Trade-offs between Distributed Ledger Technology Characteristics

NICLAS KANNENGIEßER, Institute of Applied Informatics and Formal Description Methods, Karlsruhe Institute of Technology and Blockchain Center EU, University of Kassel
SEBASTIAN LINS, TOBIAS DEHLING, and ALI SUNYAEV, Institute of Applied Informatics and Formal Description Methods, Karlsruhe Institute of Technology

When developing peer-to-peer applications on distributed ledger technology (DLT), a crucial decision is the selection of a suitable DLT design (e.g., Ethereum), because it is hard to change the underlying DLT design post hoc. To facilitate the selection of suitable DLT designs, we review DLT characteristics and identify trade-offs between them. Furthermore, we assess how DLT designs account for these trade-offs and we develop archetypes for DLT designs that cater to specific requirements of applications on DLT. The main purpose of our article is to introduce scientific and practical audiences to the intricacies of DLT designs and to support development of viable applications on DLT.

CCS Concepts: • **Software and its engineering** → **Designing software**; • **Networks** → *Network protocols*; *Application layer protocols*;

Additional Key Words and Phrases: Blockchain, distributed ledger technology, peer-to-peer, application development, suitability, viability

ACM Reference format:
Niclas Kannengießer, Sebastian Lins, Tobias Dehling, and Ali Sunyaev. 2020. Trade-offs between Distributed Ledger Technology Characteristics. *ACM Comput. Surv.* 53, 2, Article 42 (May 2020), 37 pages.
https://doi.org/10.1145/3379463

1 INTRODUCTION

Distributed ledger technology (DLT) enables the operation of a highly available, append-only database (a distributed ledger) that is maintained by physically distributed storage and computing devices (referred to as nodes) in an untrustworthy environment. DLT promises to increase efficiency and transparency of collaborations between individuals and/or organizations based on inherent qualities such as tamper resistance and censorship resistance and democratization of data [1]. As

This work was partially funded by research grants for the projects Trusted Blockchain (Metamorphoses GmbH, Munich, Germany, privately funded) and COOLedger (Helmholtz Association of German Research Centers: HRSF-0081, Russian Science Foundation: Project No. 19-41-06301). Authors' addresses: N. Kannengießer, Institute of Applied Informatics and Formal Description Methods, Karlsruhe Institute of Technology, Karlsruhe, Germany and Blockchain Center EU, University of Kassel, Kassel, Germany; email: niclas.kannengiesser@kit.edu; S. Lins, Institute of Applied Informatics and Formal Description Methods, Karlsruhe Institute of Technology, Karlsruhe, Germany; email: sebastian.lins@kit.edu; T. Dehling and A. Sunyaev, Institute of Applied Informatics and Formal Description Methods, Karlsruhe Institute of Technology, Karlsruhe, Germany; emails: {Pdehling, sunyaev}@kit.edu.

0360-0300/2020/05-ART42
https://doi.org/10.1145/3379463

a consequence, an ever-increasing number of applications on DLT have been developed in various domains, such as supply chain management [2], finance [3], or health care [4]. In supply chain management, product provenance systems employ DLT, for example, as a tamper-resistant data storage that is replicated across multiple nodes of collaborating entities in the supply chain [5, 6]. Applications use distributed ledgers as a shared infrastructure that facilitates, for instance, reliable and tamper-resistant data storage, processing of transactions (e.g., for the transfer of digital assets), and automation of business processes [7, 8]. Each application on DLT builds upon a particular DLT design (e.g., Ethereum or IOTA) that is defined as a formal specification of a DLT concept (e.g., blockchain) with unique characteristics [9].

