smaller proprietary deployments where the consortium may decide to partially manage all of the above. We observe, however, that even consortium blockchain does not lend itself to complete management. In other words, all blockchain systems are self-organizing, though the extent of self-organization varies across the systems. The scale and extent of management significantly affects design priorities and implementation components as we explain in the rest of this survey. Consortium systems are access-restricted by nature. Large-scale decentralized systems are not query-access-restricted, though they may be update-access-restricted.

Unfortunately, the above complexity is not reflected in the currently existing terminology. All blockchain systems are coarsely divided into two categories in the existing literature. The first category is referred to as public or open or permissionless while the second category is called private or closed or permissioned. The exact meaning of these six terms and differences between them are not precisely defined to the best of our knowledge.

2.3. Participants and their roles

Since the models and implementations significantly differ in various blockchain systems, we need to identify common fundamental elements in order to perform a systematic comparison. Two such unifying elements are the roles of participating computing devices and the conceptual layered architecture. We discuss these two elements in Sections 2.3 and 2.4, respectively. They apply to all blockchain systems studied in the rest of the paper, and they will likely generalize to many other blockchain approaches as well.

As commonly accepted in descriptions of distributed architectures, roles refer to the functional responsibilities. The same computing device may play a single or multiple roles in the system. The roles in a blockchain system are presented in Fig. 1.

Creators of Transactions: Different entities implementing a blockchain application can create transactions and inject them into the system by relaying them to the proposers.

Proposers and Acceptors: A central functionality of a blockchain system is to validate injected transactions and decide which transactions will be appended to the blockchain and in what order. This is the main responsibility of computing devices acting as acceptors. To this end, they need to run a distributed consensus protocol. However, all consensus solutions have an inherent limitation when it comes to scalability: they do not work very well if there are too many acceptors or too many concurrent transactions to be considered. This also makes consensus protocols susceptible to denial-of-service (DoS) attacks: an attacker can bombard a blockchain system with invalid transactions, effectively stalling consensus progress. To improve the scalability and resilience to DoS, most blockchain systems introduce the role of a proposer. Proposers act as intermediaries between creators of transactions and acceptors. They may reduce the rate of concurrent proposals by (a) verifying and pre-authorizing them locally and filtering out invalid or non-authorized transactions, (b) introducing explicit rate control, and (c) batching multiple transactions into a block. The exact distribution of responsibilities between the acceptor and proposer roles depends on a specific blockchain system but the conceptual separation applies to most systems. Since blockchain systems do not assume that all computing devices are trustworthy, they may need to incentivize the blockchain participants to perform their roles correctly, without deviations. Additionally, proposers and acceptors contribute to making the blockchain tamperproof, together with data recorders.

[1] Throughout this paper, by updating the blockchain we mean appending new data. Other types of updates are impossible due to data immutability.

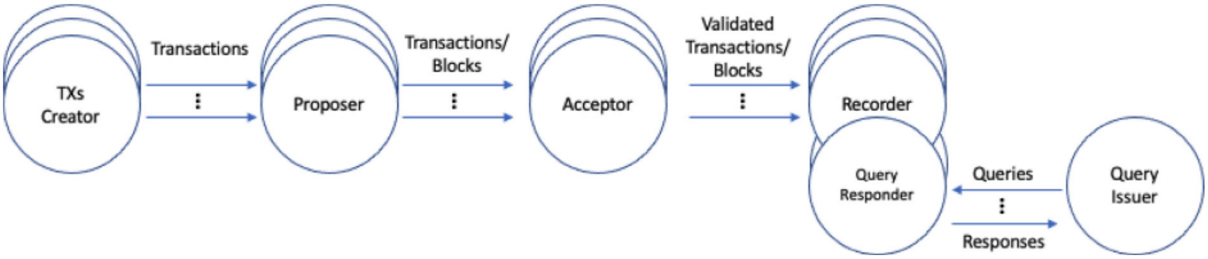

Fig. 1. Roles of the participating computing devices and the relations between these roles.

Data recorders: These entities record the additions to the blockchain accepted by the acceptors, which results in storing the entire blockchain. Along with proposers and acceptors, they contribute to data security by making the data tamperproof through the use of cryptographic primitives.

Query issuers: They issue queries of different types over the current blockchain data.

Query responders: Some of the computing devices playing the role of data recorders have an additional responsibility of responding to the queries from query issuers based on the stored blockchain data. The need for and the exact role of query responders is further detailed in Section 4.4.6.

2.4. *Architecture*

The design of blockchain systems is based on a layered architecture, which we show in Fig. 2.

Hardware Layer: This is the bottommost layer of a blockchain system. While most blockchain systems can be deployed without any specialized hardware, hardware can make the computation (e.g., of cryptographic hashes) more efficient, provide extra security of the storage and computing environment, etc.

Data Storage Layer: This is the most important part of a blockchain system when it comes to storing data, keeping it safe from modifications, and making it traceable. This layer is also responsible for providing availability and durability of data. All of these features depend on the data items used in this layer, the method of storing the data, and the structure of the storage.

Communication Layer: Any blockchain system needs a mechanism for disseminating the transactions and blocks between the participants. This is especially important for large-scale permissionless blockchain systems. The granularity of dissemination, protocol of communication, ordering guarantees, privacy and security guarantees, and propagation time are related to this layer. All computing devices participating in a deployment of a blockchain system can be divided into two categories according to their involvement in the dissemination: core nodes constituting the blockchain network and “client” computing devices. Core nodes cooperate based on a common protocol to maintain blockchain data. In contrast, client computing devices do not play an active role in the dissemination. Instead, they get specific blockchain data of interest from core nodes.

Data Manipulation Layer: The main responsibilities of the data manipulation layer include updating the blockchain and offering a search functionality for blockchain data. Since an update must be a coordinated decision in a decentralized environment with a lack of trust between individual participants, the consensus protocol plays a central role in all blockchain systems. Consensus protocols vary across blockchain systems [30], though the two most common categories are probabilistic Nakamoto consensus and BFT. Design choices of the data manipulation layer can affect different parameters related to performance and the strength of security and privacy guarantees, including the ability to withstand or mitigate DoS attacks.

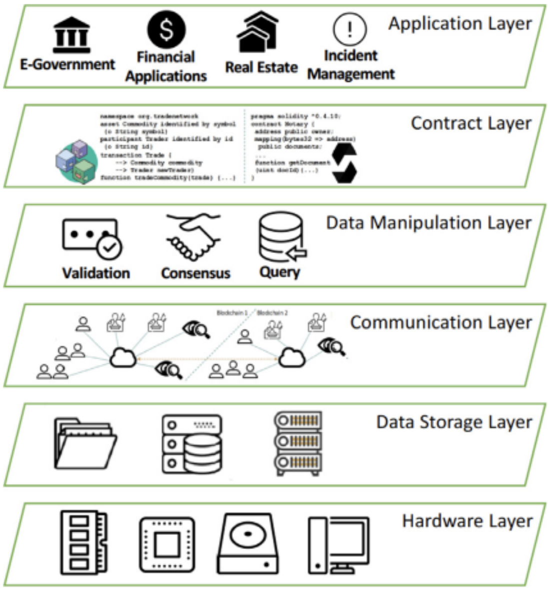

Fig. 2. Layers of a blockchain system.

Contract Layer: Blockchain systems provide the ability of defining contracts for creating and enforcing the rules among the participants of the network. Different blockchain systems may use various methods for developing the contracts based on their application domain. This layer deals with invoking and executing transactions within the contract, as well as programming languages (which may be Turing-complete or not) and execution environments.

Application Layer: This is an abstraction of the application built atop a blockchain system. While its specific design is outside the scope of a blockchain system, it defines the requirements that a blockchain system needs to satisfy.

3. Understanding blockchain systems: overview and entities

While pursuing the question of in-depth understanding of blockchain, we opt to illustrate the range of possibilities by analyzing four blockchain systems with significantly different properties and implementations. In this work, Bitcoin Core, Ethereum 1.0, Hyperledger Fabric v.2.0, and IOTA IRI v.1.5 are compared to each other in terms of blockchain entities and features and mechanisms of each architecture layer. The blockchain field is known for dynamic developments, with relatively short update cycles. However, Bitcoin does not undergo significant changes at this point so that we expect all Bitcoin-related descriptions to

Table 1
Mapping roles to the entities of different blockchain systems.

Blockchain system	Transaction creator	Proposer	Acceptor	Recorder	Query responder	Query issuer
Bitcoin	Nodes	Miners	Full nodes	Full nodes	Dedicated services on top of full nodes	Everyone[a]
Ethereum	Nodes	Miners	Full nodes	Full nodes	Dedicated services on top of full nodes	Everyone
Hyperledger Fabric	Application clients	Application clients, endorsers & orderers	Orderers & peers	Peers	Application-dependent	Application clients
IOTA	Clients	Nodes	Nodes	Nodes & permanodes	Dedicated services on top of nodes & permanodes	Everyone

[a] Everyone means any networked device.

remain valid in the future. Ethereum has declared its intent to release a new version with several significant changes, including a transition from PoW to PoS. However, no studies of the new version have been published ahead of the release so that we base our survey on the current operational version. While Hyperledger Fabric continues to be under active development, the new versions introduce new features and performance optimizations, switch to a different default consensus implementation, and modify the chaincode features and structure. They do not radically change the fundamental design of each layer, which we discuss in our survey. Finally, IOTA IRI is currently the best documented version of IOTA that is presented in the white paper [31].

Table 1 maps the blockchain roles to the entities of different blockchain implementations. In the following, we explain the functionality of each entity and the interactions happening between the entities.

3.1. Bitcoin

In the Bitcoin deployment, neither update access nor query access is restricted. Bitcoin participants are called Bitcoin nodes in the Bitcoin terminology. They are split into two categories: full and lightweight. Some roles are only performed by full nodes, which act as both block acceptors and recorders of the whole blockchain. Upon receiving a newly broadcast block from the network, they verify it using the previously stored data and potentially append it to the blockchain. Additionally, full nodes bear higher responsibility w.r.t. dissemination in the Bitcoin network as we explain in Section 4.3. A subset of full nodes is designated as miners that continuously attempt to create and propose new blocks.

Lightweight nodes are also called simplified payment verification (SPV) [32] nodes. SPV nodes neither store the entire blockchain nor participate in the decision of accepting new blocks. They can however, use the SPV method to verify that a block header is valid and that a given transaction is included into the blockchain. To this end, they can retrieve block headers and specific secure hashes from a full node and employ a Merkle verification algorithm. Merkle verification is explained in Section 4.2.

Both full and lightweight nodes are able to create transactions, thereby playing the role of a transaction creator. This effectively means that the separation between the entities is more blurred in Bitcoin (and similarly in Ethereum) compared to say, Hyperledger Fabric and IOTA. In particular, both terms of "lightweight node" and "lightweight client" are used in Bitcoin to refer to the same entities.

Transactions are propagated to the miners that play the role of proposers. As common for large-scale deployments, the Bitcoin network cannot sustain every miner making a proposal at an arbitrary point in time because of the scalability barrier and potential DoS attacks by the miners. It is therefore needed to moderate the number of proposers and rate of concurrent proposals. The entire Bitcoin system must attain a balance with respect to the proposal rate: a high rate will hamper scalability while a low rate will limit progress and transaction throughput. Furthermore, Bitcoin must achieve that moderation through distributed mechanisms, without a centralized moderating entity. In order to keep the rate down, Bitcoin introduces a *Proof-of-work (PoW)* mechanism: When forming a block, a miner needs to solve a cryptopuzzle by performing a heavy computation and showing the proof of it, in order for the block to be accepted by other full nodes.

For the Bitcoin system, there are dedicated services which act as query responders: they provide information for any computing device on the Internet even if the device is not on the Bitcoin network. However, these services can only handle fixed basic query types so that they are not as semantics-rich as specialized databases. Additional information about queries in blockchain systems is given in Section 4.4.

3.2. Ethereum

The entities in the Ethereum system and the interactions between them are similar to those in Bitcoin. However, Ethereum introduces a number of additional underlying concepts and mechanisms not present in Bitcoin. The most central of those concepts is that of gas: since transactions in Ethereum can be Turing-complete programs, it is impossible to predict how much of computational and storage resources a transaction will consume. It is only feasible to monitor resource consumption at runtime and abort the transaction if it exceeds the permitted budget. Ethereum manages budgets for executable units such as transactions, blocks, smart contracts, etc. These budgets are measured in gas and paid for by the creators of transactions. Ethereum assigns gas cost to resources (operations and storage units). When a transaction executes an operation, Ethereum deducts the cost of that operation from the budget. In Section 4, we consider additional concepts and mechanisms in the context of each layer and cover them in detail.

3.3. Hyperledger Fabric

As Hyperledger Fabric is designed for use in private blockchain systems, it does not have the concept of miners. Furthermore, its entities and their responsibilities are completely different compared to Bitcoin and Ethereum. There are three types of entities in Fabric: *application clients, peers, and orderers*. Clients are computing devices that constitute the application running atop blockchain and play the role of transaction creators. They are external to Fabric and they connect to participants in the Fabric network. The peers are authenticated and authorized participants in the Fabric network that constitute the core of the private blockchain network. The orderers are dedicated computing devices provided by the Fabric deployment.

The flow in Fabric is as follows: Clients send every proposed transaction to the *endorsing peers* (also called *endorsers*), which are a subset of all peers determined by the endorsement policy. More precisely, the endorsement policy defines the smallest set of endorsers that need to endorse a transaction in order for it to be valid [33]. The endorsers typically come from different organizations within the consortium that deploys an application.

Endorsers are responsible for simulating the transaction without updating the blockchain and verifying if the update causes a conflict with the current state or with the policy. If an endorser approves the transaction, it identifies the set of data read and updated by the transaction, which is called a read–write (or R–W) set. Then, the endorser sends the R–W set back to the client, along with the endorser's signature. If transaction approvals obtained by the client are insufficient according to the endorsement policy, the flow effectively terminates. Otherwise, the client submits the endorsed transaction and the R–W set to the *ordering service*.

Ordering service consists of orderers which are responsible for receiving the endorsed transactions and their R–W sets from different clients, ordering them, and forming a block. The orderers run a consensus protocol to achieve ordering of transactions. Once produced by the consensus protocol, a block is disseminated to all of the peers in the system. Peers validate the block's transactions by checking whether the R–W sets still match the current state of the blockchain. Finally, the peers add successfully verified blocks to the blockchain and update the state.

All peers store blocks, thereby playing the role of data recorders. There is no equivalent of Bitcoin and Ethereum lightweight nodes in Fabric. Orderers may be configured to store blocks as well.

In the above flow, the role of acceptors is distributed between endorsing peers in the first phase, orderers in the second phase, and all peers in the third phase. For example, the peers may reject the block produced by the orderers, if it is malformed or if it causes an inconsistency.

Since the deployment scale of private blockchains such as Fabric is typically smaller compared to a public blockchain such as Bitcoin, the significance of having a proposer role is reduced accordingly, see Section 2.3. However measures towards improving scalability are still taken: Endorsers perform pre-authorization of transactions, clients themselves may abort the flow if the approvals are insufficient as per the endorsement policy, while orderers batch transactions and form blocks.

Regarding query issuers and responders, application clients issue queries in Fabric, while the peers which keep the state respond to those queries.

3.4. IOTA

Similar to Bitcoin and Ethereum, IOTA is a permissionless blockchain system. IOTA organizes transactions into bundles instead of blocks. The main difference is that a bundle groups related transactions together; a single transaction in a bundle cannot be understood or performed independently of other transactions in the bundle. For example, IOTA differentiates between input transactions that may combine funds from multiple input addresses together and output transactions that may split a sum into multiple output addresses. A bundle commonly contains related input and output transactions, as well as transactions of other types. More details are given in Section 4.2.

The main purpose of grouping transactions in a block in Bitcoin, Ethereum, and Hyperledger Fabric is to improve the throughput of the consensus protocol that is used to update the ledger. Bundles cannot be used to this end because they cannot combine unrelated transactions. Interestingly, IOTA achieves higher throughput compared to Bitcoin and Ethereum by organizing bundles in a directed acyclic graph (DAG) as opposed to organizing blocks in a chain. We provide a detailed discussion in the context of the storage and data manipulation layers in Sections 4.2 and 4.4.

The roles and responsibilities in the IOTA deployment are shared by three main entities: (a) clients (b) nodes and (c) permanodes. Clients are external to the IOTA network. They connect to IOTA nodes via the HTTP API and interact with the network through the nodes. A client is responsible for creating transactions and forwarding them to a node.

Nodes are interconnected together to form the core of the IOTA network. After having received a transaction from a client, a node needs to validate it. If the transaction passes the validation, the node will create a bundle as we discuss later, in Section 4.4.4. To prevent spamming and DoS attacks, nodes have to solve low-complexity cryptopuzzles in order to propose bundles. Having performed the low-complexity PoW, the node attaches the bundle to the local copy of the ledger called the Tangle [31] and propagates the bundle to the network. Nodes run a consensus protocol to synchronize their copies of the Tangle.

Note that unlike Bitcoin and Ethereum, nodes do not create competing proposals for updating the ledger. Accordingly, there is no concept of mining in IOTA. Since nodes do not need to perform any significant computation, they are not rewarded for participation in the system either. Nodes are also able to answer limited queries about the Tangle but this is not their primary purpose.

Permanodes are dedicated nodes, many of which are provided by the IOTA Foundation itself. Similarly to nodes, they store a copy of the Tangle. However, they are external entities to the IOTA network: they do not receive bundles from the clients or participate in the consensus protocol. Instead, they receive Tangle updates from nodes. Their main purpose is to support complex queries. To this end they store Tangle information in a special database, as explained in detail in Section 4.2.

In summary, clients play the role of transaction creators and proposers. Nodes play the role of acceptors (validating the transactions in the bundle), recorders (storing the bundles in the local ledger), and query responders (supporting services with simple queries). Permanodes play the role of recorders and query responders; yet they support richer queries compared to nodes.

4. Understanding blockchain systems layer by layer

As we discussed in Section 2.4, blockchain systems are categorized into six different layers: hardware, data storage, communication, data manipulation, contract, and application. We now analyze the features and implementation of each layer in the four selected blockchain systems.

4.1. Hardware layer

While blockchain implementations are primarily software-based, hardware components are used for two purposes: improved efficiency and enhanced security. In particular, implementations of open blockchain systems such as Bitcoin, Ethereum, and IOTA, are resource-bound: there is a hardware resource that limits the ability of a single proposer to propose numerous changes to the blockchain at a fast rate. The exact resource varies across the systems, however. On the other hand, this is not required in permissioned blockchain because of a tight membership control. Table 2 highlights related aspects for the analyzed systems.

Table 2
Hardware layer of different blockchain systems.

Features	Bitcoin	Ethereum	Hyperledger Fabric	IOTA
Limiting resource	Processor	Memory bandwidth	Application-dependent	Processor & network bandwidth
Cryptopuzzle solving device	ASIC	GPU	No cryptopuzzles	Proprietary processor (being phased out)
Additional hardware for security (Research initiatives)	Hardware-based trusted execution environment (e.g. Intel SGX processor)	Hardware-based trusted execution environment	Application-dependent	No additional hardware

As mentioned in Section 3, miners in Bitcoin require to perform costly and time consuming computations in order to create a valid block. They need to solve a cryptopuzzle by exhaustively going over the solution space and calculating a hash value for each potential solution, which results in a massive amount of hash computations. Therefore, the Bitcoin technology is strongly dependent on the processing power of the miners. Nowadays, the most common hardware for the purpose of mining bitcoins are ASIC processors [22]. They are designed specifically for the purpose of computing Bitcoin hashes in an optimized way.

In contrast to Bitcoin, Ethereum follows ASIC-resistant approach in its hardware layer. The goal of choosing such an approach is coming up with a puzzle that reduces the gap between the most cost-effective customized hardware and what most general-purpose computers can do, so that it would be economical for individual users to mine with the computers they already have [22]. To achieve this goal, Ethereum uses a different PoW algorithm compared to Bitcoin, called ethash [34]. The ethash algorithm needs 64 sequential page fetches from the memory to generate a single hash and compare it with the cryptopuzzle target. Since ethash is bound by the speed of memory access rather than computation, speeding up the processor computation by ASICs does not help in a significant way. Furthermore, since an expensive top performing computer only has moderate improvement in the speed of memory access compared to commodity hardware, vertical scaling does not have a strong effect on the efficiency of mining in Ethereum. Hence, utilizing GPU processors, which can solve cryptopuzzles faster than CPUs [35], is more cost-effective for mining in Ethereum. This has an additional effect that cryptopuzzles in Ethereum can be less computationally intensive. On the other hand, the power consumption of GPU is higher compared to ASIC so that the net effect on power consumption in Ethereum is not clear.

The design of cryptopuzzles in both Bitcoin and Ethereum facilitates parallel computation, which increases the risk of mining power centralization in the hands of powerful players. While ethash limits the potential of vertical scaling for mining, horizontal scaling is widely used in both systems.

As there is no concept of mining in Hyperledger Fabric, it does not need specialized hardware components to boost the efficiency, unless required by a specific application. While IOTA does not use blocks and does not employ the concept of mining or monetary rewards, it still uses a less computationally intensive version of cryptopuzzles in order to prevent denial-of-service attacks. Therefore, IOTA is also dependent on the computation power of participating nodes. Since resource-limited IoT devices is the main focus of IOTA implementations [36], IOTA has developed a proprietary low-energy processor called JINN [37]. The main purpose is to expedite the computation of relevant cryptographic primitives, though currently the computation is typically done in software. Besides computation power, bandwidth is another critical resource in IOTA since IoT devices are also bandwidth-limited [38].

A major focus of the blockchain technology is to provide tamperproof storage in absence of trust in individual participants. In view of this, a number of initiatives such as Teechain [39] have tried to enhance the security of Bitcoin and Ethereum by taking advantage of the hardware-based Trusted Execution Environment (TEE) technology, such as Intel® SGX processor [40]. TEE is designed to create a more secure computation environment for the processor by isolating and protecting the running application against unauthorized access or tamper by the host machine. While the use of TEE still requires the trust in the TEE manufacturer, it mitigates potential attacks by individual blockchain participants.

4.2. Data storage layer

The purpose of the storage layer is to record all transactions in a distributed ledger and provide support for their efficient verification. While simple transactions merely transfer financial tokens, general transactions represent transitions in the global system state. It is important to keep track of the state, e.g., in order to perform verification of transactions. While it is possible to reconstruct the most updated state by starting from the initial state and replaying all transactions recorded in the ledger, this would be a time-consuming and inefficient process, especially since the ledger size is continuously growing. As a result, all blockchain systems store explicit information about the current state in addition to the transactions. However, the systems differ in terms of what state-related information they store and in terms of how they organize it. Additionally, the storage layer provides support for verification of transactions issued by the clients and for computation on the state in the ledger.

The rest of the storage layer description covers state tracking approach, general organization of blockchain storage, the block structure, structure of transactions and their grouping in blocks, on-disk storage, the use of trees in Bitcoin and Ethereum, in-memory storage, and data retention. Table 3 presents a comparison of salient storage design aspects across the four systems.

4.2.1. State tracking approach

So far, all blockchain systems have been employing one of the two principal approaches for tracking system state. Bitcoin employs the unspent transactional output (UTXO) approach while Ethereum falls under the arbitrary state approach. IOTA follows the UTXO approach with a minor modification, as we explain below. Fabric does not track state by default but the application developer can devise and plug in any desired implementation with some restrictions. The two approaches differ by generality, compactness of transactions, simplicity and efficiency of storage organization, ease of parallelizing transaction processing, as well as transaction linkability, i.e. that ease at which a transaction can be linked to an individual user.

In the UTXO approach, a transaction transfers currency tokens by consuming a number of input tokens and producing a

Table 3
Data storage layer of different blockchain systems.

Features		**Bitcoin**	**Ethereum**	**Hyperledger Fabric**	**IOTA**
State tracking approach		UTXO	Account-based	Application-dependent	UTXO
Higher-level structure		Sequence of blocks with transactions	Sequence of blocks with transactions	Sequence of blocks with transactions	DAG of transaction bundles with edges signifying approvals
Maximum block/bundle size		1 MB block fixed by the protocol	Block's gas limit determined by miners	Configurable block size as per application	Unlimited bundle size
On-disk storage	Ledger storage	Block data as multiple files of limited size each	Block data as multiple files of limited size each	Block data as multiple files of configured size	Tangle stored in a RocksDB database
	Index	Block index in a LevelDB database	–	Block index in a LevelDB database	–
	State-related storage	UTXO set in a LevelDB database	Each vertex of tries (state, storage, receipts, txs) in a LevelDB database	Worldstate in a LevelDB or CouchDB database	Balance info stored in RocksDB database
	Extra storage elements	Each block file has a corresponding undo file to support reorganization/fork	All block data (block header and transaction data) in a LevelDB database	–	Snapshot data as a file
In-memory storage		Block index database, UTXO cache, & Merkle trees of transactions	Cache of Merkle Patricia tries vertices	Application-dependent; peer cache	Tangle Accelerator (proxy cache for IOTA nodes)
Data retention		No retention policy	State tree pruning	No retention for ledger; application-dependent for the state	Snapshot purges the database

number of output tokens. For example, suppose Alice has previously received 6 tokens in a single transaction. In this case, the six tokens become identifiable transactional outcome (TXO), which is marked as unspent. Assume Alice later wants to transfer 4 tokens to Bob. She can use the six token UTXO as the input to the new transaction. During the transaction execution, the protocol will verify that the UTXO has not been spent. Then, three new UTXOs will be created: one UTXO with transaction fee will be transferred to the block creator, one UTXO worth 4 tokens will be transferred to Bob while another UTXO worth two tokens minus transaction fee will be transferred back to Alice. Additionally, the old six token TXO will be marked as spent. In the UTXO approach, the system only keeps track of the unspent transactional outputs because they can be used as input to future transactions. Since there is no concept of accounts or wallets at the protocol level, the "burden" of maintaining a user's balance is shifted to the client side. Wallets maintain a record of all UTXOs associated with a user and compute the total sum, which represents the balance. However, this balance is inaccessible to all but the user who owns the wallet.

The UTXO approach in IOTA slightly differs in that the term of "address" is used to refer to a collection of multiple TXOs because an address can be reused to receive funds in multiple transactions, thus acting as a pseudo-account. The storage system in IOTA keeps track of all transactions that have transferred funds to a given address. Like a TXO, an address can only be used once when sending funds such that the entire amount is spent. This minor difference in IOTA does not have a significant impact on the properties of the UTXO approach, which we discuss below.

The arbitrary state approach is more general in that it supports arbitrary state rather than just tokens transferred between the users. This naturally leads to more complex storage structures compared to the UTXO approach. In particular, it is common to keep the state partitioned. For example, each user in Ethereum has an associated account and the state is partitioned by those accounts so that the state tracking approach in Ethereum is commonly referred to as "account-based". Since there exists no single public deployment of Hyperledger, the state in Hyperledger is kept separately for each proprietary deployment. Besides, the worldstate in Fabric is further partitioned per channel. The configurable nature of Hyperledger allows an application developer to choose an appropriate approach per partition, based on the application needs.

Since the general state may be of an arbitrarily large size, a question of limiting the state size arises. This challenge is mitigated in permissioned blockchains by the ability to have a tighter control over the behavior of each particular user and application. In public blockchains, on the other hand, the users need to be disincentivized from storing too large of a state. This is achieved in Ethereum through the gas spending mechanisms mentioned in Section 3.2: the users pay gas not only for transaction execution but also for state storage, proportionally to the state size.

The per-account state in Ethereum includes the user balance as well as any additional variables required for general calculations. In the above example, a transaction transferring four tokens from Alice to Bob would need to check Alice's balance, decrease it, and add the funds to Bob's balance.

A single user may receive many token transfers in the UTXO approach and may own many unspent TXOs simultaneously. This means that when limited to applications that transfer currency tokens, the blockchain state may grow with the number of transactions over time, whereas in the account-based approach, the state size only depends on the number of users. In this scenario, the account-based approach may require less storage compared to the UTXO approach. Besides, a single UTXO-based transaction may take many input TXOs and may produce many output TXOs. This makes the verification protocol less efficient compared to the simple verification procedure in the account-based approach.

On the other hand, the UTXO approach may allow for better parallelization of transaction processing. For example, if Alice wants to transfer one UTXO to Bob and another to Carol, the two transfers can be handled in parallel. In contrast, Alice's balance would need to be checked sequentially in the account-based model, which would require serialization of the two transfers.

The UTXO approach also has an edge when it comes to privacy, specifically hiding the link between the user and her transactions and balance. In the account-based approach, the system maintains the user's balance and keeps explicit association between the transactions and accounts. No such association is maintained in the UTXO approach so that the state cannot be linked to an individual user.

As mentioned in the beginning of this section, Fabric does not provide any implementation for state tracking that would be included in the installation. The application developer, however, can implement any state tracking approach in Fabric using the key–value store provided to this end. If a financial application needs to maintain user accounts with corresponding balances, the account name can be used as a key while the balance will be included in the value. The UTXO model can be implemented in the key–value store of Fabric as follows [41]. First, every UTXO can be represented as a unique key–value entry in the store. Any unique identifier created for the UTXO can be used as a key. The value will specify (a) the amount of cryptocurrency that the UTXO holds and (b) the reference to the owner of the UTXO, which can be represented in different ways, such as the public key or the Fabric identity. Any transfer transaction will spend old UTXOs and destroy their entries in the store. It will also create new entries, one for each new UTXO.

4.2.2. Organization of blockchain storage

The storage organization is conceptually similar in Bitcoin, Ethereum, and Fabric. The transactions are grouped together into blocks to improve scalability of the consensus protocol as explained in Section 4.4. The blocks are stored in files on disk. To provide tamper resistance, the individual blocks are linked to each other to form the logical structure of a chain. Since each block includes a hash pointer to the previous block in the chain, tampering with one block would affect the hash values in all subsequent blocks in the chain.

In addition to the chain of blocks, the storage includes a key–value store for keeping auxiliary information, e.g., related to the system state as explained in Section 4.2.1. An increased emphasis on the state storage and state-related operations (such as search) in Ethereum results in significantly more complex storage mechanisms compared to Bitcoin. Besides, the balance of roles between the chain of blocks and the key–value store is fundamentally different in Bitcoin and Ethereum because a key–value store lends itself better to state-related operations. In Bitcoin, search for blocks and even transactions is conducted primarily using the chain of blocks, with the help of a block index maintained in the key–value store. In Ethereum, on the other hand, block headers and transactions are duplicated in the key–value store, which is the main storage element used in the search. Furthermore, much of the data kept in the key–value store is organized in cryptographic search trees (henceforward referred to as cryptographic tries), while Bitcoin only constructs trees in memory and for a different purpose, as we elaborate upon in Section 4.2.6.

IOTA is different from Bitcoin, Ethereum, and Fabric in three respects: there are no blocks, the structure is different from the linked list, and the transactions themselves are stored in a database rather than a ledger. As opposed to the chain of blocks, IOTA utilizes a directed acyclic graph (DAG). In essence, the vertices represent transactions while the edges signify approval of the transactions. A new transaction has to verify and approve two existing transactions in order to be included in the DAG. Accordingly, the consensus algorithm significantly differs as we explain in Section 4.4.4. When it comes to storage, DAG storage requires more space compared to systems that use a chain of blocks. This is because there are more vertices in the DAG than blocks in a chain, and every vertex has non-trivial meta-information attached to it.

4.2.3. The block structure in Bitcoin, Ethereum, and Fabric

The key differences in the block storage structures in Bitcoin, Ethereum, and Fabric are mainly due to two important design choices: (a) record keeping model and (b) consensus protocols. As the account model requires more storage to represent the association between the accounts and the balances, the block structures for account models are relatively complex compared to those for UTXO. As we detail in Section 4.2.5, Ethereum has to keep track of the states and changes associated with each account in contrast to UTXO-based Bitcoin.

The block body in all three systems includes a list of transactions. In Ethereum, it additionally includes a list of uncles (whose concept is explained in Section 4.4.2).

The block header in all three systems contains the parent block hash. This hash is inherent to all blockchain systems that create a chain of blocks and presents the cornerstone for the immutability property. The hash is only calculated after the block formation has been completed; thus, a hash of the block cannot be included in the block itself. Instead, Bitcoin, Ethereum, and Fabric nodes compute the block hash when they receive the block from the network. Fabric nodes, however, additionally store the hash of the block data in the block header.

Nonce is another important field in the block header of blockchain systems that use mining such as Bitcoin, Ethereum, and IOTA; it contains a solution for the block cryptopuzzle. In Ethereum, the header additionally includes a hash of the uncle list stored in the block body, as explained above. Furthermore, Ethereum stores the root hash for each of the multiple search tries in the block header, as we explain in Section 4.2.6.

Considering the permissioned nature of Hyperledger, the block storage structures in the system include the real identity and signatures of the clients, endorsing peers, and the orderer. Such identities are not present in Bitcoin, Ethereum, and IOTA. Furthermore, invalid transactions may be included as part of the confirmed block due to the working principles of the consensus protocol of the Hyperledger as explained in Section 4.4. Therefore, there is a filter flag for each stored transaction, which is used to differentiate between valid and invalid transactions in the block.

4.2.4. Structure of transactions and their grouping in a block

The maximum number of transactions that can be grouped together is determined by the maximum size of the block. For example, in Bitcoin, the maximum size of the block is fixed by the protocol at 1MB. The transaction size in Bitcoin varies since a UTXO transaction can have multiple inputs and multiple outputs. Unlike Bitcoin, Ethereum uses the concept of gas to determine the size of the block. Every transaction in the block has a gas price and the sum of prices for all transactions in a block should not exceed the maximum gas limit of the block set by Ethereum miners. In the case of Hyperledger, the maximum block size can be configured by the administrator of the network, in accordance with the application requirements.

While not using blocks, IOTA utilizes the concept of bundles. A bundle is a group of transactions that are tied to each other. The need for bundles arises because, unlike Bitcoin, transactions in IOTA are limited in terms of the maximum allowed number of inputs and outputs. Since proposing each transaction in a bundle requires a separate proof-of-work puzzle (see Section 4.4.5), these limits translate to the maximum number of inputs and outputs permitted for each puzzle; longer bundles would thus require more work. Thus, if Alice wants to consolidate funds from multiple addresses and transfer those funds to several different users, she would need to create a bundle with multiple transactions. Thus, the entire bundle is issued by the same client. The size of the bundle is determined by the input; it is unpredictable and beyond the system control. Furthermore, IOTA does not have a predefined limit on the number of transactions in the bundle.

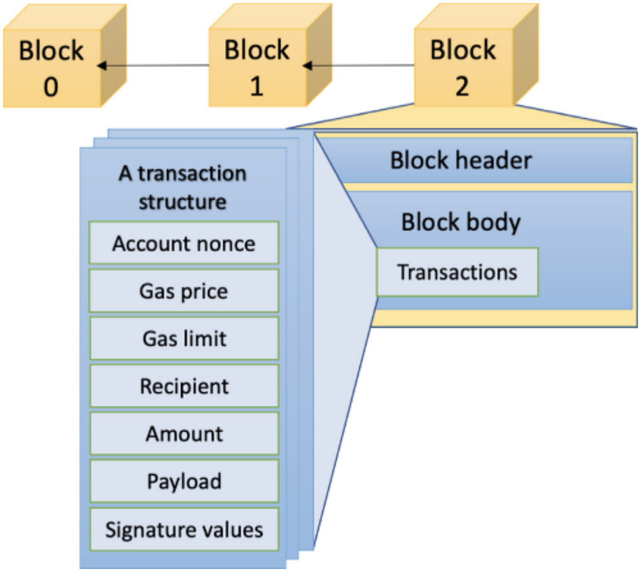

Fig. 3. An Ethereum transaction structure.

In the IOTA implementation presented in this survey [42], the bundles can be used as a vertex in the DAG as opposed to individual transactions.

Bitcoin transaction structure is described both in the specifications [43] and textbooks [22] in detail. Its main fields are the list of inputs and the list of outputs common in the UTXO model as explained in Section 4.2.1. As Ethereum does not follow the UTXO model, it has a different transaction structure. As shown in Fig. 3, the transaction structure in Ethereum includes the following information:

Account nonce (called *nonce* in Ethereum code [44]): An Ethereum node issuing transactions on behalf of the user maintains a counter of the number of transactions sent from the user's account. When the transaction is issued, the node sets account nonce to the current value of the counter. Keeping track of this count over time is important for two reasons: first, it establishes sequential order between transactions sending funds from the same account so that the miners or validators will process the transactions in that order. Second, the account nonce prevents transaction replay attacks by the recipient of funds because a repeated transaction with the same nonce will be detected as invalid.

Gas price (called *gas_price* in Ethereum code) is the price of each unit of gas, expressed in Ether.

Gas limit (called *gas* in Ethereum code) is the maximum amount of gas that can be consumed by executing this transaction.

Recipient (called *to* in Ethereum code) is the address of the transfer receiver.

Amount (called *value* in Ethereum code) is the value to be transferred to the recipient.

Payload (called *data* in Ethereum code): If the transaction is meant to be an execution of a contract (the recipient is a contract account), the payload field would be a message that identifies the function and argument values of the contract. Otherwise, if the transaction is for payment (the recipient is a user account), the payload field would be empty.

Signature values: This field includes components of the signature algorithm of the sender. Ethereum transactions use ECDSA (Elliptic Curve Digital Signature Algorithm) [45] as their digital signature for verification. The signature of a transaction confirms that the sender has authorized this transaction.

In Hyperledger Fabric, the design of the transaction structure depends on the application and state tracking approach chosen for the specific deployment.

4.2.5. On-disk storage

The most important element stored on the disk is the ledger. Bitcoin, Ethereum, and Hyperledger Fabric store their block data as multiple files, each of which is limited in size. Although the file size limit is fixed in Bitcoin and Ethereum, it is configurable in Hyperledger Fabric [46]. Block files store the blocks appended to the local copy of the ledger [47].

Bitcoin additionally stores block indexes on the disk to keep track of the information available in the chain of blocks. These indexes are stored in a LevelDB (key–value store) database. Similar to Bitcoin, Hyperledger Fabric utilizes indexes. By default, Fabric also stores indexes in LevelDB but it can be replaced by other databases. The primary purpose of the indexes is to support search functionalities for blocks and transactions stored in the file through various keys such as a block number, block hash, transaction id, and transaction number. The values of the LevelDB key–value store are corresponding file location pointers to the chain of block files [47,48]. When an API call to access blocks and transactions of the blockchain occurs, these search indexes are utilized to find the actual data location in the file. On the other hand, Ethereum relies on a database for keeping track of changes and finding data items, so that has no need for indexing the files.

State-related storage is another salient on-disk element. Bitcoin, Ethereum, and Hyperledger Fabric use a database of key–value pairs for keeping track of the latest blockchain state. In Bitcoin, the UTXO set which stores all unspent outputs is all information that is required to validate a new transaction without the need to traverse the whole blockchain [49]. Bitcoin keeps the UTXO set in a LevelDB storage called *chainstate*. On the other hand, Hyperledger Fabric keeps track of the latest values of the business assets in its state-related storage called *worldstate*. The state-related storage of Hyperledger Fabric is represented on the disk as a LevelDB key–value store by default. The keys and values of the state-related storage reflect the data model described in the application *chaincode*, as defined by the developer (see Section 4.5). In Fabric, if the business application requires a complex data model and access pattern, various other databases such as CouchDB, GraphDB, and a relational datastore can be used to represent the worldstate instead of LevelDB. These alternatives support rich queries and data types [50]. Similarly, Ethereum stores the latest values of all the accounts in a LevelDB database as the state-related storage to facilitate calculating the current value of any account available in the ledger. For this purpose, Ethereum constructs the following four tries:

State trie represents the global state that efficiently stores the mapping between all the addresses and accounts.

Storage trie is responsible for maintaining the relationship between the account and the corresponding balance. State trie is linked to the storage trie.

Transactions trie represents the transactions that change the state of the Ethereum.

Receipts trie represents the outcome of the successful execution of transactions.

Individual systems additionally maintain auxiliary meta-data stored on the disk. In Bitcoin, each block file has a corresponding undo file which contains the information that is necessary to roll back and remove a block from the blockchain in the event of a reorganization/fork [47]. In Ethereum, in addition to storing block data as files, all block data including the block header and the block body is stored in a LevelDB database as shown in Fig. 4.