Despite the promising benefits of DLT, past implementations of applications on DLT reveal critical dependencies between DLT characteristics that result in trade-offs; that is, the improvement of one DLT characteristic interferes with another DLT characteristic. For example, a trade-off exists between achieving availability and consistency in distributed ledgers [10]. High availability of a distributed ledger can be achieved by increasing the number of replications of the ledger. As a consequence, the network of nodes in the distributed ledger increases; however, this leads to reduced consistency due to increased message propagation delays [11]. Given the prevalent trade-offs between DLT characteristics, there will be no one-size-fits-all DLT design for applications on DLT. Rather, there will be DLT designs that are specialized to fulfill certain requirements but perform poorly on other requirements (e.g., low throughput, poor scalability, or high cost) due to drawbacks resulting from DLT-inherent trade-offs [9, 12]. It is thus challenging to select suitable DLT designs for an application and to assess potential drawbacks for the respective application on DLT. Making careful and well-founded decisions in favor for a (suitable) DLT design to develop viable applications on DLT is even more crucial, because technical differences between DLT designs (e.g., different data structures and consensus mechanisms) impede the migration of data between distributed ledgers [13]. In this context, viability refers to applications' ability to operate over a long period under consideration of potentially changing requirements or improvements and resulting updates. To understand the trade-offs between DLT characteristics and their impact on the viability of applications on DLT, a comprehensive analysis of dependencies between DLT characteristics and resulting trade-offs is required.

While the body of research on DLT was ever increasing in the past decade, related research on DLT characteristics predominantly focuses on assessing the importance of characteristics for particular use cases (e.g., cryptocurrencies [14]) and on comparing application requirements with capabilities of selected distributed ledgers (e.g., [15]). Prior analyses of dependencies between DLT characteristics only consider a sparse set of DLT characteristics (e.g., integrity or scalability [16, 17]). In addition, research on DLT characteristics and their dependencies is largely scattered across disciplines and needs to be synthesized to obtain a comprehensive understanding of dependencies between DLT characteristics and resulting trade-offs that limit the applicability of DLT designs to certain applications on DLT. We therefore strive to answer the following research question:

How do trade-offs between DLT characteristics impact the viability of applications on DLT?

To answer our research question, we applied a three-step research approach. First, we identified prevalent DLT characteristics by conducting a comprehensive literature review composed of 191 articles and surveying DLT experts. Second, we analyzed the identified DLT characteristics in detail to uncover trade-offs in DLT designs, which were then applied to the most fitting DLT designs. Finally, we consolidated the identified trade-offs into archetypes and derived implications for applications on DLT.

Our study identified a consolidated list of 40 DLT characteristics that are fundamental for assessing the suitability of DLT designs for applications on DLT, which we grouped into 6 DLT properties. This manuscript uncovers and explains 24 trade-offs between DLT characteristics and discusses the resulting drawbacks for applications. The identified DLT characteristics and properties range from purely technical (e.g., *strength of cryptography* in *security*) to social (e.g., *degree of decentralization* in *policy*), which highlight the complexity of DLT. Finally, we consolidated our findings into six DLT archetypes that indicate benefits and drawbacks for applications on DLT resulting from choice and configuration of a DLT design optimized toward a certain DLT property.

Our work contributes to the development of viable applications on DLT by discussing benefits and drawbacks of applications on DLT and presenting six archetypes of DLT designs. This work forms a bridge between currently separated research streams on DLT and forms a foundation for research assessing the suitability of DLT designs for applications. Our work allows practitioners and researchers to better understand which drawbacks for applications on DLT result from what configurations of DLT characteristics. Overall, we contribute to the scientific knowledge base by making it possible to set DLT characteristics into relation with applications on DLT and vice versa.

The manuscript is structured as follows: First, we introduce the current state of research on DLT and outline smart contract vulnerabilities and several attacks on distributed ledgers. This knowledge is required to understand the origins of trade-offs between DLT characteristics and drawbacks for applications on DLT. Second, we describe the methods applied. Third, we present the identified DLT characteristics, the derived trade-offs between DLT characteristics, and the generated archetypes. Finally, we discuss our principal findings, summarize the implications for both practice and research, discuss research limitations, and give an outlook for future research.

2 RESEARCH BACKGROUND AND RELATED RESEARCH

2.1 Distributed Ledger Technology

In its essence, DLT serves as a shared, digital infrastructure for applications on DLT (e.g., in financial transactions [18]) by enabling the operation of a highly available, append-only distributed database (referred to as distributed ledger) in an untrustworthy environment [19], where separated storage and computing devices (referred to as nodes) maintain a local replication of the ledger. Nodes are maintained and controlled by individuals or organizations (referred to as node controllers[1]). An untrustworthy environment is characterized by the arbitrary occurrence of Byzantine failures [20, 21], including crashed or (temporarily) unreachable nodes, network delays, and malicious behavior of nodes.