Due to the different organization of blockchain storage in IOTA as described in Section 4.2.2, Tangle is stored in a database rather

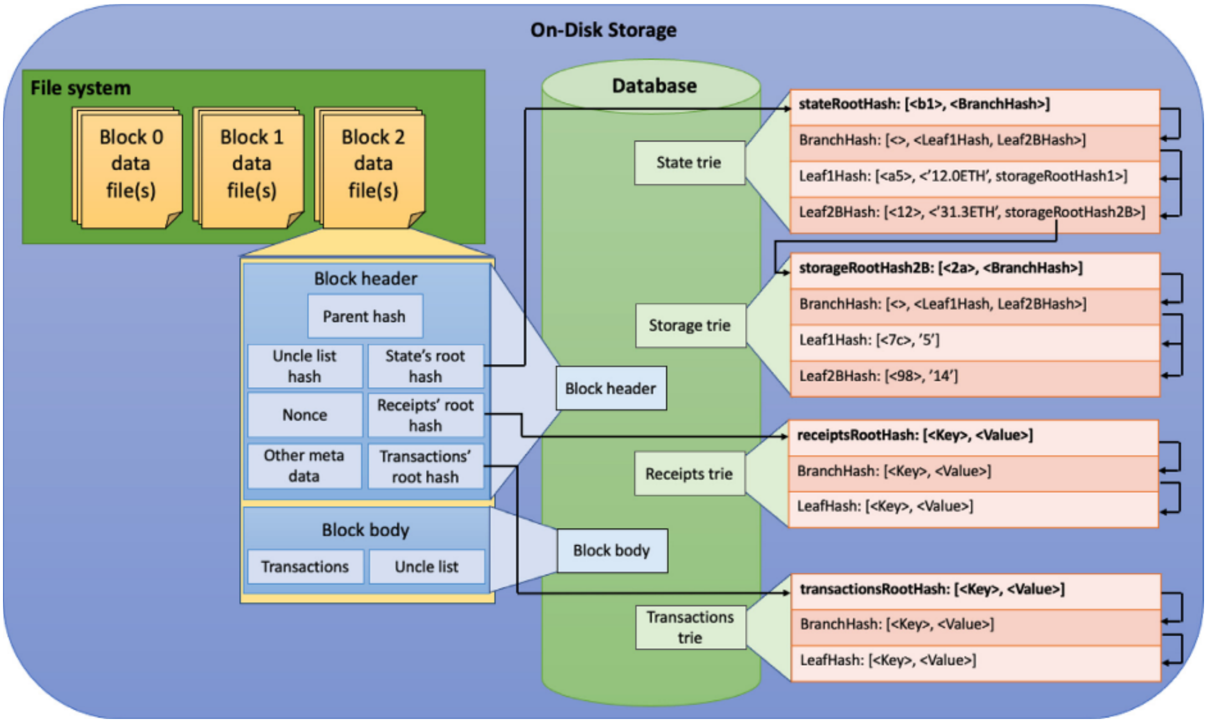
On-Disk Storage
File system
Block 0 data file(s)
Block 1 data file(s)
Block 2 data file(s)
Database
Block header
Parent hash
Uncle list hash
State's root hash
Nonce
Receipts' root hash
Other meta data
Transactions' root hash
Block body
Transactions
Uncle list
State trie
Storage trie
Block header
Receipts trie
Block body
Transactions trie
stateRootHash: [<b1>, <BranchHash>]
BranchHash: [<>, <Leaf1Hash, Leaf2BHash>]
Leaf1Hash: [<a5>, <'12.0ETH', storageRootHash1>]
Leaf2BHash: [<12>, <'31.3ETH', storageRootHash2B>]
storageRootHash2B: [<2a>, <BranchHash>]
BranchHash: [<>, <Leaf1Hash, Leaf2BHash>]
Leaf1Hash: [<7c>, '5']
Leaf2BHash: [<98>, '14']
receiptsRootHash: [<Key>, <Value>]
BranchHash: [<Key>, <Value>]
LeafHash: [<Key>, <Value>]
transactionsRootHash: [<Key>, <Value>]
BranchHash: [<Key>, <Value>]
LeafHash: [<Key>, <Value>]

Fig. 4. On-disk storage of Ethereum consists of the block data and database records.

than in files. Thus, all the information about both transactions and balances is kept in the RocksDB database. As an extra element, an IOTA node keeps snapshot data (explained in Section 4.2.8) in a file. Each time a new snapshot arrives, it removes all transactions from the Tangle, i.e. RocksDB database, and moves only non-zero balances to a file in order to be used by the new pruned Tangle. Thus, similarly to Hyperledger Fabric, IOTA maintains only the current balances associated with the addresses.

4.2.6. On the use of trees in Bitcoin and Ethereum

Bitcoin and Ethereum use trees to organize data items (transactions and in the case of Ethereum, other data elements related to state tracking) in the context of a given block. The trees are cryptographic: every non-leaf vertex stores a secure hash value for each of its child vertices. This way, tampering with a single tree vertex *n*, or inserting a new vertex *n* will lead to modification of all the hashes on the path from *n* to the root of the tree, making it easily detectable. Due to this, the hash of the root vertex becomes a cryptographic fingerprint of the entire tree structure. Because of its significance, the hash of every root vertex is kept in the corresponding block of Bitcoin or Ethereum for the sake of verification.

However, Bitcoin and Ethereum use trees for different purposes, and accordingly, their implementation is quite different. Bitcoin uses binary Merkle trees as shown in Fig. 5. For a given block, the tree is constructed as follows: each leaf represents a block transaction. The order of leaves corresponds to the deterministic order in which the transactions are listed in the block body. Each non-leaf vertex is created by computing a secure hash of the concatenation of the two child values. Due to this construction, the tree is deterministically reproducible from the block body stored on the disk. This is important because Bitcoin constructs Merkle trees in memory on demand.

The main purpose of utilizing Merkle trees in Bitcoin is to minimize the amount of bandwidth used for disseminating the proof that a given transaction is included in a given block. It is a typical situation that a Bitcoin lightweight node, without the knowledge of the ledger, inquires about the inclusion status of a particular transaction in a new block. To this end, the node sends a verification request including the transaction hash to a full node which keeps track of the ledger. The full node first needs to locate the Merkle tree of the block in memory and then, sequentially scan the leaves of the tree in order to verify the inclusion. While a sequential scan is not an efficient operation, its inefficiency does not have a strong impact because the number of transactions in a block is relatively small. However, once the transaction is found, the full node can construct the Merkle proof efficiently [51]. The Merkle proof includes all vertices on the path from the leaf with the transaction to the root of the tree, along with the children of these vertices. The full node can then send the Merkle proof back to the lightweight node. The crucial point is that the lightweight node can verify the inclusion based on the Merkle proof, without requiring the rest of the vertices in the Merkle tree. The size of the Merkle proof is logarithmic with the size of the Merkle tree.

In addition to constructing Merkle proofs, there are in fact very few operations supported by Bitcoin Merkle trees. Miners benefit from being able to efficiently append new leaves with transactions because new transactions can arrive while mining a block. Merkle trees allow for efficiently implementing this particular type of vertex insertion because a new transaction becomes the rightmost leaf. Additionally, the trees can be pruned: for example, if we do not need to verify the inclusion of Transactions 1 and 2 in Fig. 5 any longer because their TXOs have already been spent, we can remove corresponding leaves, along with their hash vertices. We do need to keep the vertex with *Hash(Hash(TX1)+Hash(TX2))*, however, for the purpose of constructing Merkle proofs, as explained above. On the other hand, vertex search, update, and deletion are neither needed nor efficiently supported.

This is in a stark contrast with Ethereum, which uses the trees for storing the state, in addition to transactions. Unlike transaction history which is fixed, the state in Ethereum needs to be frequently changed: new accounts are inserted and old accounts are deleted. The state of a particular account needs to be searched and updated as well, potentially often. Ethereum uses

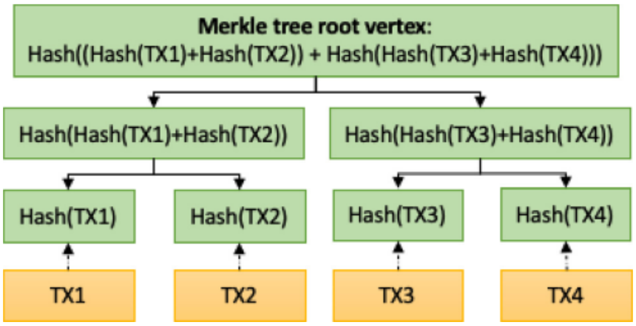
Merkle tree root vertex:
Hash((Hash(TX1)+Hash(TX2)) + Hash(Hash(TX3)+Hash(TX4)))
Hash(Hash(TX1)+Hash(TX2))
Hash(Hash(TX3)+Hash(TX4))
Hash(TX1)
Hash(TX2)
Hash(TX3)
Hash(TX4)
TX1
TX2
TX3
TX4

Fig. 5. Bitcoin's transaction Merkle tree.

Merkle Patricia trie, which provides logarithmic complexity for search, insertion, update, and deletion, in addition to efficient construction of Merkle proofs. Similarly to the Merkle tree, data items are stored in the leaves. However, Merkle Patricia trie is a prefix tree that uses hashing mechanisms to produce a key string that determines the path of the data item within the tree. Normally, prefix trees create a new tree level for each character in the key: the first character will determine the branch at the root, the second character will determine the branch at the next level, and so on. However, Merkle Patricia trie is a compact radix tree: if there are two vertices with keys "*abcd*" and "*abef*" respectively, an intermediate vertex with the key "*ab*" will be created. This vertex will have two direct children corresponding to "*abcd*" and "*abef*", without any additional intermediate levels. In Ethereum, the key is encoded as a hexadecimal value so that each vertex has at most 16 children.

The key string that determines the path is produced differently for each trie type. For the state trie, each leaf corresponds to an account in Ethereum. The key string is derived from the hash of the account address, while the value of the leaf includes the balance and the root hash for the storage trie of the account. For the storage trie, each leaf represents an Ethereum contract's data. The key string is derived from the hash of the Ethereum contract's address, while the values reflect the data model described in the Ethereum smart contract. An updated state of an Ethereum account is retrievable by traversing the storage trie vertices in the LevelDB database.

As shown in Fig. 4, each vertex of every Ethereum Merkle Patricia trie is stored as a separate record in the LevelDB storage. The key for the record is derived from the hash of the trie vertex and stored in the parent vertex record. Since the hash for the root vertex is stored in the block header, it is used as the starting point for finding other trie vertices in the database, which allows for an efficient traversal of the entire trie.

Fig. 4 provides an illustration for the storage organization on disk in Ethereum. Header and body for *Block 2* are stored in both *Block 2* data file(s) and in the database. The header contains root hashes for the state, receipt, and transaction tries. Each root hash is a key for the corresponding root vertex record in the database. From the root record of the state trie, we can reach the branch vertex record, which points to *Leaves 1* and *2B*, each of which corresponding to a different account. The record of *Account 2B* points to the root of the storage trie for that account. The pointers between the vertices of the storage trie in the database are presented in a similar way.

Another interesting functionality of Ethereum is that it continues maintaining correct tries for old blocks, not just for the most recent block. For example, we can query the value of an account or the value of a variable (maintained by a smart contract) in a past block, even if those values have later been updated. This functionality allows for an efficient verification of transactions of block n: when applied on the state of block $n-1$, these transactions, if valid, will update the state to that of block n. To implement this functionality in a space-efficient fashion, Ethereum uses an interesting variation of the copy-on-write technique, as illustrated in Fig. 6. While the transaction and receipt tries are strictly separate for each block, the state and storage tries can be partially shared across the blocks. In the illustration, the key for the state record of *Account 1* in *Block 1* hashes to value *b10a5*. The record is placed in *Leaf 1*, which is the leftmost child of the branch vertex. Similarly, the key for the state record of *Account 2* in *Block 1* hashes to value *b1d14* and it is placed in *Leaf 2A*. The state of *Account 1* has not been modified by the transactions of *Block 2*. Therefore, the branch vertex in *Block 2* points to the same record of *Leaf 1* as the branch vertex in *Block 1*. On the other hand, the state of *Account 2* has been modified by the transactions of *Block 2*. Since the path to the state of *Account 2* is the hash of the account address i.e., *b1d14*, Ethereum creates a new record called *Leaf 2B* in the same path under the branch vertex in *Block 2* to update the balance. Due to the creation of a new leaf record, Ethereum also creates new branch and state root vertices in *Block 2*. Note that by starting from the old root vertex in *Block 1*, we can still find the old state of *Account 2*.

Storage tries are updated in a similar way. As a result, the state and storage tries are only partially updated due to the transactions in every new block. In reality, the changes in the state and storage tries of each Ethereum block are fairly small [52], which makes the copy-on-write technique described above particularly effective.

4.2.7. *In-memory storage*

Bitcoin creates Merkle trees in memory on demand, when a need arises (see Section 4.2.6). As mentioned in Section 4.2.6, the trees are deterministically reproducible in memory from transactions stored on disk. Furthermore, in Bitcoin, the entire LevelDB database of block indexes is loaded into memory on startup to achieve a better performance of finding the blocks [47]. UTXO cache is another critical data structure maintained in Bitcoin memory [53]. As the UTXO set stored in LevelDB is accessed frequently during block validation, the access time becomes a major bottleneck. Therefore, during the block validation, the required unspent coins are pulled from LevelDB into an in-memory cache. Since the size of the entire UTXO set exceeds 8 GB, cache management is required.

In Ethereum, different node implementations such as Geth maintain an in-memory cache of configurable size, designated for vertices of various tries. In the context of Hyperledger Fabric, when using a state database, read delays during endorsement and validation phases have historically been a performance bottleneck. With Fabric v2.0 [54], a new peer cache replaces many of the expensive lookups with fast local cache reads, and the cache size can be configurable. Moreover, what is stored in the memory storage of Hyperledger Fabric depends on the implementation of the application. Finally, IOTA uses Tangle Accelerator [55], which is an intermediate caching proxy server between the client and IOTA node to speed up the attachment process of a bundle to tangle.

4.2.8. *Data retention*

Data retention is performed differently in each of the representative blockchain systems. The ledger in Bitcoin grows more slowly compared to Ethereum and IOTA because Bitcoin stores less data per-block compared to Ethereum and because new blocks are appended less frequently compared to ledger changes in Ethereum or IOTA. As a result, Bitcoin is less in need for ledger pruning mechanisms. For the time being, Bitcoin does not

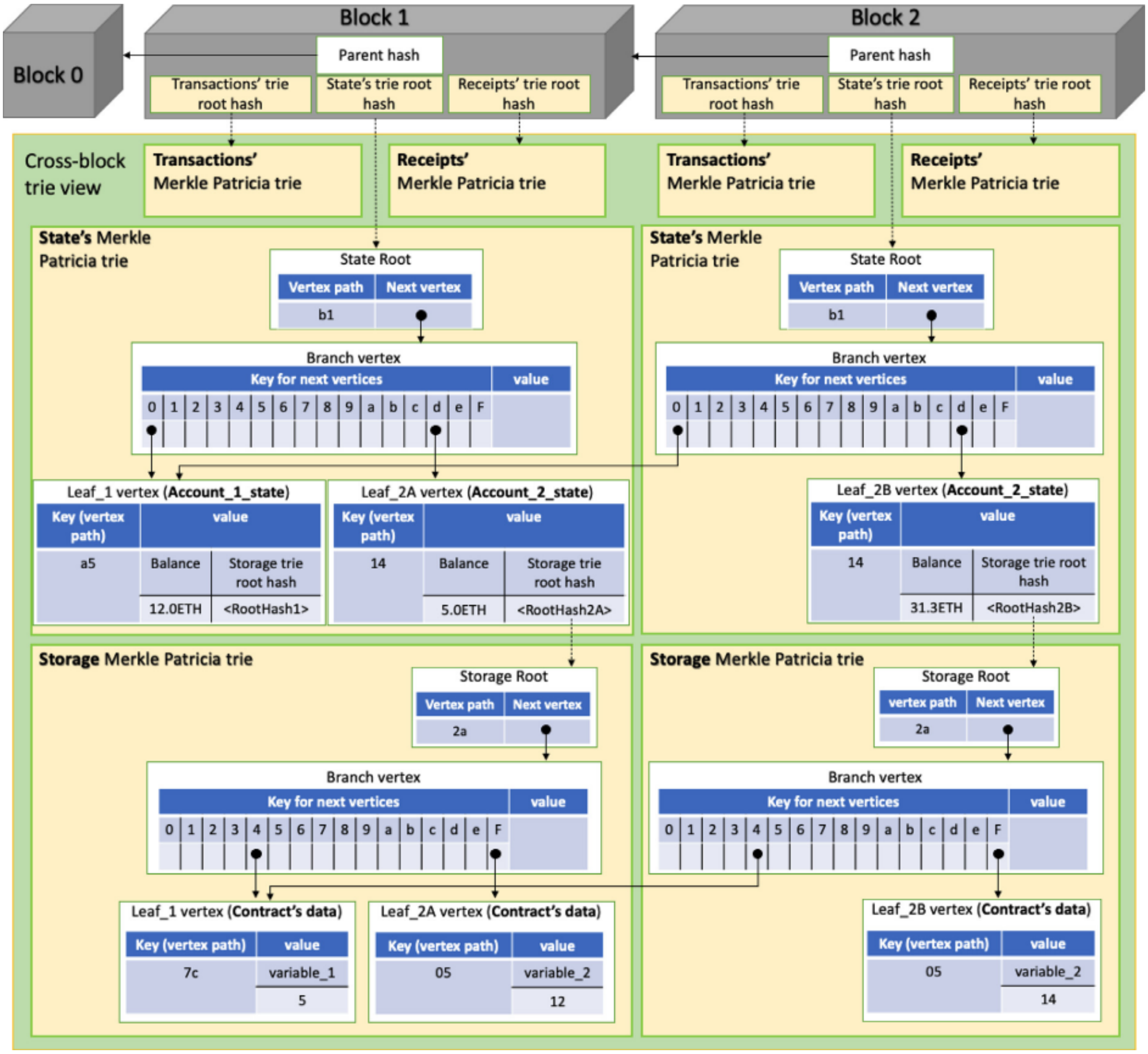

Fig. 6. Constructing tries across multiple blocks in Ethereum.

have any data retention policy and keeps all of its transactions forever. As a consequence, the storage on Bitcoin full nodes can potentially grow large in size over time, which may become an even bigger issue if Bitcoin manages to overcome its scalability barrier and raise the average rate of transactions.

Similar to Bitcoin, Ethereum keeps its transactions forever in the ledger. However, Ethereum has a state trie pruning mechanism [52] for obsolete states. This mechanism tracks when trie vertices are no longer referenced by the state trie (for example, the *Leaf2A* vertex in Fig. 6 has dropped out of the state trie in *Block 2*), and at that point places the dropped vertices on a death row in the database. From this point, after 5000 new blocks have been appended, the vertex will be permanently deleted from the database. Essentially, Ethereum stores the trie vertices that are part of the current state or part of the recent history [52] (up to 5000 recent blocks).

Similar to Bitcoin and Ethereum, transactions on the ledger will never be deleted in Hyperledger Fabric. However, data from the worldstate database can be deleted if an application developer defines the procedure of pruning obsolete data in the implementation.

IOTA provides snapshots regularly in order to reduce the size of storage needed by nodes. The snapshot process removes all the transaction history and the addresses with zero balances and creates the list of addresses with a non-zero balance to reduce the Tangle size [56]. This in turn speeds up the time required to do synchronization among nodes. Another advantage is that the snapshot can indirectly improve the throughput of transactions in IOTA by increasing the speed of computing the balance for the accounts. Considering the nature of permanodes (refer to Section 3.4), the snapshot process does not have any impact on their storage.

4.3. Communication layer

Comparison between the communication layer features of the surveyed blockchain systems is shown in Table 4. Only full nodes are part of the blockchain network in Bitcoin; they receive all blocks and transactions by running the dissemination protocol. Lightweight nodes connect to the full node of their choice. Unlike full nodes, lightweight nodes only receive a subset of transactions, filtered for them by the full nodes to which they are connected [57]. Furthermore, lightweight nodes rely on full nodes for propagating transactions to the network. In terms of receiving blocks, lightweight nodes download only the block headers from

Table 4
Communication layer of different blockchain systems.

Features	Bitcoin	Ethereum	Hyperledger Fabric	IOTA
Granularity of dissemination	Whole network	Whole network	Per channel	Whole network
Entities forming the network	Full nodes	Full nodes	Orderers and peers	Nodes
Communication protocol	Inventory push-gossiped & blocks pulled by full nodes	Blocks or block hashes push-gossiped & block headers and bodies pulled	leader pulls blocks from orderers & push-gossips to peers	Push-flooding
Mean time to receive a block	About 12.6 s	About 109 ms	Application-dependent	No studies conducted
Ordering guarantees	No guarantees	No guarantees	Totally ordered broadcast	No guarantees
Privacy & security guarantees	No guarantees	Proprietary deployment can enable encrypted & authenticated messages	Authenticated channels	Authenticated and encrypted messages, partial anonymity
Initial peer discovery	Through a set of DNS seeds or direct conn. to a known full node	Through a set of bootnodes or direct conn. to a known full node	Anchor peering & channel membership	Manual conn. through a node list
Geo-proximity in the network	135 ms (avg.) peer-to-peer latency	171 ms (avg.) peer-to-peer latency	Depends on consortium network topology	No studies conducted

the full nodes. Ethereum is absolutely identical to Bitcoin in terms of network formation and the division between full and lightweight nodes. Fabric and IOTA are also quite similar in this regard: peers constitute the peer-to-peer network in Fabric while clients connect to the peers which selectively relay transactions and blocks from and to the clients. In IOTA, clients connect to the network of IOTA nodes, the latter relaying bundles from and to the clients.