In DLT, data are transferred and appended to the ledger in the form of transactions and are stored in a chronologically ordered sequence. Each transaction contains metadata (e.g., transaction recipient or timestamp) and a digital representation of certain assets (e.g., coins) or program code of a smart contract (see Section 2.3) [22]. When a node receives a new transaction, the transaction is validated by a proof of ownership for the digital representation of the asset based on digital signatures and public key cryptography [22, 23].

DLT covers various DLT concepts, DLT designs, DLT properties, and DLT characteristics [9, 24] (see Figure 1). ***DLT concepts*** describe the basic structure and functioning of DLT designs on a high level of abstraction. For instance, blockchain is a DLT concept describing the use of blocks that form a linked list. Each block contains multiple transactions that have been added into the block by nodes. Blockchains mostly follow the concept of replicated state machines, where each node maintains a local replication of the ledger in a certain state s_n with an incrementing counter $n \in \mathbb{N}_0$, which expresses the height of a ledger (also called *block height* in blockchain). Appending

[1]We prefer the term *node controller* to *node provider*, because the *node provider* could be a cloud service provider (e.g., in Blockchain as a Service) that only hosts the node, while the *node controller* could maliciously influence the behavior of the node.

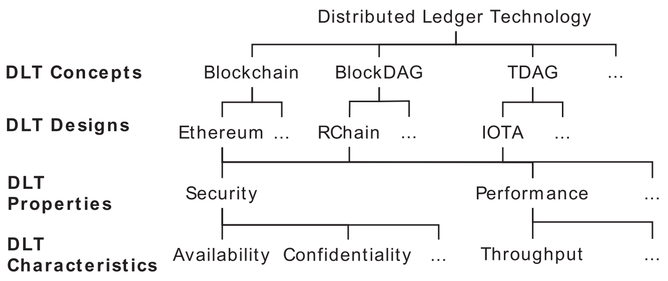

Fig. 1. Hierarchical structure of distributed ledger technology (DLT), subordinate DLT concepts, and DLT designs and their respective DLT properties and DLT characteristics.

new blocks or transactions to the local replication of the ledger represents a transition from a state s_n to the subsequent state s_{n+1}. For example, Alice owns 10 coins in s_n and she sends 2 coins to Bob. The transaction is put in a queue of transactions to be processed and is eventually committed to the ledger. This commit initiates a transition from s_n to s_{n+1}, in which Alice has a new balance of 8 coins and Bob's balance is increased by 2 coins. Other DLT concepts do not rely on generating a single chain of blocks or even do not use blocks at all. For example, the DLT concept BlockDAG links generated blocks in a directed acyclic graph (DAG), while in a transaction-based DAGs (TDAG) transactions are linked directly with each other.[2]

DLT designs specify an abstract description of DLT concepts by adding concrete values and processes for inherent DLT characteristics such as a maximum block size or a consensus mechanism to achieve a certain fault tolerance. There are important differences between DLT designs. The differences make DLT designs suitable for some applications and unsuitable for others. For instance, the DLT design Bitcoin creates a new block every 10 minutes and has a fixed, maximum block size of 1 MB [23]. In contrast, the DLT design Ethereum publishes new blocks on average every 17 seconds and block size is individually decided by nodes to increase flexibility of the distributed ledger. A distributed ledger is an instantiation of the formal specification of a DLT design.

DLT characteristics represent features of DLT designs, which are of technical (e.g., block creation interval) or administrative (e.g., node controller verification) nature. The technical characteristics constrain future changes of the administrative characteristics (e.g., lack of scalability regarding network size of a distributed ledger). ***DLT properties*** are groups of DLT characteristics and shared by each DLT design. For instance, throughput and scalability are both associated with the DLT property performance. Although all DLT designs cover all DLT properties, DLT designs must not cover all DLT characteristics. For instance, TDAGs do not use blocks and do not feature any DLT characteristics related to blocks (e.g., block size, block creation interval).

All nodes of a distributed ledger maintain a local replication of the ledger, which is why all nodes must be synchronized and agree on a common state of the distributed ledger to reach consistency (e.g., agreeing that Bob's balance increased after receiving coins from Alice). For this purpose a consensus mechanism is employed to manage the negotiation between nodes, which (eventually) agree on a common state of the distributed ledger [23, 27]. Consensus mechanisms build upon trust models, which consider threats and uncertainties in the process of consensus finding such as Byzantine failures. Trust models form a set of assumptions, which must hold to ensure consensus finding among nodes (e.g., at least 51% of nodes must agree on a certain state). In Bitcoin, the first

[2]Although blockchain represents a special type of BlockDAG, we decided to separate blockchain from BlockDAGs because of the different validation processes, data structures, and block storage organization [22, 23, 25, 26]. While in blockchain all nodes work on the same block and only one block is appended to the blockchain, in BlockDAGs nodes work on different blocks that are added in parallel.

Table 1. Selection of Relevant Consensus Mechanisms for This Work

Consensus Mechanism	Identifier	DLT Concept	Finality	Exemplary DLT Designs
CBC Casper	Casper	BlockDAG	Probabilistic	RChain [25]
Delegated Proof of Stake	DPoS	Blockchain	Total	EOS [31]
Delegated Proof of Stake	DPoS	TDAG	Probabilistic	Nano [32]
Modified Nakamoto Consensus using Greedy Heaviest Observed Sub Tree (GHOST)	PoW	Blockchain	Probabilistic	Ethereum [22]
Nakamoto Consensus	PoW	Blockchain	Probabilistic	Bitcoin [23]
PHANTOM	PHANTOM	BlockDAG	Probabilistic	soteriaDAG [26]
Practical Byzantine Fault Tolerance	PBFT	Blockchain	Total	Hyperledger Fabric [29]
Proof of Authority	PoA	Blockchain	Total	Ethereum [33]
Proof of Elapsed Time	PoET	Blockchain	Probabilistic	Hyperledger Sawtooth [34]
Proof of Reputation	PoR	Blockchain	Total	GoChain [35]
Proof of Stake	PoS	Blockchain	Probabilistic	Dash [36]
Tendermint Core	Tendermint	Blockchain	Total	Tendermint [30]
Tangle	Tangle	TDAG	Probabilistic	IOTA [37]

Byzantine fault–tolerant consensus mechanism that can be applied on a large scale and is able to prevent double spending (see Section 2.2) was presented: the Proof of Work (PoW)-based Nakamoto consensus [23]. Nevertheless, Nakamoto consensus comes with several drawbacks, such as poor throughput, exhaustive energy consumption, and vulnerability to attacks on integrity (see Section 2.2). To overcome drawbacks of the Nakamoto consensus, numerous alternative consensus mechanisms have been developed and already applied to DLT designs, such as GoChain [28], Hyperledger Fabric [29], soteriaDAG [26], and Tendermint [30]. In addition, BlockDAGs and TDAGs often employ alternatives to replicated state machines, where not all nodes need to maintain an identical replication of the ledger, in their consensus mechanism. Such alternatives make use of random walks (e.g., in IOTA), clustering (e.g., in seele), or only keep transactions of a certain user on individual nodes (e.g., in Nano). The consensus mechanisms discussed in this work are summarized in Table 1.

In large, distributed ledgers (e.g., Bitcoin or Ethereum), where nodes can arbitrarily join and leave the network, it is not possible to reach consensus among all nodes before new data are committed to the ledger [38]. Thus, newly appended data are not finalized and only *probabilistic finality* is given; a certain probability that the data can be altered or removed remains [39]. The probability of finality of a transaction increases with more blocks (or transactions) that are appended to the distributed ledger after the transaction. Accordingly, the trust model of probabilistically final DLT designs (e.g., Bitcoin) allows for network partitions. Some nodes may agree on a state $s_{n,1}$ and others agree on $s_{n,2}$ with $s_{n,1} \neq s_{n,2}$. Network partitions that maintain different states are called *forks*. There can be an arbitrary number of forks in a distributed ledger and the DLT design needs to apply a rule to decide on a block (or transaction) being included into the main branch of the ledger and the ones not being part of it (named *stale blocks* in blockchains and BlockDAGs or *stale transactions* in TDAGs). Fork resolution rules determine a certain state of the ledger to be correct, thereby, returning the distributed ledger to a consistent state. In contrast to probabilistic finality, there is *total finality* (often abbreviated with finality), where all nodes agree on the new state before data are appended to the ledger [40]. Once appended, data cannot be altered or removed anymore and forks such as in Bitcoin or Ethereum are not even possible (see Table 1).