In order to manage the *granularity of dissemination*, Hyperledger Fabric is using the concept of channels. Transactions submitted and propagated in a channel are isolated from the other channels. The channels are independent of each other in terms of the order, delivery, and processing of the transactions. On the other hand, transactions in Bitcoin, Ethereum and IOTA are propagated through the whole network.

Next, we consider the *communication protocol* and *propagation time* for each of the four systems. In Bitcoin, each full node sends inventory messages (*inv*) [58] periodically to advertise its knowledge of transactions or blocks to the neighbors in the network. This advertisement contains the hash value of the transactions or blocks as their identifiers, which is much shorter than the actual transaction or block. Then, each neighbor that receives the advertisement checks if it mentions knowledge of transactions or blocks the neighbor has not seen before. In this case, the neighbor requests the missing transactions or blocks from the sender via the *getData* message [58].

Experiments in [59] indicate that it takes about 40 s for a new block to propagate to 95% of the Bitcoin network (the mean time to receive a block for a full node is 12.6 s). This time includes the transmission time (for *inv* message, *getData* message, and delivery), as well as the verification time. The propagation time is likely to be even shorter for the transactions compared to blocks.

In Ethereum, a miner that creates a block or another full node that receives a new block from the network use the same propagation protocol. Namely, the full node considers a set S of its neighbors and picks a random subset $S' \subset S$ of size $|S'| = \sqrt{|S|}$. Then, the full node forwards the entire block to full nodes in S' while only sending a hash of the block to full nodes in $S \backslash S'$. The neighbors receiving the hash of a new block request the block header via a GetHeader message [60] from either the sender or any of their neighbors that possess the block. Finally, after getting the new block header, the Ethereum full node requests the block body in a way similar to requesting the header.

The study of [61] shows that 95% of the blocks are propagated through the Ethereum network within 211 ms (the average time of a block propagation delay is stated to be 109 ms). An experiment in [60] estimates that it takes only 3 or 4 hops to broadcast a new block to the entire Ethereum network. This small world of an Ethereum network compared to the Bitcoin network can be the reason for the lower block propagation time in Ethereum.

Peers in Hyperledger Fabric use both push and pull methods in the communication protocol [62]. In the case of block dissemination, a leader is selected from the peers. The leader becomes responsible for pulling blocks from the orderers and then initiating a push-gossip protocol to propagate the blocks to the peers: Each peer broadcasts the blocks to a random set of neighbors in the channel. In addition to push-gossiping, each peer is also responsible for selecting a number of random peers regularly and attempting to pull the missing blocks from them. The design choice to use a leader for initiating push-gossip is motivated by the need to reduce the load of sending blocks from the orderers to the network. Orderers in Hyperledger Fabric also enforce basic access control for channels, restricting who can read and write data to the channels, and who can configure them [63]. In Fabric, the propagation time is dependent on application characteristics, especially on the transactions type and network size.

In IOTA, the transactions are flooded throughout the network similarly to Ethereum. On the other hand, since no studies have been conducted for the propagation time in IOTA, it can be considered an interesting subject for future research.

Regarding the *ordering guarantees*, Hyperledger Fabric is the only system among the blockchain systems covered in this survey which supports total-order broadcasting [64] within each communication channel. This guarantee ensures that different endorsers and orderers receive transactions from each given client ordered in the same way relatively to the transactions from other clients. The relative order of transactions is important because of the effects that the transactions may have on each other.

To support *privacy and security guarantees*, transferred transactions between the Ethereum nodes are encrypted and authenticated [65], which makes the Ethereum communication layer more secure compared to Bitcoin. Furthermore, Whisper [66], a configurable messaging protocol, which can be enabled in proprietary Ethereum deployments, provides the benefit of hiding the location of sender and receiver in the network, if such privacy

Table 5
Data manipulation layer of different blockchain systems.

Features	**Bitcoin**	**Ethereum**	**Hyperledger Fabric**	**IOTA**
Consensus protocol	Chain convergence using longest-chain rule	Chain convergence using GHOST protocol	Configurable consensus module (Raft is the default)	DAG convergence using tip selection algorithm
Mining difficulty	About 10 min to mine a block	About 12–15 s to mine a block	No mining	No mining
Throughput (tps)	3–7	About 15	Configuration-dependent	About 50
Mining power utilization	Above 99%	Below 97%	No mining	No mining
Transaction confirmation	Probabilistic based on the number of blocks	Probabilistic based on the number of blocks	Deterministic, after committing on the peers	Probabilistic based on confirmation confidence
Mitigating DoS attacks	PoW	PoW + Gas price	Permissioned authentication	PoW
Rich search functionality	No	No	SQL-like query	No

requirements arise. Whisper can achieve this benefit by using onion routing and other techniques common in the Tor [67] network. Hyperledger Fabric utilizes the authenticated channels to provide the privacy and security of the network. IOTA has also introduced the functionality of issuing and accessing encrypted data streams by implementing the Masked Authenticated Messaging (MAM) protocol [68] as a second layer communication protocol to improve the privacy and security of the communication layer.

Geographical proximity between participants in the blockchain network is one of the parameters that have effect on the block propagation time throughout the network. Measurements in [65] indicate that Bitcoin exhibits a higher degree of clustering compared to Ethereum, i.e., it has many more groups of full nodes that are geographically close to each other. The measurements also show that an estimated peer-to-peer latency between Ethereum full nodes is 26.7% higher on average compared to Bitcoin. The main reason for this higher clustering degree is that many Bitcoin full nodes are running in data centers. This research claims that 56% of Bitcoin full nodes belong to autonomous systems that provide dedicated hosting services whilst the percentage for Ethereum full nodes is 28%. On the other hand, in permissioned blockchains such as Hyperledger Fabric, geographical proximity between peers depends on the consortium network topology. To the best of our knowledge, geographical proximity between IOTA nodes has not been studied yet.

4.4. Data manipulation layer

Data manipulation in blockchain primarily consists of two operations: updating the state of the blockchain and querying it. Since the transactions stored in the blockchain are immutable, the only update operation allowed is adding a new transaction to the ledger, which additionally results in performing bookkeeping needed to keep track of the updated state. While the goal is simple, the update process is quite complex because it requires running a consensus protocol across the participants in the blockchain network. The blockchain type and even design goals vary across the systems, which results in significant variations in the flavor of consensus protocols used [30,69]. This is the reason why blockchain systems exhibit significant variations in their performance and in security assumptions and threats they can withstand. Besides, the performance is affected by deployment characteristics, such as the blockchain network, transaction rate, duration of transaction execution, and the distribution of mining power in systems that use mining. Accordingly, most of this section is dedicated to the update protocol in the four systems considered. For each system, we also discuss the security and performance properties. Table 5 shows the differences in the features of the representative blockchain systems in the data manipulation layer.

Querying the blockchain is conceptually much simpler than updating it because it does not require coordination across the participants in the blockchain network and because the scalability and security constraints are not nearly as severe. However, the differences in blockchain types and stored data (see Section 4.2) lead to different query models.

Finally, security plays a very important role in the design of the data manipulation layer. Yet, different blockchain systems exhibit differences in the security assumptions and tolerated threats so that there exist no easily identifiable baseline to compare blockchain systems against. For example, the issues of branching and concentration of computing power in the hands of a single participant (or a group of colluding participants) is a major consideration for Bitcoin and Ethereum. However, it is a non-issue for Hyperledger Fabric which does not use PoW and where the branching is impossible due to the consensus protocol being deterministic. The security model behind IOTA has only been partially explored and defined. Accordingly, we discuss security aspects of each system separately in the context of presenting the consensus protocol. However, all blockchain systems must be resilient to denial-of-service (DoS) attacks, which is why we include a focused discussion comparing the defense mechanisms to DoS attacks in all four systems.

4.4.1. Updating blockchain in Bitcoin

The mechanism of updating the blockchain in Bitcoin has been presented in other survey works (see Section 5) and in a textbook [22]. In this section, we provide an extended description written in a pedagogical way. While containing limited novelty by itself, this description serves as a basis for comparison with the more novel descriptions of updating the blockchain in the other three systems, presented in the later sections.

The general flow of updating the blockchain can be found in Section 3.1. What is especially interesting about the blockchain update solution in open blockchain such as Bitcoin is that it is fully self-moderated: the nodes are fully autonomous. They do not have to follow the complex protocol but do so because they are given a combination of incentives and protective mechanisms. For example, the consensus protocol is very sensitive to the rate of proposals: too many and the protocol stops to scale; too few and the system does not make any progress.

The PoW mechanism prevents the miners from issuing proposals at too high of a rate. The rate is regulated by the complexity of the cryptopuzzle, called *mining complexity*. Bitcoin adjusts the mining complexity every 2016 blocks so as to keep the average interval between consecutive block proposals limited to 10 min. To be precise, every miner in Bitcoin adjusts its own mining complexity independently based on its local copy of the chain. However, since the adjustment is based on the height of a block and not on the data in the chain, every miner performs the same adjustment after the same block number, which ensures fairness of mining.

In order to keep the rate up, Bitcoin incentivizes miners to spend their resources on solving cryptopuzzles. Specifically, it introduces two incentive mechanisms: (a) a block creation reward that is given to a miner whose block has been successfully included into the chain, and (b) a transaction inclusion fee that a transaction creator may voluntarily pay to the miner that includes the transaction into the proposed block. The block creation reward is regulated by Bitcoin similarly to the mining complexity. It is halved every 210 000 blocks. Taking into account the average 10 min interval between consecutive blocks, the reward is expected to become zero before 2140. Currently, the block creation reward plays a more important role than a transaction inclusion fee but as the reward gets smaller, the transaction inclusion fee will start playing an increasingly more important role. Every block includes a single transaction assigning block creation reward to the proposer; this is the only mechanism by which Bitcoin injects new funds into circulation. In order to increase the chance of receiving block creation reward, a group of miners can organize themselves into a *mining pool*. Then, all members of the pool work together to mine each block, and the revenue of mining a block is shared among them.

In Bitcoin, there is no entity in the system that knows the precise state of the updated global chain. Every full node is trying to approximate the updated knowledge of the global chain by maintaining a local copy of it (ledger and UTXO information as explained in Section 4.2); the full node may update the local copy upon receiving a new block proposal. As different Bitcoin miners may create and propagate different valid block proposals in parallel, the local ledger copies on different full nodes may temporarily diverge. In the example of Fig. 7, Block 2a and Block 2b are both valid and created in parallel as successors to Block 1. This situation when different blocks are referring to the same predecessor block is called a *fork* in the blockchain. In addition to the accidental fork creation which happens because of concurrent block proposals, there are also intentional forks. Whenever the blockchain protocol is updated (for example when the block size is changed) on a subset of nodes, it can result into two types of fork: hard and soft. In the hard fork situation, the nodes with updated protocol cannot interoperate with the nodes running the old version of the protocol at all. A hard fork can lead to creation of a new cryptocurrency. For example, Bitcoin Cash [70] has been created as a hard fork of Bitcoin, and is now completely independent of Bitcoin. On the other hand, soft forks support backward compatibility. It means that transactions from nodes running the old protocol are accepted by the nodes running the new one, but transactions created by the new protocol addresses are not accepted by the old nodes. Segregated Witness (SegWit) [71] is an example of a soft fork. SegWit is created as a protocol upgrade of Bitcoin to improve block capacity and prevent the transaction malleability (changing a transaction after signing it) problem by introducing a new transaction structure.

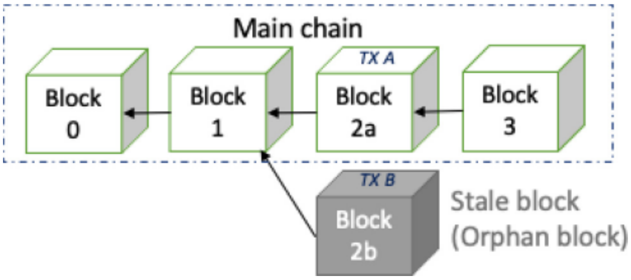

Fig. 7. Bitcoin chooses the main chain based on the longest chain rule. Any valid block outside the main chain will be ignored and called a stale block.

In the case of accidental fork creation, the miners have to decide which of the branches will be included in the main chain and which branch will be discarded. In order to reach a consensus on the included branch, Bitcoin has the longest chain rule for resolving conflicts between competing branches [32]. This rule stipulates that the longest branch wins and becomes part of the main chain for building the subsequent block upon. The rationale for this rule is that the longest chain corresponds to the biggest amount of work on the cryptographic puzzles; by including this branch and discarding shorter branches, we are wasting the least amount of work. Eventually, the local copies converge to the same global state.

In the example of Fig. 7, Block 3 points to Block 2a as its predecessor while ignoring Block 2b. This may happen because the miner creating Block 3 received 2a before 2b. In this case, Block 2b is discarded according to the longest chain rule despite being valid and verified. It is called a *stale* (sometimes called *orphan*) block [72].

It should be noticed that because the convergence of the local copies is only eventual, this complicates the reasoning about the blockchain state. A full node may attempt to infer information about the global state based on the local copy, without any consultation with other full nodes. The most common case of inference is when a full node tries to establish whether a given transaction has been included in the blockchain. The inference is correct with a certain probability, which never reaches 100%: A transaction A may be included in a block and added to the local copy, but at a later time, a different branch may become the longest and win the race, in which case A will not be included in the chain. However, the more subsequent blocks are added to the local copy after the block B_A containing A, the longer the branch in the local copy becomes, which reduces the probability that a competing branch not including B_A will become longer yet. In Fig. 7, transaction A ($TX\ A$) and transaction B ($TX\ B$) are included in Block 2a and Block 2b respectively. The inclusion probability of transaction A in the main chain is increased by creating Block 3 upon Block 2a, while, the chance of inclusion in the chain for transaction B is reduced since it is not included in the longer chain. A more precise calculation of probabilities for the finality of transaction inclusion can be found, e.g., in [73]. Bitcoin full nodes use this reasoning to provide a confirmation about the inclusion of transactions: after 6 subsequent blocks are included into the local copy, the full node considers the probability high enough to send a confirmation to the user. For example, a transaction may transfer a fee in bitcoins to a service provider.

However, a malicious miner that owns more than 50% of the network mining power, may overcome the network by mining, producing valid blocks, and building longest chains faster than the rest of the network. In particular, it gains the ability to tamper with the data and subvert the blockchain. This situation is called a 51% attack. The miner performing the attack can exploit the situation in different ways, such as ignoring some specific transactions or depriving the competing miners from receiving block creation rewards. It can also define a new policy and impose it onto the network. As an example, the exact amount of block creation reward is configurable and subject to an agreed policy. An attacker will be able to increase the block creation reward in hope that the attacker will be mining most blocks and ripping the benefits of a higher reward. Furthermore, the attacker that owns more than 50% of the network mining power can perform a *double-spending* attack, that is to transfer the same virtual coin in two different transactions. For example, an attacker creates one transaction to send some bitcoins to a recipient in exchange for a service or product. This transaction is included in Block 2b, which is appended to the blockchain. Let us denote the predecessor of Block 2b as Block 1 (see Fig. 7). After receiving the service or product from the recipient of the transaction, the attacker creates another transaction that sends the same bitcoins to a different address of his own. The powerful attacker quickly creates and mines Block 2a with the new transaction, and then creates and mines Block 3 as a successor of Block 2a, in order to ensure that Block 2a is included in the longest branch, and that Block 2b becomes stale. As a result, the attacker gets the service or product without paying any bitcoins for it.

The probability of a 51% attack being successful depends on a number of factors [74], above all those contributing to the presence of forks. The situation illustrated in Fig. 7 is undesirable from the security point of view. This is because Block 2b does not contribute to securing the main chain so that the attacker requires less mining power to subvert the chain. To capture this point, [75] has introduced the metric of mining power utilization, which is defined as the ratio between the mining power that secures the main chain and the total mining power. Mining power wasted on work that does not appear on the blockchain accounts for the difference. The higher the mining power utilization, the more mining power an attacker must have in order to subvert the chain.

The argument of mining power utilization is the main reason against reducing the complexity of PoW in Bitcoin. Even though such a reduction may improve the throughput of transactions and reduce transaction confirmation time (that is the time required for a transaction to reach a certain probability threshold for being included in the chain), it will lead to significant forking and reduced security [74,76].

The subject of block size is a hotly debated topic in Bitcoin where no common view has been reached. Original Bitcoin design by Satoshi Nakamoto has introduced a limit on the maximum block size; this limit still applies. A bigger block size would allow more transactions to be included in the block, thereby improving the throughput of transactions. On the other hand, a bigger block would potentially increase the block propagation time (see Section 4.3), which may give an advantage to the miner of the current block who can start mining the next block earlier before it is disseminated to other miners. This may potentially increase the centralization in the Bitcoin network. However, the exact effect of the latter is very difficult to capture and analyze without deployment, which contributes to the controversy; some believe that a limit on the block size is artificial and can be relaxed or even eliminated.

An attack strategy called *selfish mining* [77] was proposed to show that even with less than half of the network's mining power, misusing of the Bitcoin's longest chain rule is possible. The principal claim of this work is that a selfish mining pool with 1/3 mining power of the network can still defeat the honest mining protocol, and the revenue of a selfish pool rises superlinearly as the pool size grows. However, some researches such as [78,79] refute selfish mining strategy's assumptions and benefits, and claim that there is no advantage for a selfish miner to follow the attack strategy instead of the honest one. Selfish mining has not been observed in the Bitcoin network in practice which reinforces these claims. Even if a selfish mining attack succeeds in the short term, it may lower the value of Bitcoin, which further reduces the benefits of the attack compared to honest mining.

4.4.2. Updating blockchain in Ethereum

Conceptually, the protocol of updating the blockchain in Ethereum is similar to that of Bitcoin: Both systems are permissionless and use mining. Both systems use the same entities that play the same roles, e.g., the miners propose new blocks of transactions. In both systems the full nodes maintain local copies of the ledger which eventually converge but may temporarily diverge.

At the same time, the two systems exhibit significant differences related to the design goals and choices, and deployment characteristics, as we elaborate below. These differences affect both performance metrics (such as transaction throughput, transaction confirmation time and chain convergence speed, and energy consumption) as well as system security and the risk of centralization.

With respect to design goals, Ethereum supports general transactions written in a Turing-complete programming language. This means that the execution of transactions themselves is more resource- and time-consuming in Ethereum compared to Bitcoin. This leads to reduced throughput and increased energy consumption. Since executing transactions is unlikely to be as computationally intensive as solving cryptopuzzles, the effect is likely to be less pronounced compared to that of PoW. However, it has never been analyzed. Another interesting aspect of Turing-complete transactions is that it becomes absolutely essential to limit a block. However, unlike in Bitcoin, it does not make sense to express the limit in bytes because the block size is no longer the main limiting factor. Instead, the limit is specified in the amount of computation required, which is expressed in the maximum amount of Ethereum gas consumed. Of course, the limit on the amount of computation also indirectly restricts the length of transactions and their number in a block and thereby, the block size. In September 2019, the average block gas limit was around 10,000,000 units [80], which translated to the average block size of between 20 to 30 kilobytes [81], based on the transaction rate and transaction computational requirements in Ethereum at that point in time.

Another difference in design goals is that Ethereum supports more complex state storage (see Section 4.2.1), which results in bigger state storage sizes compared to Bitcoin. While this has an impact on most performance metrics, the exact effect has never been scientifically studied.

Regarding design choices, Ethereum uses a somewhat different conflict resolution rule and a block reward model, as well as less complex cryptopuzzles. Besides, a higher number of subsequent blocks is required for transaction confirmation in Ethereum compared to Bitcoin. We now consider these in detail.

For conflict resolution, Ethereum is using a modified version of Greedy Heaviest Observed Subtree (GHOST) protocol [74] instead of the longest-chain rule. This fact is mentioned in [82] without comparing the original version of the GHOST protocol with the version used in Ethereum. We believe that our analysis below provides the first complete comparison in the literature.

The first point of departure in all versions of the GHOST protocol compared to Bitcoin is that the miner producing a block is incentivized to reference a limited number of stale blocks, in addition to the previous block. In Ethereum, the miner can specifically reference up to two stale blocks from the header of the newly generated block, as shown in Fig. 8. The miner is incentivized to do so by increased block creation reward for each referenced stale block. A stale block that is not included in the main chain but referenced by a main chain's block is called *uncle*

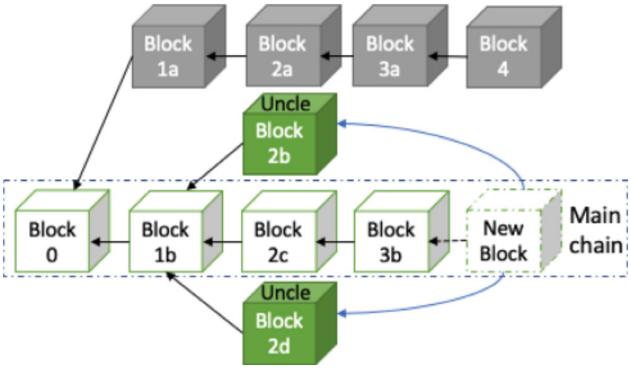

Fig. 8. Ethereum chooses the main chain based on the GHOST protocol. Uncle blocks are considered when choosing the main chain (the heaviest chain rule) and their miners are rewarded in Ethereum.

or *ommer* block in Ethereum. In the example of Fig. 8, Blocks 2b and 2d are uncle blocks for the new block. The uncle block's miner is also compensated, so that the miner's effort of producing the block does not go unrewarded.

Secondly, both the original GHOST protocol and the version used in Ethereum consider the uncle blocks and merge the amount of work that has been done for each block in different branches in order to choose the main chain. For example, in Fig. 8, the main chain is chosen based on the GHOST protocol, and the result is different from the longest chain rule. The algorithm starts from Block 0 and at each fork, chooses the block leading to the heaviest subtree. In the given example, the subtree of Block 1a contains 4 blocks, while the subtree of Block 1b (prior to the inclusion of the new block) consists of 5 blocks. Assuming that each block has the same cryptopuzzle complexity, Block 1b will be chosen. According to the same rule, Blocks 2c and 3b are included in the main chain. Then, a miner that wants to propose a new block, will choose Block 3b as the predecessor.

The original GHOST protocol additionally considers transactions in uncle blocks when computing the blockchain state. This element is not present in Ethereum.

Since the stale blocks are adding to the weight of the main chain, forks are less affecting mining power utilization in Ethereum compared to Bitcoin [83], which makes the main chain more secure and reduces the risk of centralization. As a consequence, Ethereum can afford shorter block creation time, which results in a higher number of forks but also better decentralization, transaction throughput, and confirmation time [74,76]. Specifically, the average time to solve a cryptopuzzle is 12–15 s in Ethereum compared to 10 min in Bitcoin [84].

Regarding the confirmation time, an Ethereum full node confirms inclusion of a transaction after 12 subsequent blocks are included into the local copy of the blockchain [85], in contrast to 6 blocks in Bitcoin. The probability of a transaction inclusion being final has not been analyzed as thoroughly for Ethereum as it has been for Bitcoin. However, as the Ethereum's block creation time is shorter compared to Bitcoin's, Ethereum transactions are confirmed faster.

Table 6 presents a summary of the discussion in this section, with a particular emphasis on how each design difference in Ethereum compared to Bitcoin affects a variety of performance and security metrics. The comparison is based on the conceptual differences rather than empirical measurements. For example, the fact that transactions in Ethereum are Turing-complete programs may make them much more computationally intensive compared to simpler transactions in Bitcoin, resulting in worse throughput, higher energy consumption, and increased confirmation time. It does not mean, however, that actual Ethereum transactions are more computationally intensive in practice because the users may be unwilling to exploit the capabilities provided by Ethereum and pay the higher gas price of such transactions. The "unknown" entries signify research gaps: they indicate that it is difficult to assess the impact without further analysis and that the volume of studies analyzing the effect of the corresponding design element on a particular metric is limited.

Finally, the characteristics of the main Ethereum deployment, namely the Ethereum network, distribution of the mining power across the miners, and the rate and computational requirements of transactions have a non-trivial impact on the performance and security metrics. An empirical study of various metrics in Ethereum is available in [65]. Such a study can only evaluate the actual performance of the system as a whole; it cannot evaluate the effect of each design element separately from the rest of the system. This work shows, e.g., that the mining power utilization in Ethereum is 97% compared to 99% in Bitcoin. Besides, the throughput in Ethereum is 15 transactions per second [86] compared to 7 transactions per second in Bitcoin [87].

4.4.3. Updating blockchain in Hyperledger Fabric

The consensus protocol in Hyperledger Fabric is fundamentally different from those in Bitcoin and Ethereum. First, the entities are different as explained in Section 3.3. Second, Fabric provides the ability to partition a consortium network into independent channels so that each channel can execute a consensus protocol in parallel and independently of other channels. In a sense, there is a separate ledger maintained for each channel. Besides, only authorized peers can participate in the consensus for each channel, as determined by the consortium policies and related channel configuration. Third, the consensus protocol in Fabric is deterministic unlike Bitcoin and Ethereum where there could temporarily be divergent branches of the ledger that eventually and probabilistically converge. In Fabric, there cannot be divergent branches and the transaction becomes final and 100% confirmed relatively fast.

A different class of consensus protocols leads to a remarkable contrast in performance characteristics. As discussed in Sections 4.4.1 and 4.4.2, Bitcoin and Ethereum can achieve throughput of 3–7 and 12–15 transactions per second respectively when updating the ledger. On the other hand, [88] has shown that the throughput in Hyperledger Fabric can reach many thousands of transactions per second with specific optimizations in place, as we discuss in the later part of this section. While providing

Table 6
How differences from Bitcoin affect various performance and security metrics in Ethereum.

Factor	Throughput	Confirmation time	Energy consumption	Decentralization and Mining power utilization
General transactions	Worsens	Worsens	Worsens	Worsens
Complex storage	Worsens	Worsens	Worsens	Unknown
Considering uncles	Unknown	Unknown	Unknown	Improves
Less complex cryptopuzzles	Improves	Improves	Improves	Worsens
Deployment characteristics	Unknown	Unknown	Unknown	Unknown

a high transaction rate, the deterministic consensus protocol in Fabric does not scale well w.r.t. the number of peers in any single channel. At the same time, the probabilistic consensus protocols of Bitcoin and Ethereum can handle the scale of many thousands of full nodes, which is beyond the reach for Fabric [30].

Ledger update in Fabric is performed in three phases called proposal, ordering, and validation [63]. In the proposal phase, clients send every transaction proposal to the endorsers (see Section 3.3), which are responsible for simulating the transaction without updating the database. The endorsement policy specifies which endorsers a transaction is sent to for approval, and how many endorsements are needed for a transaction. For example, a policy can specify that m signatures out of n endorsers are enough for a given transaction to proceed to the ordering phase.

During the ordering phase, the orderer service runs a consensus protocol and decides on the order in which concurrent endorsed transactions will be executed and added to the ledger. Hyperledger Fabric uses pluggable modules for the ordering service which allows for significant configurability. Currently, the Raft [63] implementation based on the Raft protocol [89] is the recommended option due to being easier to set up and manage compared to the alternatives [63]. However, PBFT [90], Apache Kafka [91], BFT-SMaRt [92], SBFT [93], and other implementations can also be used as the orderer service for Hyperledger Fabric. The choice of the implementation and specific consensus protocol may significantly affect the throughput of adding transactions to the ledger.

Additionally, the resilience of the entire ledger update procedure is mainly inherited through the fault-tolerant guarantees offered by the chosen ordering protocol. The two most popular families of protocols for implementing ordering are Paxos and View-stamped replication [94]. Both protocols tolerate crash failures when the number of faulty processes is below $n/2$, where n is the total number of participants. To tolerate byzantine failures, ordering protocols such as BFT-SMaRt [92] or PBFT [90] can be employed [69]. These protocols work when the number of faulty participants is below $n/3$.

Then, in the validation phase, the orderer service sends the block containing ordered transactions to the peers, and the peers validate the correctness of transactions. Transactions successfully validated and committed at this stage are never revoked, which means that their inclusion into the chain is final and deterministic.

It is interesting to observe that in Bitcoin and Ethereum, the block proposer (miner) broadcasts the blocks only after verifying that all transactions in the block are valid. In contrast, the block proposer (orderer) in Fabric does not validate transactions while the blocks are created in phase two. Transaction validation only happens at the proposal (first) phase by the endorsers and at the validation (third) phase of the consensus protocol. While this makes the protocol susceptible to denial-service of attacks by polluting the blocks with invalid transactions, the permissioned access allows Fabric to make an assumption that there is no significant incentive for the entities participating in the consensus protocol to launch a deliberate security attack. The design decision to offload the transaction validation task from the block proposer confers performance benefits in terms of higher throughput at the cost of ledger potentially storing invalid transactions. The study of [95] shows a number of interesting performance characteristics of the Hyperledger Fabric as follows. First, there is a saturation point for throughput at around 140 transactions per second (tps). When the transaction arrival rate reaches the saturation point, the commit latency of blocks is increased from 100 ms to tens of seconds. The raise in the latency is due to the increased number of ordered transactions waiting in the queue for the peers to validate their correctness in the validation phase. In other words, the validation phase becomes the bottleneck. Furthermore, when the transaction arrival rate is lower than the saturation threshold, the increase in block size leads to the increase in the block creation time in the orderer phase which in turn impacts the transaction latency. Additionally, when the arrival rate is 50 tps, the increase in the block size from 10 to 100 causes transaction latency increase by a factor of five from 242 ms to 1250 ms. On the other hand, it has also been demonstrated that when the transaction arrival rate is higher than the saturation threshold, the increase in block size leads to decreased transaction latency and enhanced throughput. This is due to the fact that the time taken to validate and commit one large block is shorter than the time taken to validate and commit many small blocks.

Additionally, it is demonstrated in [95] that heterogeneity in the resources of peers and networks of organizations leads to performance degradation. Furthermore, this work shows that increasing the number of channels, using fewer endorsers, and performing bulk read/write operations contribute to better throughput and reduced latency in Fabric.

Moreover, common system optimization techniques are used in [88] to achieve end-to-end transaction throughput of around 20,000 transactions per second. Examples of these techniques include separation of concerns (separating metadata from data while creating blocks in the ordering service), using parallelism and caching during the transaction validation, and replacing the database of worldstate with lightweight in-memory data structure such as hashtable. One of the key assumptions in [88] towards achieving high throughput is that the incoming transaction workload is contention-free. When there is a high contention and a large number of incoming transactions compete for a small set of hot keys in the worldstate, the throughput drops down significantly as discussed in [96].

The authors of [96] propose changes to the transaction flow in Hyperledger Fabric to handle both high and low contention transaction workloads. They show that the achieved transaction throughput (3000 transactions per second) is significantly better than in Fabric and [88] when the workload exhibits high contention. One of the key design ideas in [96] is to separate the dependent and independent sets of transactions in the block and process both sets concurrently during the validation phase. To realize this idea, a transaction dependency analyzer is introduced, whose goal is to isolate the transactions that are having overlap on the keys of their R–W sets with transactions that do not have any overlaps. As a consequence, the knowledge regarding the subset of transactions that can be processed concurrently is gained. The dependent transactions are marked invalid in the R–W set validation and will be re-executed on the latest version of the worldstate. The independent set of transactions that are marked valid can be successfully committed concurrently.

4.4.4. Updating blockchain in IOTA

While the consensus protocol in IOTA is probabilistic, it is different from the protocols in Bitcoin and Ethereum in many respects. Neither the concept of miner nor the chain of blocks exist in the IOTA network. As it is explained in Section 4.2, IOTA utilizes a specially crafted DAG called Tangle instead of a chain of blocks. In a Tangle, vertices represent bundles (or transactions) while the edges signify approval of the bundles (see Fig. 9). A Tangle starts with a genesis bundle and grows as new bundles are added. Each new bundle is supposed to validate and approve two bundles already included in the Tangle, though there are exceptional cases when only one bundle is validated and approved, instead of two. Similar to the ledger in Bitcoin and Ethereum, each node participating in the IOTA network maintains its own copy of the Tangle and synchronizes it with other

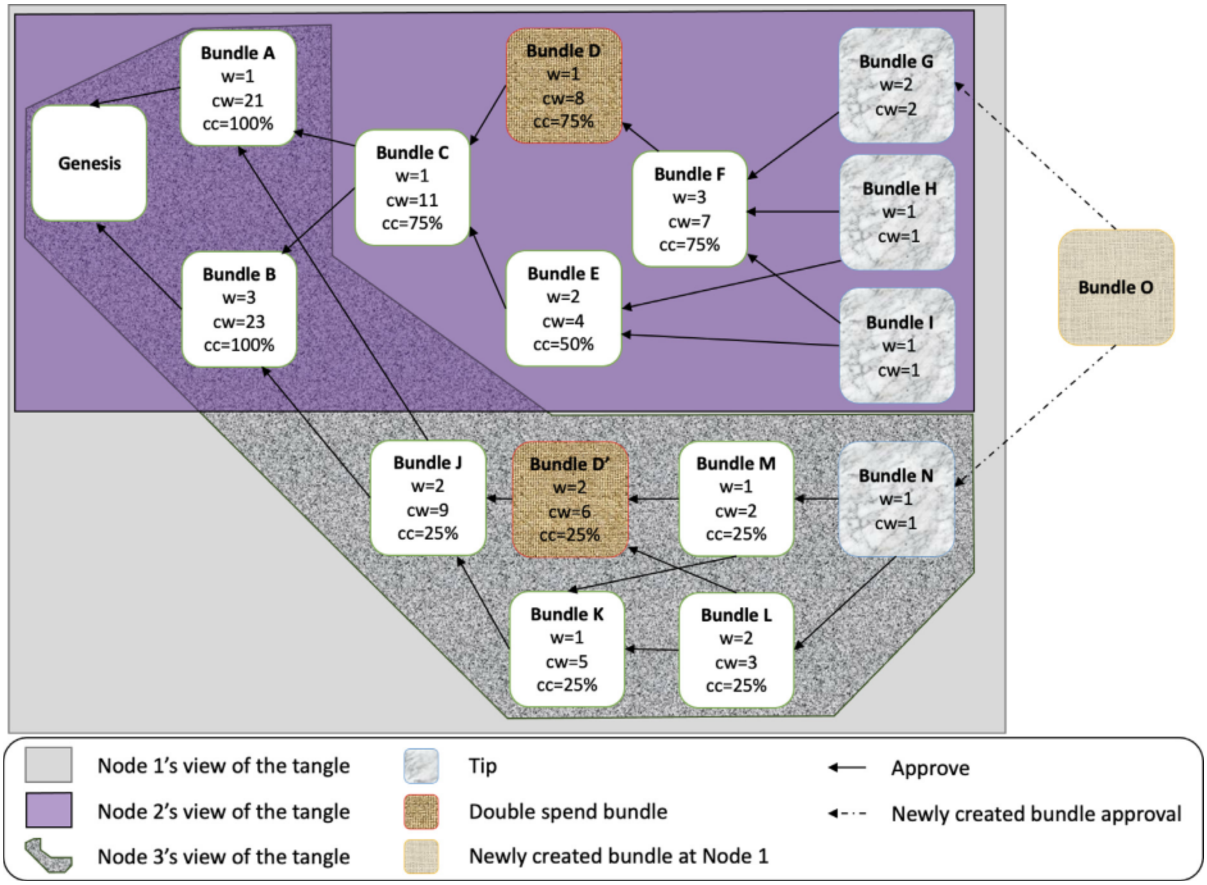

Fig. 9. Every node has its own copy of the Tangle; copies might be diverging.

nodes; these copies may be temporarily diverging. Fig. 9 illustrates a situation where participating nodes 2 and 3 have a partial knowledge of the Tangle while node 1 possesses combined knowledge.

In IOTA, every node x that wants to issue a transfer of funds has to perform the following sequence of actions:

Bundle creation: x creates a bundle including all input and output transactions that together constitute the transfer (see Section 4.2.4). For example, node 1 creates bundle O in Fig. 9.

Weight assignment: x decides on an integer weight w for the bundle. A bigger weight results in a probabilistically faster confirmation for the bundle (as we explain below) at the expense of a more significant computational effort for x.

Cryptopuzzle solution: x needs to solve a cryptopuzzle for each of the bundle's transactions. The difficulty of the cryptopuzzle is proportional to w selected by x. It is however, significantly lower compared to Bitcoin and Ethereum because in the latter, the cryptopuzzles regulate proposal creation rate while in IOTA they only prevent DoS attacks.

Tip selection: x selects two *tips* to approve where the tip is a bundle already included in the Tangle but not yet approved by any other bundle. For examples, bundles G, H, I, and N are tips in Fig. 9. To this end, x runs a tip selection algorithm [31] twice, once for each tip. We describe the tip selection algorithm in detail below. In addition to the tip itself, the algorithm selects a path from the genesis bundle to the tip.

In exceptional cases, only one tip exists in the local copy of the DAG maintained by x when x creates a new bundle. For example, when node 3 created bundle K, only a single tip J was available in the DAG. In such cases, only a single tip is selected and approved by the new bundle. However, such cases are fairly uncommon in practice because the deployment involves many nodes and the rate of producing concurrent bundles is high.

Validation: First, x validates the integrity of every bundle on the two selected paths. If any structurally invalid transaction or bundle is found, it is excluded from the Tangle. Secondly, x validates that the two paths contain no conflicting bundles that lead to double-spending. To detect possible double-spending or prevent spending a larger sum than what is available in an account, x needs to keep values of all addresses in all the verified bundles on the two selected paths. If any of the values turns negative, then we know that the paths include conflicting transactions but we do not necessarily learn which transactions are conflicting.

If the validation fails because of any of the above two issues, the tip selection algorithm is invoked again to find a different pair of tips and paths. Then, the validation is performed for the new pair of paths. This happens repeatedly until a valid pair is found. Some additional optimizations are implemented in case of structurally invalid bundles, which we do not cover in our description.

In Fig. 9, no bundle violates structural integrity but bundles D and D' are conflicting. In this case, node 1 selects a new pair of tips as shown in Fig. 10. The new pair of paths passes the validation so that the algorithm stops.

Bundle propagation: x propagates the new bundle along with the id of the two selected tips using the communication layer. Interestingly, a proof of neither correct tip selection nor performed validation is supplied in the message. When a node other than

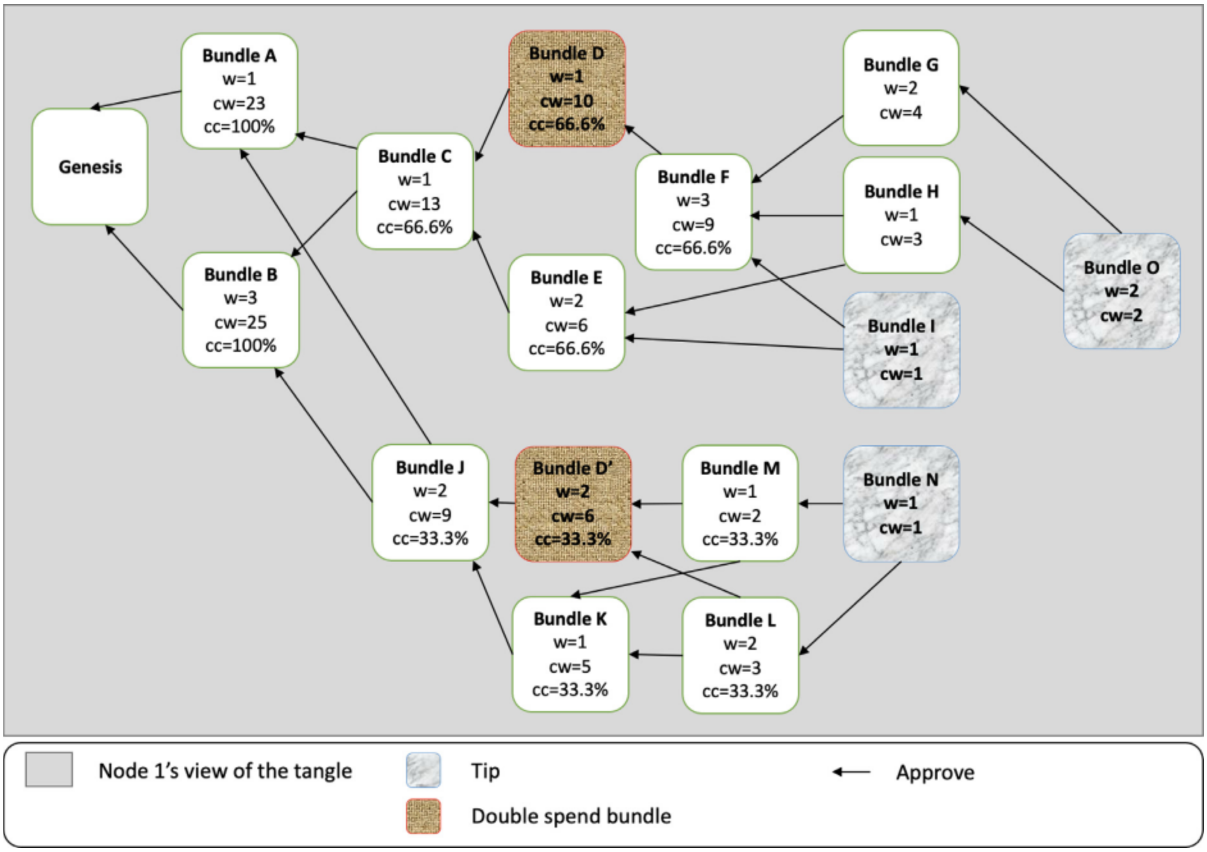
Genesis
Bundle A w=1 cw=23 cc=100%
Bundle B w=3 cw=25 cc=100%
Bundle C w=1 cw=13 cc=66.6%
Bundle D w=1 cw=10 cc=66.6%
Bundle E w=2 cw=6 cc=66.6%
Bundle F w=3 cw=9 cc=66.6%
Bundle G w=2 cw=4
Bundle H w=1 cw=3
Bundle I w=1 cw=1
Bundle O w=2 cw=2
Bundle J w=2 cw=9 cc=33.3%
Bundle D' w=2 cw=6 cc=33.3%
Bundle M w=1 cw=2 cc=33.3%
Bundle N w=1 cw=1
Bundle K w=1 cw=5 cc=33.3%
Bundle L w=2 cw=3 cc=33.3%
Node 1's view of the tangle
Tip
Approve
Double spend bundle

Fig. 10. Choosing a pair of tips happens repeatedly until a valid pair is found.

x receives the bundle from the communication layer, it performs the cheaper integrity validation for that bundle. However, it does not perform an expensive detection of double-spending nor does it traverse any path from the genesis block to the new bundle.

Monitoring of confirmation and reattachment: The validation and tip selection algorithms include a lot of randomness as we discuss below. Because of this randomness, some bundles may get confirmed much faster than the others, while some valid bundles may in fact be never confirmed and be *left behind*. In view of this, x needs to monitor the Tangle for signs of confirmation, which we also discuss below. IOTA introduces the concept of reattachment: node 1 may create a new bundle in the future that specifies the very same transfer as bundle O in the illustration, and inject this new bundle into the Tangle by performing all the above steps. In this case, the new bundle will be conflicting with O so that node 1 should only do it if O is not getting confirmed for a long period of time.

The tip selection algorithm is essentially a weighted random walk through the Tangle starting from the genesis bundle and moving towards the tips. At each step, the algorithm proceeds from the current bundle to one of the bundles directly approving the current bundle. The probability of choosing a bundle as next step is proportional to the cumulative weight of that bundle. The cumulative weight of each bundle Y is equal to the own weight of Y plus the sum of own weights of all bundles that directly or indirectly approve Y. Fig. 9 illustrates the cumulative weight of each bundle (denoted as *cw* inside the bundles) as calculated by node 1 (the values for nodes 2 and 3 will be different). For instance, the cumulative weight of A is equal to the sum of weights of bundles A, C, D, E, F, G, H, I, J, D', K, L, M, and N. When the random walks determines the next step from the genesis bundle, it will make a probabilistic choice between bundles A and B based on their cumulative weights (of 21 and 23 respectively). The random walk stops when it reaches one of the tips.

Based on the above algorithm, it is clear that choosing a bigger weight for a new bundle increases the probability of that bundle to be selected by the random walk performed during the tip selection. This ultimately results in probabilistically faster confirmation for a bundle with a large weight.

Besides, it is obvious that the cumulative weight of a bundle incrementally increases as it is getting approved directly or indirectly by new bundles added to the Tangle. As a result, the cumulative weight can be used as a metric for confirming bundle inclusion in the Tangle, similarly to how the number of subsequently added blocks is used as a metric for confirming block inclusion in Bitcoin and Ethereum. This is important, e.g., when a node decides to reattach the bundle. Notably, heavy bundles tend to gain the weight faster due to the weighted random walk. In some scenarios, this may lead to situations when a bundle is never confirmed, thereby rendering the reattachment mechanism essential. However, as the number of users and transactional rate in the network increase, the probability and speed of confirming any bundle regardless of its weigh grow.

Apart from the cumulative weight, there exists another metric in IOTA used for monitoring bundle inclusion. The metric is called *confirmation confidence* and denoted *cc* in Fig. 9. The confirmation confidence of a bundle is the percentage of the tips which directly or indirectly approve the bundle. In Fig. 9, if we consider the Tangle status on node 1 before O is included, the confirmation confidence of bundle D is %75. This is because among the four tips of G, H, I, and N, the three tips G, H, and I are approving D while N is not. Note, however, that confirmation confidence is not always a good indicator of bundle inclusion if the Tangle copies on different nodes are diverging. For example, the confirmation confidence for bundle K is %100 on node 3 but this is only because node 3 does not know about bundles G, H, and I yet.

In terms of Tangle security, several attacks and defenses have been considered in the literature [21,31]. The *replay attack* [97,98] performs double-spending by exploiting the reattachment mechanism: the reattached bundle normally invalidates the original bundle but in the attack, both bundles remain valid, which results in double-spending. The attack only works under reuse of addresses, which is within the control of the users and which the users should strive to avoid as per IOTA recommendation. In the *parasite chain attack* [31,99], the attacker aims to replace the current Tangle with its own subgraph, which the attacker builds in secret while taking steps to ensure that the secret subgraph will have sufficiently high cumulative weights. To perform double-spending, the attacker initiates two conflicting transactions. The first transaction is injected into the current Tangle while the second transaction is included into the secret subgraph. After the first transaction is confirmed and the attackers receives the services or benefits for the payment, the attacker broadcasts all bundles in the secret subgraph, which replaces the Tangle. This way the attacker gets the second transaction confirmed as well. In the *splitting attack* [31,100], the attackers partitions the Tangle by making sure that two subgraphs within the Tangle maintain a similar cumulative weight over time, and injecting transactions so as to maintain this balance. Such partitioning allows the attacker to perform double-spending by placing two conflicting transactions, one in each subgraph.

The author of [31] proposes the following idea to defend against the parasite chain and splitting attacks. The probabilistic choice of the next step in the random walk is governed by parameter α in addition to the cumulative weights of candidate vertices. A big value of α means that the vertex with a larger cumulative weight wins with a very high probability while a small value of α means that the winning chance for a heavier vertex is only slightly higher. This work shows that a large value of α reduces the risk of successful parasite chain and splitting attacks. Unfortunately, larger values of α increase the probability for a bundle to be left behind and never confirmed [99,101]. In [99], the authors propose a detection mechanism for parasite chains, with the idea of using smaller values of α in a normal situation but adaptively increasing the values if a parasite chain is detected. A different idea advocated in [101] is to modify the tip selection algorithm so as to explicitly search for left-behind tips and increase the chance of their selection.

Since the tip selection and validation are nontrivial algorithms in IOTA, it is interesting to consider incentives to perform them for node x that wants to issue a new bundle. This is especially important because during the bundle propagation phase, x does not supply any proof of correct tip selection and validation. However, while a node receiving a bundle from x does not perform a random walk with validation immediately, it will do so upon creating its own bundle in the future. If x does not perform the validation properly and approves two tips from two traversal paths that are conflicting, this may also negatively affect the probability of confirmation for the new bundle produced by x. In the illustration of Fig. 9, if O approves G and N, then the conflict between D and D' may be discovered by a later random walk, which will reduce the probability of O to be selected. However, there are multiple paths from the genesis bundle to O that do not traverse D and D', in which case O may still be selected. Note that if a conflict between D and D' is later detected by other nodes, it is impossible for them to establish whether x deliberately neglected validation or simply was unlucky with detection because x selected non-conflicting paths leading to G and N during its random walk. The exact quantification of the reduction in confirmation probability requires a very complex probabilistic analysis, which has not been performed in the literature to the best of our understanding.

In the context of tip selection, different tips have different probability to be reached by the random walk. If x just arbitrarily selects two tips without performing the random walk, this may result in reduced probability for the new bundle to be reached by the random walk performed by other nodes at a later point. As a result, x will be punished by a higher confirmation time. On the other hand, x may minimize the confirmation time for its bundle by performing a deterministic walk which selects a vertex with the biggest cumulative weight at each step. Eliminating the randomness during the walk may exacerbate the problem of heavy vertices gaining weight very fast, resulting in a higher number of reattachments. There exist considerations, however, that detract from the benefits of such a selfish node strategy, as outlined in [31].

The consensus algorithm in IOTA is relatively lightweight and scalable in the sense that it can sustain a higher rate of adding new transactions to the storage and confirming them. This improvement eliminates the need in two design elements present in the other three systems we consider: batching transactions into a block and having long intervals between consecutive blockchain updates. As a result of allowing shorter intervals between updates, PoW is much more lightweight in IOTA compared to Bitcoin and even Ethereum. While the typical IOTA throughput has been reported as 45–50 transactions per second [102], IOTA has mentioned a throughput of above 500 transactions per second achieved under a proprietary stress test [103]. These advantages do come at a cost. Since discovery of conflicts is probabilistic, it may take a lot of time until double-spending is discovered. The Tangle storage requires more space compared to systems that use a chain of blocks, as explained in Section 4.2.2. Most importantly, the paths produced by the tip selection algorithm become long as the Tangle grows, resulting in a very expensive verification procedure.

To address the challenge of scaling the Tangle and the validation algorithm, IOTA employs a snapshot process that prunes the transaction history (see Section 4.2.8) and shortens validation paths by creating a new snapshot bundle that effectively serves as a new genesis bundle. The snapshot process is periodically performed by the IOTA Foundation, which goes against the blockchain spirit of eliminating the trust in any single entity or organization. The Coordicide project [104] is focused on removal of coordination and in particular, the need for a snapshot process.

4.4.5. Defense against DoS attacks

DoS attacks in Bitcoin, Ethereum and IOTA are mitigated by the Proof-of-Work mechanism. As discussed in Section 3.1, a PoW algorithm requires solving cryptographic puzzles for creating new blocks. If a message with a new block or transaction does not contain a valid solution to a cryptopuzzle, the message is rejected without further processing, which makes it easier for the system to filter out spurious messages.

However, PoW requires a significant amount of energy [105] and wasted computation performed by legitimate miners. Proof-of-Stake (PoS) protocols were developed as energy-saving alternatives to PoW [106]. Instead of solving a cryptopuzzle as in the PoW, a block creator is selected based on its stakes (the number of digital tokens that it holds) in PoS. As there is no heavy computation process in PoS, the block creation time is much shorter than in PoW, thereby resulting in a higher transaction throughput. In order to prevent DoS, a block proposer needs to make a deposit to gain the right to make a proposal. Since the deposit is locked until the next block is selected, this makes it more expensive for the miner to create multiple proposals within a short period of time. Besides, the deposit can be confiscated for malicious behaviors such as performing a DoS attack.

The Ethereum network has developed the Casper protocol [107] in an attempt to ease the transition from the current PoW protocol to a pure PoS protocol. Ethereum 2.0 will deploy the Casper protocol on top of the existing PoW protocol, prior to switching to a pure PoS protocol in future releases. So, while blocks are still going to be mined via PoW in Ethereum 2.0, every fixed interval (every 50 blocks [108]) is going to be a PoS checkpoint where the finality of blocks is assessed by a dynamic committee that votes via a BFT protocol. To join the committee, a validator has to make a deposit to gain a voting right proportional to that deposit.

An additional vector of attacks for a user in Ethereum is to inject a computationally- or storage-expensive transaction that would consume resources on all of the full nodes in the network. The gas budget concept (see Section 3.2) plays an essential role in defending against this attack: the attacker itself would need to pay a substantial amount of gas in order to launch the attack.

While IOTA is using PoW as well, the cryptopuzzles in IOTA are much simpler compared to Bitcoin and Ethereum which leads to a higher throughput. The reason for choosing a lower-difficulty PoW in IOTA is that IOTA uses PoW just as a spam protection [104]; whereas, in Bitcoin and Ethereum, PoW is additionally used for controlling the rate of block production. On the other hand, a separate PoW is required for each transaction in a bundle since a bundle may theoretically contain an unlimited number of transactions. While the current implementation of IOTA is utilizing a simple PoW mechanism, the IOTA Coordicide [104] project has proposed an adaptive rate control mechanism which intelligently adjusts the difficulty of the PoW per IOTA node based on different factors, such as a number of recently-issued transactions and reputation of the issuer [104].

Hyperledger Fabric does not have a single main deployment. Instead, there is a multitude of proprietary deployments, where each deployment is partitioned by channels (see Section 4.3). Besides, the system is permissioned and requires authorization for joining each channel. These factors significantly mitigate the risk of DoS attacks.

4.4.6. *Querying blockchain*

In Section 4.2, we describe different data items stored by the data recorders of each blockchain system. Any data stored by the recorders can in principle be locally queried by them. If a data recorder acts as a query responder, it can essentially answer any queries related to the stored data that are initiated by query issuers.

However, the fact that the data is stored does not necessarily mean that there exist means to query it efficiently. The functionality of blockchain systems is different compared to general-purpose databases (see Section 2.1). The native implementation of most blockchain systems only efficiently supports limited queries as simple as retrieving a data item by its hash code. The main reason for the limited support of queries is the limitations in the default database implementation of the blockchain systems. For example, levelDB used in Bitcoin, Ethereum, and Hyperledger Fabric as a key–value database supports only querying based on keys. Another reason is the way of utilizing the database for storing and accessing data. For example, retrieving values in Ethereum requires traversing multiple trie structures (see Fig. 4 in Section 4.2.5), and accessing values associated with the queried hash code (i.e. the key) requires searching multiple keys over the disk, which is time-consuming [109].

Due to these reasons, supporting analytical and complex queries (querying based on values other than keys) such as filter, aggregation, and sorting queries on the blockchain systems require some extra considerations and functional components. The modularity of Hyperledger Fabric and its focus on highly configurable proprietary deployments allow for modifying the native implementation. Specifically, replacing the default database module of LevelDB with CouchDB allows for supporting more complex search semantics in an efficient way [50]. For example, as CouchDB allows storing data in JSON format, it is possible to query by data values, and not just by keys. The tradeoff, however, is that CouchDB requires more disk space for storing the data compared to LevelDB.

Research proposals of adding an extra query layer to existing blockchain systems have also emerged. For example, *EtherQL* [109] supports a set of analytical queries such as aggregation and top-k queries on top of Ethereum. These proposals, however, have not been implemented in the actual systems as of 2021.

The most common way of solving the issue of rich query semantics is designating a subset of data recorders as dedicated query responders (see Section 2.3). In practice, query responders maintain a separate database in addition to the blockchain data, which allows them to answer richer queries efficiently. Such designated query responders are used in Bitcoin, Ethereum, and IOTA. *Blockchain.com* [110] and *etherscan.io* [111] are websites that allow anyone to search for a specific address, transaction, or block on Bitcoin and Ethereum databases respectively. However, these sites do not provide more advanced search functionalities such as looking for transactions with a specific amount of input. Some other services store the blockchain in an SQL database and provide the ability to issue general SQL queries. Notably, *BlockchainSQL* [112] does it for Bitcoin while *Anyblock* [113] provides such a service for Ethereum and some other blockchain systems. IOTA foundation provides a website (*explorer.iota.org* [114]) as an external dedicated service for querying IOTA. This website supports basic search functionalities for the IOTA Tangle based on keys, such as searching based on a transaction hash, a bundle hash, or an IOTA address. As an example, searching for an IOTA address through this website gives information about the balance of the address and all transactions related to that address. *thetangle.org* [115] is another external dedicated service for IOTA, which provides visualization for a number of specific queries.

However, such designated query responders pose two major challenges. First, the exact rich query semantics is up to a specific implementation; it is not standardized in any way. Secondly and most importantly, they do not provide the security, dependability, and availability guarantees commonly attributed to blockchain systems. This is because they are managed by individual computing devices that are fundamentally untrusted in the blockchain paradigm. In particular, query responders can decide on an access control policy that is not aligned with the access control policy of a particular permissioned or permissionless blockchain system. For example, *Anyblock.tools* provides different query services based on the paid subscription tier of the users.

In summary, query responders serve two purposes: (a) to provide rich query semantics otherwise unavailable in the blockchain system and (b) to provide information to computing devices that do not belong to the blockchain network.

4.5. *Contract layer*

The contract layer allows users of the blockchain system to develop programmatic extensions and install them atop blockchain. Smart contracts are similar to distributed objects in nature. A contract implements a collection of procedures, each of which can be

Table 7
Contract layer of different blockchain systems.

Features	Bitcoin	Ethereum	Hyperledger Fabric	IOTA
Type	Scripting system	Smart contracts	Smart contracts packaged into a chaincode	No support yet (in progress)
Programming language	Forth-like	Several languages, Solidity as the most popular one	Go, Node.js, Java	–
Turing completeness	No	Yes	Yes	–
Executing computing devices	Full nodes	Full nodes	Specified by endorsement policy	–
Execution language	Bitcoin script	Ethereum Bytecode	Programming language-dependent	–
Execution environment	Script interpreter	Ethereum Virtual Machine	Docker container	–

invoked remotely and executed as a transaction. A contract may additionally invoke transactions implemented by other contracts.

Features of the different blockchain systems in the contract layer are shown in Table 7. As Bitcoin was designed with a specific defined application in mind, it does not implement a general-purpose contract layer. Instead, it uses a simple scripting system for its transactions. Each transaction contains an output and an input script [22]. When a transaction executes, its inputs are verified. Let us assume transaction T_1 has an input I, which is an output of transaction T_0. In this case, I is verified by executing the output script of T_0 followed by an input script of T_1. By combining the input script of the new transaction with the output script of the referenced transaction, a full node can validate that the new transaction can redeem the previous transaction output. The reason for concatenating the input and output scripts is that the output script specifies the verification procedure, whereas the input script contains cryptographically secure information about the redeeming node (specifically, the public key and a signature of that public key), which is used as an input to the verification procedure. Bitcoin provides a default implementation of an output script (called scriptPubKey) and a default implementation of an input script (called scriptSig). This default implementation tests if the hash of the provided public key matches the hash in the scriptPubKey, and then checks the signature against the public key to finally validate the new transaction. Bitcoin full nodes are responsible for validating the transactions by executing Bitcoin scripts in their script interpreter environment. The programming language of Bitcoin scripts resembles Forth [116]. Similar to Forth, Bitcoin scripts are stack-based and they use reverse-Polish notation (processed from left to right). Since Bitcoin scripts are not intended for writing general-purpose applications, they are not Turing-complete [22]. In particular, the language does not support jumps and loops so that malicious users cannot create code that would waste the power and computing resources of the Bitcoin network.

Instead of simple scripts, Ethereum utilizes smart contracts, which can be used for implementing a variety of decentralized applications. One of the main differences between the Ethereum smart contracts and Bitcoin scripts is that smart contracts are Turing complete, which is essential for a general-purpose blockchain platform. When a contract procedure is invoked, it is executed on top of the blockchain platform by Ethereum full nodes, using the Ethereum Virtual Machine (EVM) environment. As discussed in the storage layer Section 4.2, a smart contract stores its own data. Remote invocations that update the contract state are executed under transactional semantics.

Currently, the most common and supported language for writing smart contracts is Solidity [117,118]. What makes this language popular is that Solidity is object-oriented and high-level, and it provides a rich support for developing contracts. Solidity has similarities to C++, Python, and JavaScript. LLL (a low-level Lisp-like language) [119] and Vyper (Python-derived) [120] are other popular alternative languages for writing smart contracts. A code written in any of these languages needs to be complied to Ethereum Bytecode in order to run in the EVM environment.

Hyperledger Fabric also has the concept of smart contracts, similar to that of Ethereum. However, related smart contracts can be grouped together and packaged into a chaincode in Fabric; While smart contracts are governing transactions, chaincode governs how smart contracts are packaged for deployment [121]. When a chaincode is instantiated on a channel, an endorsement policy is defined for it [121]. Therefore, the endorsement policy applies to all smart contracts defined within the same chaincode in the context of a channel. In Fabric, unlike Bitcoin and Ethereum, transactions and smart contracts are only executed by endorsers (see Sections 4.4.3 and 3.3), which create W-R sets. The other peers validating or applying the transaction only process W-R sets instead of executing the code of the transaction. Currently, smart contracts of Fabric can be written in Go, Node.js, and Java [122], which are all Turing complete languages. Endorsers execute the chaincode and its smart contracts in a Docker container environment corresponding to the contract programming language [122]. Docker container is an isolated sandbox inside the endorser whose purpose is to prevent the chaincode from accessing the endorser's local environment and resources.

The IOTA foundation has stated that smart contracts will not be a feature of IOTA core [123]. However, there are some projects such as Qubic protocol [124] which aim to design a contract layer on top of IOTA Tangle in order to deploy smart contracts and other applications on it.

5. Related work

All published surveys on blockchain technology can be categorized as surveys on individual aspects or on blockchain as a whole. For the former survey type, [69,125,126], and [127] concentrate specifically on the consensus mechanisms of blockchain. In [128–130], and [131], the focus is on the security analysis. Smart contracts are covered in depth in [132,133]. Data management and analysis of blockchain data are investigated in [134, 135]. Network layer aspects of permissionless blockchains are addressed in [136]. Surveys of [137–140], and [141] review integration of blockchain with other technology, namely with edge computing, machine learning, cloud storage, cloud computing, and cloud of things respectively. The works of [142–144], and [145] survey on applying blockchain to various application domains such as IoT, healthcare, and smart cities.

In contrast, blockchain surveys of the second type provide a general exposition of blockchain systems. However, the coverage significantly varies across specific surveys. As our paper falls in this category of blockchain surveys, we explore the related surveys of this type in greater detail and compare them with our work in the rest of this section (see Table 8).

The study of [19] has been one of the first surveys on blockchains as a whole. However, it is limited to the cryptocurrency use

Table 8
Comparison with state of the art surveys on blockchains as a whole.

Main analyzed aspects of blockchains		Our work	[19]	[20]	[146]	[147]	[148]	[149]	[150]	[151]	[21]	[152]	[153]	[17]	[18]
Comparison of alternative definitions		✓[1]	✗[2]	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
Roles and entities		U[3]	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	S[4]	U
Hardware layer		B[5], E[6], H[7], I[8]	B, etc.	B, etc.	✗	✗	B, E, etc.	✗	✗	✗	✗	✗	✗	✗	B[9], E[10]
Data storage layer	State tracking	B, E, H, I	B	B	✗	✗	B, E, etc.	✗	✗	✗	I[11], etc.	✗	✗	B, E, etc.	B, E, H, etc.
	Disk	B, E, H, I	✗	B	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	H[12]
	Memory	B, E, H, I	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
Communication layer		B, E, H, I	B	B	✗	✗	B, E, etc.	✗	✗	✗	✗	✗	✗	B, E, etc.	B, E, H, etc.
Data manipulation layer	Consensus algorithm	B, E, H, I	B	B, etc.	B, E, etc.	B, E, etc.	B, E, etc.	✗	B, E, etc.	E, H, etc.	I, etc.	✗	B, E, H, I, etc.	B, E, etc.	B, E, H, etc.
	Quantitative performance comparison	B, E, H, I	✗	✗	✗	✗	✗	✗	✗	✗	✗	E, H, etc.	✗	✗	✗
	DoS prevention	B, E, H, I	B, etc.	B, etc.	✗	✗	✗	✗	✗	✗	✗	✗	E, H	✗	B, E
	Querying	B, E, H, I	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
Contract layer		B, E, H, I	B	B, etc.	B, E	E, H, etc.	B, E	✗	B, E	E, H, etc.	✗	B, E, H, I, etc.	B, E, etc.	✗	B, E, H, etc.

[1] ✓: Considered,[2] ✗: Not considered,[3] U: Universal,[4] S: System specific,[5] B: Bitcoin,[6] E: Ethereum,[7] H: Fabric,[8] I: IOTA,[9] B: Briefly mentioned in Bitcoin,[10] E: Briefly mentioned in Ethereum,[11] I: Briefly mentioned in IOTA,[12] H: Briefly mentioned in Fabric.

case of blockchain. This work reviews research perspectives and challenges for Bitcoin and other cryptocurrencies. It focuses on Bitcoin and presents transactions (including scripts), the consensus protocol, and the communication network as the main technical components. Incentivizing correct behavior in Bitcoin, transition towards more powerful and energy-efficient customized hardware, and security issues of cryptocurrencies are the other important aspects covered in [19]. Another technical survey on Bitcoin and cryptocurrencies [20] is similar to [19] in terms of the goals. Additionally, it includes a tutorial-style introductory part and an in-depth overview of more recent existing literature.

The authors of [146] compare Bitcoin with Ethereum and propose a design taxonomy based on this comparison. Discussion of the architectural design in [146] is structured by topics of decentralization, computation, infrastructural configuration, among others. Based on this discussion, the work evaluates consensus protocols and system performance. Bitcoin scripts and Ethereum smart contracts are also compared as another principal aspect of blockchain in this paper. Similar to [146,147] is another survey using Ethereum and Bitcoin as a case study to describe the inner workings of blockchain in detail. The survey covers blockchain incentive structures, smart contracts, and scalability issues such as transaction throughput and latency. The study of [148] provides yet another comparison of Bitcoin with Ethereum which considers a number of additional aspects: UTXO and account model for tracking the states, different hardware used for mining in Bitcoin and Ethereum, and the differences in the network layer. Similarly to [146,147], consensus protocol and scripting language of Bitcoin and Ethereum are also reviewed in [148]. As another work that compares Bitcoin and Ethereum, [154] introduces a quantitative framework to analyze the security and performance implications of various consensus and network parameters of PoW blockchains. Impact analysis of parameter choices such as block interval, stale block rate, and average block size on the network propagation time and the throughput of PoW blockchain systems is one of the main contributions in [154].

Lack of formalization and standardization in blockchain technology has prompted a research on ontologies that aim at providing a vocabulary of key blockchain terms. A high-level ontology for concepts and definitions without technical details is proposed in [149]. A more technical ontology is introduced in [150]. Furthermore, this work provides an overview of a number of blockchain components with the main focus on Bitcoin and Ethereum, namely of consensus, transaction model, scalability limits, scripting language, and the reward incentives. These ontologies, however, do not survey alternative term definitions in this turbulent area and do not aim to analyze the differences.

In contrast to the other related works that concentrate on Bitcoin as a basis for analyzing blockchain design, [151] focuses on aspects unrelated to cryptocurrency. This paper compares Ethereum, Hyperledger Fabric, and Corda in terms of participation of peers, consensus, and smart contracts to show the most suitable use cases for each blockchain system. The only other related survey that is not based on Bitcoin is [21] which provides a comprehensive analysis of DAG-based blockchain systems. In this recent survey, consensus over DAGs, performance analysis, and transaction models are considered based on over 20 DAG-based systems including IOTA.

The work of [152] proposes a benchmark framework for evaluating the performance of a variety of blockchain networks, both public and private. The authors apply the framework to Ethereum, Hyperledger Fabric, and another private blockchain system and provide both qualitative and quantitative performance analysis. Additionally, the authors focus on smart contracts and provide a short survey of other blockchain-related mechanisms without specifically focusing on individual systems.

The comprehensive survey of [153] identifies a list of 40 DLT characteristics that are fundamental for assessing the suitability of DLT designs for applications on DLT. These characteristics are grouped into 6 categories. Then, based on the introduced characteristics, [153] proposes 24 trade-offs in the DLT design. Performance is one of the categories analyzed in this survey. As a consequence, performance characteristics such as block creation interval, block size limit, throughput, etc. are explored for Bitcoin, Ethereum, Hyperledger Fabric, IOTA, and other blockchains. According to the classification of [153], smart contracts are one of the characteristics mentioned for the flexibility category; thus, they are discussed in the context of a trade-off with the other DLT characteristics in the survey. In addition to the characteristics explored in [153], consensus mechanisms of Bitcoin, Ethereum, Hyperledger Fabric, and IOTA are briefly introduced, and DoS prevention solutions of Ethereum and Hyperledger Fabric are explained.

Data models, network discovery process, and consensus process across four blockchain platforms including Bitcoin and

Ethereum are compared in [17]. This work also briefly considers actors and roles in each of the systems. The excellent recent tutorial of [18] touches upon a large number of aspects and state-of-the-art mechanisms related to blockchain, including roles, entities, data model, communication protocols, consensus mechanisms, smart contracts, and much more. The tutorial mentions the implementation of as many as seven systems, including Bitcoin, Ethereum, and Hyperledger. The main goal of [18] is to provide a very broad picture of the state-of-the-art and answer a number of important high-level questions, while we provide in-depth design comparison of the four systems.

In Table 8, we explain how our survey differs from the aforementioned studies. The rows of the table correspond to seven major aspects of blockchain: (1) comparison of alternative definitions, (2) taxonomy of roles and entities, (3) coverage of the hardware layer, (4) coverage of the data storage layer including state tracking, on-disk, and in-memory storage, (5) coverage of the communication layer, (6) coverage of the data manipulation layer at fine granularity including consensus algorithm, quantitative performance comparison, DoS prevention, and blockchain querying, and (7) coverage of the contract layer. The coverage of aspects 3 to 7 is shown in the table at the granularity of individual systems we consider in our survey.

The first aspect, i.e, the comparison of alternative blockchain definitions, has not been considered in the state-of-the-art surveys and is one of the contributions of our work. In Section 2.1 we identify five blockchain definitions used in the literature and existing blockchain-based systems. The second aspect is a comprehensive taxonomy of roles and entities. While there are works that consider roles and entities in specific systems, the only prior cross-system taxonomy is given in [18] to the best of our knowledge. We extend this taxonomy to additional roles and present the implementation of each system in the light of this taxonomy. Our analysis of roles and entities, given in Section 2.3, is universal and applicable to all blockchain systems.

The third aspect for comparison is coverage of the hardware layer. While a few related works discuss the mining devices of Bitcoin or Ethereum, we compare all four systems in terms of the limiting resources, cryptopuzzle solving device, etc. We also consider how additional hardware is used for security.

Fourth, we provide a comprehensive coverage of the data storage layer in Section 4.2 including state tracking, on-disk and in-memory storage. While other surveys have considered the storage in Bitcoin, in-depth understanding of the storage in Ethereum, Hyperledger Fabric, and IOTA requires reading system documentation, blog posts, and even the source code. To the best of our knowledge, we provide the first in-depth survey coverage for these systems.

The fifth aspect is coverage of the communication layer. While the basic communication protocol in Bitcoin is explained in a few related works, our description in Section 4.3 considers ordering guarantees, privacy and security, propagation time, initial peer discovery, and geographical proximity between network participants in Bitcoin, Ethereum, Hyperledger Fabric, and IOTA.

The sixth aspect of consideration is coverage of the data manipulation layer. This is possibly the most substantial layer of blockchain; it is touched upon by almost all the related works but the breadth and depth of coverage varies. Consideration of the consensus algorithm has received a lot of attention in each of the systems, yet cross-system comparisons are rare beyond the general comparison between permissioned and permissionless systems. For example, our survey is the first to provide a comprehensive comparison between consensus in Bitcoin and Ethereum and to contrast the consensus in IOTA with other systems not based on DAG. In particular, we provide the most complete presentation of the consensus protocol in IOTA that covers incentive-based attacks. Some of the elements pertaining to the data manipulation layer, such as DoS prevention and querying blockchain are covered in much greater detail in our survey compared to related work. When it comes to performance comparison, we contrast quantitative and qualitative findings about each individual system produced by the developers and researchers and attempt to place them in a unifying framework.

Regarding the seventh aspect, i.e., coverage of the contract layer, state-of-the-art mostly considers the scripting language of Bitcoin and smart contract languages of Ethereum. In comparison, we compare different systems by the contract executing computing devices, programming vs. execution language, and contract execution environment.

6. Conclusions

We have presented a comparative study of Bitcoin, Ethereum, Hyperledger Fabric, and IOTA. The study is organized by roles of the participants, system entities, as well as system layers in a cross-system blockchain architecture: hardware, storage, communication, data manipulation, and contract. We have also discussed the performance of the four systems based on the previously published information. The study has emphasized the differences in the design goals and principles between the systems. We hope that the study can be used as educational material in courses and tutorials. It is our conjecture that this first cross-system comparison will facilitate similar studies in the future, and that such studies will collectively contribute towards standardization of the area.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

Acknowledgments

This work was supported by the Research Council of Norway through IKTPLUSS Program under Grant 288106 and through INTPART Program under Grant 288126. We would like to thank the editor and anonymous reviewers for providing many helpful and insightful comments and suggestions towards improving the manuscript. We are deeply grateful to Robbert van Renesse for his continuous support and encouragement to work on this topic and for attentively proofreading many of the sections and giving valuable feedback. Hein Meling posed questions about the early versions of this work that contributed to shaping the objectives and structure of the paper. Leander Jehl provided suggestions for improving the technical descriptions of Ethereum and IOTA. Several members of the Blockchain Lab at the University of Oslo (Salar Arbabi, Arlindo Da Conceição, Daniel Gerbi Duguma, Thiago Garrent, Andrea Merlina, and Praveensankar Manimaran) provided much appreciated feedback on the manuscript. Finally, we would like to thank many researchers who encouraged us to continue working on the manuscript and expressed their potential interest in using it as education material in their courses.

References

[1] Coindesk, State of blockchain- Q4 2017 enterprise blockchain summary, 2018, https://s3-us-west-2.amazonaws.com/guizishanren/pdf/Coindesk-201712-State-of-Blockchain-2018.pdf.

[2] A. Rana, Is blockchain the solution for healthcare? - dataconomy, 2017, https://dataconomy.com/2017/03/blockchain-solution-healthcare/.

[3] F. Hasse, A. von Perfall, T. Hillebrand, E. Smole, L. Lay, M. Charlet, Blockchain–an opportunity for energy producers and consumers, PwC Glob. Power Util. (2016) 1–45.

[4] P. Windley, D. Reed, Sovrin: A protocol and token for self-sovereign identity and decentralized trust, Whitepaper, the Sovrin Foundation, 2018.

[5] M.M. Lab, What we learned from designing an academic certificates system on the blockchain, 2016, Medium https://medium.com/mit-media-lab/what-we-learned-from-designing-an-academic-certificates-system-on-the-blockchain-34ba5874f196.

[6] CoreLedger, Land registry on blockchain, 2020, Medium https://medium.com/coreledger/land-registry-on-blockchain-a0da4dd25ea6.

[7] Blockchain for supply chain - IBM blockchain 2021, 2021, https://www.ibm.com/blockchain/industries/supply-chain.

[8] M. Gray, Ethereum blockchain as a service now on Azure, 2015, https://azure.microsoft.com/en-us/blog/ethereum-blockchain-as-a-service-now-on-azure/.

[9] P. Rizzo, IBM goes big on blockchain unveiling services suite and strategy, 2016, CoinDesk https://www.coindesk.com/ibm-goes-big-on-blockchain-unveiling-ambitious-new-service-offerings-and-strategy.

[10] The diem association, 2023, https://www.diem.com/en-us/. (last Accessed on Jul 2023).

[11] R. Wattenhofer, The Science of the Blockchain, Inverted Forest Publishing, 2016.

[12] M. Huillet, Dutch ministry develops national blockchain research agenda, 2018, Cointelegraph https://cointelegraph.com/news/dutch-ministry-develops-national-blockchain-research-agenda.

[13] Blockchain partnership | shaping Europe's digital future, 2023, https://digital-strategy.ec.europa.eu/en/policies/blockchain-partnership. (last Accessed on Jul 2023).

[14] European Blockchain Services Infrastructure (EBSI), CEF Digital, 2023, https://ec.europa.eu/digital-building-blocks/wikis/display/EBSI/Home. (last Accessed on Jul 2023).

[15] P. Zhou, Y. Du, B. Li, et al., China blockchain technology and application development white paper, in: Ministry of Industry and Information Technology of the People's Republic of China, Ed. China Blockchain Technology and Application, 2016.

[16] M. Stone, The tiny European country that became a global leader in digital government, 2016, Forbes https://www.forbes.com/sites/delltechnologies/2016/06/14/the-tiny-european-country-that-became-a-global-leader-in-digital-government/.

[17] A. Ellervee, R. Matulevicius, N. Mayer, A comprehensive reference model for blockchain-based distributed ledger technology, in: ER Forum/Demos, 2017, pp. 306–319.

[18] M. Belotti, N. Božić, G. Pujolle, S. Secci, A vademecum on blockchain technologies: When, which, and how, IEEE Commun. Surv. Tutor. 21 (4) (2019) 3796–3838.

[19] J. Bonneau, A. Miller, J. Clark, A. Narayanan, J.A. Kroll, E.W. Felten, Sok: Research perspectives and challenges for bitcoin and cryptocurrencies, in: 2015 IEEE Symposium on Security and Privacy, IEEE, 2015, pp. 104–121.

[20] F. Tschorsch, B. Scheuermann, Bitcoin and beyond: A technical survey on decentralized digital currencies, IEEE Commun. Surv. Tutor. 18 (3) (2016) 2084–2123.

[21] Q. Wang, J. Yu, S. Chen, Y. Xiang, SoK: Diving into DAG-based blockchain systems, 2020.

[22] A. Narayanan, J. Bonneau, E. Felten, A. Miller, S. Goldfeder, Bitcoin and cryptocurrency technologies: A comprehensive introduction, Princeton University Press, 2016.

[23] R. Browne, A Little-Known Digital Currency Surges 90% After Teaming Up with Firms Like Microsoft, CNBC, 2017, https://www.cnbc.com/2017/12/04/cryptocurrency-iota-rallies-after-launch-of-data-marketplace.html.

[24] B. McElrath, Braiding the blockchain, in: Presentation at Scaling Bitcoin Hong Kong (7 December 2015), 2015, https://scalingbitcoin.org/hongkong2015/presentations/DAY2/2_breaking_the_chain_1_mcelrath.pdf.

[25] M. Hearn, R.G. Brown, Corda: A distributed ledger, Corda Technical White Paper, 2019.

[26] Chainlink, Smart contract platforms, 2023, Chainlink Blog, https://blog.chain.link/smart-contract-platforms/.

[27] The reality of blockchain, 2019, Gartner, https://www.gartner.com/smarterwithgartner/the-reality-of-blockchain.

[28] D. Canellis, Bitcoin has nearly 100,000 nodes, but over 50% run vulnerable code, 2019, https://thenextweb.com/news/bitcoin-100000-nodes-vulnerable-cryptocurrency.

[29] S. Delgado-Segura, C. Pérez-Solà, J. Herrera-Joancomartí, G. Navarro-Arribas, J. Borrell, Cryptocurrency networks: A new P2P paradigm, Mob. Inf. Syst. (2018) 1–16.

[30] M. Vukolić, The quest for scalable blockchain fabric: Proof-of-work vs. BFT replication, in: Open Problems in Network Security, Springer International Publishing, 2016, pp. 112–125.

[31] S. Popov, The tangle, 2018.

[32] S. Nakamoto, Bitcoin: A peer-to-peer electronic cash system, Citeseer, 2008, http://bitcoin.org/bitcoin.pdf.

[33] Endorsement policies, 2020, https://hyperledger-fabric.readthedocs.io/en/release-2.0/developapps/endorsementpolicies.html.

[34] Ethash, 2018, https://github.com/ethereum/wiki/wiki/Ethash.

[35] Why a GPU mines faster than a CPU, 2013, https://en.bitcoin.it/wiki/Why_a_GPU_mines_faster_than_a_CPU.

[36] M. Alshaikhli, T. Elfouly, O. Elharrouss, A. Mohamed, N. Ottakath, Evolution of Internet of Things from blockchain to IOTA: A survey, IEEE Access 10 (2021) 844–866.

[37] Jinn, 2023, https://iotanodes.org/jinn/. (last Accessed on Jul 2023).

[38] Proof of scarcity and Sybil attacks, 2018, Hello IOTA. https://helloiota.com/proof-of-scarcity-and-sybil-attacks/.

[39] J. Lind, O. Naor, I. Eyal, F. Kelbert, E.G. Sirer, P. Pietzuch, Teechain: A secure payment network with asynchronous blockchain access, in: Proceedings of the 27th ACM Symposium on Operating Systems Principles, 2019, pp. 63–79.

[40] F. McKeen, I. Alexandrovich, A. Berenzon, C.V. Rozas, H. Shafi, V. Shanbhogue, U.R. Savagaonkar, Innovative instructions and software model for isolated execution, in: Hasp@ isca, Vol. 10, 2013.

[41] E. Androulaki, A. Barger, V. Bortnikov, et al., Hyperledger fabric: A distributed operating system for permissioned blockchains, in: Proceedings of the 13th EuroSys Conference, 2018, pp. 1–15.

[42] IOTA reference implementation, 2020, https://github.com/iotaledger/iri.

[43] Transaction - bitcoin wiki, 2023, https://en.bitcoin.it/wiki/Transaction. (last Accessed on Jul 2023).

[44] Python Implementation of the Ethereum protocol_2023, ethereum, 2023, https://github.com/ethereum/py-evm/blob/1af151ab218b905f4fdf7a285cbe14ebf094a7c4/eth/rlp/transactions.py.

[45] H. Mayer, ECDSA security in bitcoin and ethereum: A research survey, CoinFaabrik 28 (126) (2016) 50.

[46] Hyperledger fabric block file manager, 2023, https://github.com/hyperledger/fabric/blob/main/common/ledger/blkstorage/blockfile_mgr.go. (last Accessed on Jul 2023).

[47] Bitcoin core 0.11 (ch 2): Data storage - bitcoin wiki, 2020, https://en.bitcoin.it/wiki/Bitcoin_Core_0.11_(ch_2):_Data_Storage.

[48] GitHub fabric storage source code, 2023, https://github.com/hyperledger/fabric/blob/86f87ebcd8a1819e3832b289f2d94d4a88bb6e05/common/ledger/blkstorage/fsblkstorage/blockindex.go. (last Accessed on Jul 2023).

[49] S. Delgado-Segura, C. Pérez-Sola, G. Navarro-Arribas, J. Herrera-Joancomartí, Analysis of the bitcoin utxo set, in: International Conference on Financial Cryptography and Data Security, Springer, 2018, pp. 78–91.

[50] CouchDB as the state database – hyperledger-fabricdocs master documentation, 2023, https://hyperledger-fabric.readthedocs.io/en/release-2.0/couchdb_as_state_database.html. (last Accessed on Jul 2023).

[51] B. Prahalad, Merkle proofs explained, 2018, Crypto-0-nite, https://medium.com/crypto-0-nite/merkle-proofs-explained-6dd429623dc5.

[52] V. Buterin, State tree pruning, 2015, https://ethereum.github.io/blog/2015/06/26/state-tree-pruning/.

[53] Jamesob, Coins: allow Flush without cache drop, Bitcoin Core PR Review Club, 2020, https://bitcoincore.reviews/17487.

[54] What's new in Hyperledger Fabric v2.x – hyperledger-fabricdocs master documentation, 2023, https://hyperledger-fabric.readthedocs.io/en/release-2.2/whatsnew.html. (last Accessed on Jul 2023).

[55] Tangle-accelerator, DLTcollab, 2021, https://github.com/DLTcollab/tangle-accelerator.

[56] R. Rottmann, IOTA snapshot, 2019, https://iota-news.com/iota-snapshot/.

[57] A. Gervais, S. Capkun, G.O. Karame, D. Gruber, On the privacy provisions of bloom filters in lightweight bitcoin clients, in: Proceedings of the 30th Annual Computer Security Applications Conference, 2014, pp. 326–335.

[58] Bitcoin wiki, 2023, https://en.bitcoin.it/wiki/Protocol_documentation. (last Accessed on Jul 2023).

[59] C. Decker, R. Wattenhofer, Information propagation in the bitcoin network, in: IEEE P2P Proceedings, 2013, pp. 1–10.

[60] T. Wang, C. Zhao, Q. Yang, S. Zhang, S.C. Liew, Ethna: Analyzing the underlying peer-to-peer network of ethereum blockchain, IEEE Trans. Netw. Sci. Eng. 8 (3) (2021) 2131–2146.

[61] P. Silva, D. Vavricka, J. Barreto, M. Matos, Impact of geo-distribution and mining pools on blockchains: A study of ethereum, in: 2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN, IEEE, 2020, pp. 245–252.

[62] Gossip data dissemination protocol – hyperledger-fabricdocs master documentation, 2023, https://hyperledger-fabric.readthedocs.io/en/release-2.0/gossip.html. (last Accessed on Jul 2023).

[63] The ordering service – hyperledger-fabricdocs master documentation, 2023, https://hyperledger-fabric.readthedocs.io/en/release-2.0/orderer/ordering_service.html. (last Accessed on Jul 2023).

[64] Architecture explained – hyperledger-fabricdocs master documentation, 2023, https://hyperledger-fabric.readthedocs.io/en/release-1.3/arch-deep-dive.html. (last Accessed on Jul 2023).

[65] A.E. Gencer, S. Basu, I. Eyal, R. van Renesse, E.G. Sirer, Decentralization in bitcoin and ethereum networks, in: International Conference on Financial Cryptography and Data Security, 2018, pp. 439–457.

[66] Whisper - ethereum wiki, 2023, GitHub, https://github.com/ethereum/wiki/wiki/Whisper. (last Accessed on Jul 2023).

[67] P. Syverson, R. Dingledine, N. Mathewson, Tor: The second generation onion router, in: Usenix Security, 2004.

[68] I. Latino, MAM protocol: Alternative messaging thanks to the IOTA platform, 2018, Medium, https://medium.com/iotalatinoamerica/mam-protocol-alternative-messaging-thanks-to-the-iota-platform-f92a5ef92ebe.

[69] C. Cachin, M. Vukolić, Blockchain consensus protocols in the wild, 2017.

[70] Bitcoin cash, 2023, https://www.bitcoincash.org/. (last Accessed on Jul 2023).

[71] E. Lombrozo, J. Lau, P. Wuille, Segregated witness (consensus layer), 2015, https://github.com/bitcoin/bips/blob/master/bip-0141.mediawiki.

[72] Orphan block, 2023, https://en.bitcoin.it/wiki/Orphan_Block. (last Accessed on Jul 2023).

[73] Y. Kawase, S. Kasahara, Priority queueing analysis of transaction-confirmation time for bitcoin, J. Ind. Manag. Optim. 16 (3) (2020) 1077.

[74] Y. Sompolinsky, A. Zohar, Secure high-rate transaction processing in bitcoin, in: International Conference on Financial Cryptography and Data Security, Springer, 2015, pp. 507–527.

[75] I. Eyal, A.E. Gencer, E.G. Sirer, R. van Renesse, Bitcoin-ng: A scalable blockchain protocol, in: 13th {*USENIX*} Symposium on Networked Systems Design and Implementation ({*NSDI*} 16), 2016, pp. 45–59.

[76] V. Bagaria, S. Kannan, D. Tse, G. Fanti, P. Viswanath, Prism: Deconstructing the blockchain to approach physical limits, in: Proceedings of the 2019 ACM SIGSAC Conference on CCS, 2019, pp. 585–602.

[77] I. Eyal, E.G. Sirer, Majority is not enough: Bitcoin mining is vulnerable, Commun. ACM 61 (7) (2018) 95–102.

[78] C.S. Wright, S. Savanah, The fallacy of the selfish miner in bitcoin: An economic critique, 2017, https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3009466.

[79] Marlow, Why Selfish miners do not exist and never will, 2018, Medium, https://medium.com/ProfFaustus/why-selfish-miners-do-not-exist-and-never-will-6d68b5ad571c.

[80] E. Conner, Understanding ethereum gas, blocks and the fee market, 2019, Medium, https://medium.com/eric.conner/understanding-ethereum-gas-blocks-and-the-fee-market-d5e268bf0a0e.

[81] Ethereum block size chart, 2023, BitInfoCharts, https://bitinfocharts.com/comparison/ethereum-size.html. (last Accessed on Jul 2023).

[82] V. Buterin, Ethereum: A next-generation smart contract and decentralized application platform, 2014, https://ethereum.org/en/whitepaper.

[83] V. Buterin, EIP 100: Change difficulty adjustment to target mean block time including uncles, 2016, Ethereum Improvement Proposals, https://eips.ethereum.org/EIPS/eip-100.

[84] V. Gramoli, From blockchain consensus back to byzantine consensus, Future Gen. Comput. Syst. (2017) (2017).

[85] I. Weber, V. Gramoli, A. Ponomarev, M. Staples, R. Holz, A.B. Tran, P. Rimba, On availability for blockchain-based systems, in: 36th Symposium on Reliable Distributed Systems, SRDS, IEEE, 2017, pp. 64–73.

[86] A. Hertig, How will ethereum scale?, 2018, https://www.coindesk.com/information/will-ethereum-scale.

[87] K. Croman, C. Decker, I. Eyal, A.E. Gencer, A. Juels, A. Kosba, A. Miller, P. Saxena, E. Shi, E.G. Sirer, et al., On scaling decentralized blockchains, in: International Conference on Financial Cryptography and Data Security, 2016, pp. 106–125.

[88] C. Gorenflo, S. Lee, L. Golab, S. Keshav, Fastfabric: Scaling hyperledger fabric to 20,000 transactions per second, in: International Conference on Blockchain and Cryptocurrency, ICBC, IEEE, 2019, pp. 455–463.

[89] D. Ongaro, J. Ousterhout, In search of an understandable consensus algorithm, in: 2014 USENIX ATC, 2014.

[90] M. Castro, B. Liskov, et al., Practical Byzantine fault tolerance, in: OSDI, Vol. 99, no. 1999, 1999, pp. 173–186.

[91] N. Garg, Apache Kafka, Packt Publishing Ltd, 2013.

[92] A. Bessani, J. Sousa, E.E. Alchieri, State machine replication for the masses with BFT-SMaRt, in: 2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, IEEE, 2014, pp. 355–362.

[93] G.G. Gueta, I. Abraham, S. Grossman, D. Malkhi, B. Pinkas, M.K. Reiter, D. Seredinschi, O. Tamir, A. Tomescu, SBFT: A scalable decentralized trust infrastructure for blockchains, 2018, arXiv preprint arXiv:1804.01626.

[94] R. van Renesse, N. Schiper, F.B. Schneider, Vive la différence: Paxos vs. viewstamped replication vs. zab, IEEE Trans. Dependable Secure Comput. 12 (4) (2014) 472–484.

[95] P. Thakkar, S. Nathan, B. Viswanathan, Performance benchmarking and optimizing hyperledger fabric blockchain platform, in: MASCOTS, IEEE, 2018, pp. 264–276.

[96] C. Gorenflo, L. Golab, S. Keshav, XOX fabric: A hybrid approach to blockchain transaction execution, in: International Conference on Blockchain and Cryptocurrency, ICBC, IEEE, 2020, pp. 1–9.

[97] P. Ryszkiewicz, 2023, GitHub https://github.com/pRizz/iota-transaction-spammer-webapp. (last Accessed on Jul 2023).

[98] G. De Roode, I. Ullah, P.J. Havinga, How to break IOTA heart by replaying? in: 2018 IEEE Globecom Workshops, GC Wkshps, IEEE, 2018, pp. 1–7.

[99] A. Penzkofer, B. Kusmierz, A. Capossele, W. Sanders, O. Saa, Parasite chain detection in the IOTA protocol, 2020, arXiv preprint arXiv:2004.13409.

[100] B. Kusmierz, A. Gal, Probability of being left behind and probability of becoming permanent tip in the Tangle v0. 2, IOTA Foundation, 2018.

[101] G. Bu, Ö. Gürcan, M. Potop-Butucaru, G-IOTA: Fair and confidence aware tangle, in: IEEE INFOCOM 2019-IEEE Conference on Computer Communications Workshops, INFOCOM WKSHPS, IEEE, 2019, pp. 644–649.

[102] In tangle we trust - 10 amazing facts about IOTA, 2017, Steemit, https://steemit.com/iota/@steemhoops99/in-tangle-we-trust-10-amazing-facts-about-iota-1000000-token-giveaway-faucet.

[103] A primer on IOTA (with presentation), 2017, IOTA Foundation Blog, http://blog.iota.org/a-primer-on-iota-with-presentation-e0a6eb2cc621.

[104] I.F. Coordicide Team, The coordicide, 2019.

[105] A. De Vries, Bitcoin's growing energy problem, Joule 2 (5) (2018) 801–805.

[106] C.T. Nguyen, D.T. Hoang, D.N. Nguyen, D. Niyato, H.T. Nguyen, E. Dutkiewicz, Proof-of-stake consensus mechanisms for future blockchain networks: Fundamentals, applications and opportunities, 7, IEEE, 2019,

[107] V. Buterin, V. Griffith, Casper the friendly finality gadget, 2019, arXiv preprint arXiv:1710.09437v4.

[108] V. Buterin, D. Reijsbergen, S. Leonardos, G. Piliouras, Incentives in ethereum?s hybrid casper protocol, in: 2019 IEEE International Conference on Blockchain and Cryptocurrency, ICBC, IEEE, 2019, pp. 236–244.

[109] Y. Li, K. Zheng, Y. Yan, Q. Liu, X. Zhou, EtherQL: A query layer for blockchain system, in: International Conference on Database Systems for Advanced Applications, Springer, 2017, pp. 556–567.

[110] Blockchain.com explorer, 2023, https://www.blockchain.com/explorer. (last Accessed on Jul 2023).

[111] Ethereum (ETH) blockchain explorer, 2023, Ethereum (ETH) Blockchain Explorer, http://etherscan.io/. (last Accessed on Jul 2023).

[112] Bitcoin blockchain SQL query, 2023, http://blockchainsql.io/. (last Accessed on Jul 2023).

[113] Anyblock analytics GmbH, 2023, https://github.com/anyblockanalytics. (last Accessed on Jul 2023).

[114] IOTA tangle explorer, 2023, https://explorer.iota.org/mainnet. (last Accessed on Jul 2023).

[115] IOTA tangle explorer and statistics, 2023, https://thetangle.org/. (last Accessed on Jul 2023).

[116] L. Brodie, C. Forth Inc, Starting Forth, Prentice-Hall, Inc., 1987.

[117] Y. Hirai, Programming languages that compile into EVM, 2017, Ethereum Wiki, https://eth.wiki/en/concepts/evm/ethereum-virtual-machine-(evm)-awesome-list.

[118] Solidity is twice as popular as the next blockchain coding language, 2019, Medium, https://media.consensys.net/solidity-is-twice-as-popular-as-the-next-blockchain-coding-language-9330af9aeaa3.

[119] LLL introduction, 2023, https://lll-docs.readthedocs.io/en/latest/lll_introduction.html. (last Accessed on Jul 2023).

[120] Vyper documentation, 2023, https://vyper.readthedocs.io/en/latest/. (last Accessed on Jul 2023).

[121] Smart contracts and chaincode – hyperledger-fabricdocs master documentation, 2023, https://hyperledger-fabric.readthedocs.io/en/release-2.0/smartcontract/smartcontract.html. (last Accessed on Jul 2023).

[122] Chaincode tutorials – hyperledger-fabricdocs master documentation, 2023, https://hyperledger-fabric.readthedocs.io/en/release-2.0/chaincode.html. (last Accessed on Jul 2023).

[123] R. Rottmann, About smart contracts in IOTA, 2018, Medium, https://medium.com/ralf/about-smart-contracts-in-iota-626d2bd3619e.

[124] IOTA qubic, 2023, https://iota-news.com/about-qubic/. (last Accessed on Jul 2023).

[125] C. Natoli, J. Yu, V. Gramoli, P. Esteves-Verissimo, Deconstructing blockchains: A comprehensive survey on consensus, membership and structure, 2019, arXiv preprint arXiv:1908.08316.

[126] Y. Xiao, N. Zhang, W. Lou, Y.T. Hou, A survey of distributed consensus protocols for blockchain networks, IEEE Commun. Surv. Tutor. 22 (2) (2020) 1432–1465.

[127] S. Gupta, J. Hellings, M. Sadoghi, Fault-tolerant distributed transactions on blockchain, Synth. Lect. Data Manag. 16 (1) (2021) 1–268.

[128] R. Zhang, R. Xue, L. Liu, Security and privacy on blockchain, ACM Comput. Surv. 52 (3) (2019) 1–34.

[129] T. Salman, M. Zolanvari, A. Erbad, R. Jain, M. Samaka, Security services using blockchains: A state of the art survey, IEEE Commun. Surv. Tutor. 21 (1) (2018) 858–880.

[130] M. Saad, J. Spaulding, L. Njilla, C. Kamhoua, S. Shetty, D. Nyang, D. Mohaisen, Exploring the attack surface of blockchain: A comprehensive survey, IEEE Commun. Surv. Tutor. 22 (3) (2020) 1977–2008.

[131] I. Homoliak, S. Venugopalan, D. Reijsbergen, Q. Hum, R. Schumi, P. Szalachowski, The security reference architecture for blockchains: Toward a standardized model for studying vulnerabilities, threats, and defenses, IEEE Commun. Surv. Tutor. 23 (1) (2020) 341–390.

[132] M. Bartoletti, L. Pompianu, An empirical analysis of smart contracts: platforms, applications, and design patterns, in: International Conference on Financial Cryptography and Data Security, Springer, 2017, pp. 494–509.

[133] Á.J. Varela-Vaca, A.M.R. Quintero, Smart contract languages: A multivocal mapping study, ACM Comput. Surv. 54 (1) (2021) 1–38.

[134] J. Eberhardt, S. Tai, On or off the blockchain? Insights on off-chaining computation and data, in: European Conference on Service-Oriented and Cloud Computing, Springer, 2017, pp. 3–15.

[135] H.T. Vo, A. Kundu, M.K. Mohania, Research directions in blockchain data management and analytics, in: EDBT, 2018, pp. 445–448.

[136] T. Neudecker, H. Hartenstein, Network layer aspects of permissionless blockchains, IEEE Commun. Surv. Tutor. 21 (1) (2018) 838–857.

[137] R. Yang, F.R. Yu, P. Si, Z. Yang, Y. Zhang, Integrated blockchain and edge computing systems: A survey, some research issues and challenges, IEEE Commun. Surv. Tutor. 21 (2) (2019) 1508–1532.

[138] Y. Liu, F.R. Yu, X. Li, H. Ji, V.C. Leung, Blockchain and machine learning for communications and networking systems, IEEE Commun. Surv. Tutor. 22 (2) (2020) 1392–1431.

[139] P. Sharma, R. Jindal, M.D. Borah, Blockchain technology for cloud storage: A systematic literature review, ACM Comput. Surv. 53 (4) (2020) 1–32.

[140] K. Gai, J. Guo, L. Zhu, S. Yu, Blockchain meets cloud computing: A survey, IEEE Commun. Surv. Tutor. 22 (3) (2020) 2009–2030.

[141] D.C. Nguyen, P.N. Pathirana, M. Ding, A. Seneviratne, Integration of blockchain and cloud of things: Architecture, applications and challenges, IEEE Commun. Surv. Tutor. 22 (4) (2020) 2521–2549.

[142] L. Lao, Z. Li, S. Hou, B. Xiao, S. Guo, Y. Yang, A survey of IoT applications in blockchain systems: Architecture, consensus, and traffic modeling, ACM Comput. Surv. 53 (1) (2020) 1–32.

[143] M.S. Ali, M. Vecchio, M. Pincheira, K. Dolui, F. Antonelli, M.H. Rehmani, Applications of blockchains in the Internet of Things: A comprehensive survey, IEEE Commun. Surv. Tutor. 21 (2) (2018) 1676–1717.

[144] E.J. De Aguiar, B.S. Faiçal, B. Krishnamachari, J. Ueyama, A survey of blockchain-based strategies for healthcare, ACM Comput. Surv. 53 (2) (2020) 1–27.

[145] J. Xie, H. Tang, T. Huang, F.R. Yu, R. Xie, J. Liu, Y. Liu, A survey of blockchain technology applied to smart cities: Research issues and challenges, IEEE Commun. Surv. Tutor. 21 (3) (2019) 2794–2830.

[146] X. Xu, I. Weber, M. Staples, L. Zhu, J. Bosch, L. Bass, C. Pautasso, P. Rimba, A taxonomy of blockchain-based systems for architecture design, in: 2017 IEEE International Conference on Software Architecture, ICSA, IEEE, 2017, pp. 243–252.

[147] J. Kolb, M. AbdelBaky, R.H. Katz, D.E. Culler, Core concepts, challenges, and future directions in blockchain: A centralized tutorial, ACM Comput. Surv. 53 (1) (2020) 1–39.

[148] F. Haffke, Technical analysis of established blockchain systems, (Master's thesis), TU Munich, 2017.

[149] J. De Kruijff, H. Weigand, Understanding the blockchain using enterprise ontology, in: International Conference on Advanced Information Systems Engineering, Springer, 2017, pp. 29–43.

[150] P. Tasca, T. Thanabalasingham, C.J. Tessone, Ontology of blockchain technologies. Principles of identification and classification, 10, 2017,

[151] M. Valenta, P. Sandner, Comparison of ethereum, hyperledger fabric and corda, Vol. 8, Frankfurt School Blockchain Center, 2017.

[152] T.T.A. Dinh, R. Liu, M. Zhang, G. Chen, B.C. Ooi, J. Wang, Untangling blockchain: A data processing view of blockchain systems, IEEE Trans. Knowl. Data Eng. 30 (7) (2018) 1366–1385.

[153] N. Kannengießer, S. Lins, T. Dehling, A. Sunyaev, Trade-offs between distributed ledger technology characteristics, ACM Comput. Surv. 53 (2) (2020) 1–37.

[154] A. Gervais, G.O. Karame, K. Wüst, V. Glykantzis, H. Ritzdorf, S. Capkun, On the security and performance of proof of work blockchains, in: Proceedings of the 2016 ACM SIGSAC Conference on CCS, 2016, pp. 3–16.