Despite the widespread distinction between public and permissioned DLT designs (e.g., [41, 42]) or public, consortium, and private DLT designs (e.g., [43]), we use a more granular

Table 2. Classification of Exemplary DLT Designs According to the Used Terminology and the Respective Focus

	Public	**Private**
Permissioned	GoChain [28] *High performance general purpose*	Quorum [44] *Financial asset transfers*
Permissionless	Ethereum [22] *General purpose*	ARK Ecosystem [45] *Flexibility for developers*

terminology to make the trade-offs in the following sections unambiguous (in line with [15]). We distinguish between public and private DLT designs depending on the fact whether a new node can directly join a network (referred to as public DLT) or whether a permission must be granted first (referred to as private DLT). The distinction into public-private refers to read permissions and can be further distinguished into permissionless and permissioned, which refers to write permissions. Nodes do not require any permissions to participate in the distributed ledger (referred to as permissionless) or must first be granted permission to validate and commit new data (referred to as permissioned). The used terminology is summarized in Table 2.

In public DLT designs (e.g., Bitcoin), an incentive mechanism is required, because validating nodes must be motivated to share their computational resources. The incentive mechanism specifies a reward scheme for nodes that participate in the generation and/or validation of blocks and transactions, consensus finding, and maintenance of the distributed ledger. The participation of nodes in a distributed ledger to receive a monetary reward is called mining. Accordingly, validating nodes are often referred to as miners. For example, validating nodes in the Bitcoin network receive a certain amount of coins if they are the first to create a valid new block. Such incentive mechanisms are predominantly applied to distributed ledgers that employ nodes of unknown node controllers, thus, allow for a high degree of decentralization.

Assuming that all nodes operate under equal conditions, a distributed ledger's degree of decentralization refers to the number of independent validating node controllers reduced by the number of controllers that control more than average validating nodes divided by the total number of node controllers in the DLT network. Consequently, a distributed ledger's degree of decentralization is determined by two dimensions: the number of independent validating node controllers (e.g., companies or individuals) and the number of validating nodes (see Figure 2). If the number of validating nodes increases, and all additional nodes are maintained by the same controller, the degree of decentralization would decrease, because this controller gains unproportional influence on the distributed ledger's consensus finding and integrity. On the contrary, the degree of decentralization is increased as independent node controllers add nodes of at most average computational resources to the distributed ledger.

2.2 Attack Vectors and Vulnerabilities

To understand drawbacks of applications on DLT that result from vulnerabilities, it is important to introduce potential attack vectors. Although DLT is often considered to be immutable, there have already been millions of dollars lost due to successful attacks on distributed ledgers that rewrote the transaction history (e.g., 51% Attack [46]). In this section, we explain the most prominent attacks on the integrity of a DLT design, which play a role in the identified trade-offs. It should be noted that the explained attacks predominantly target forkable DLT designs (e.g., Bitcoin or Ethereum), because there is only little research on the security of DAGs.

Double Spending. Double spending refers to multiple use of a particular asset by the same user for different purposes without the asset being returned before using it again [23]. In a double

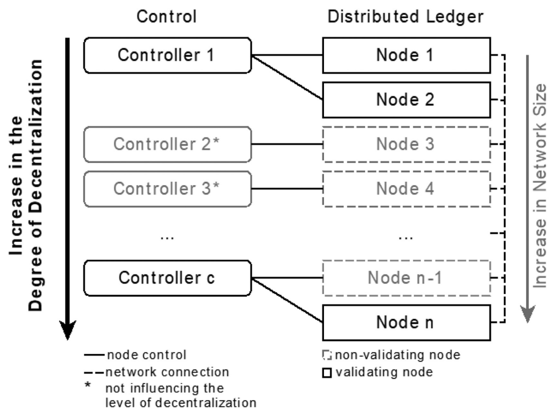

Fig. 2. The degree of decentralization is determined by the number of independent validating node controllers (e.g., an organization or individuum) and the number of validating nodes in the distributed ledger. An increase in number of independent controllers who control validating nodes increases the overall degree of decentralization of a distributed ledger.

spending attack, the attacker suggests to a user that a product was paid on a certain network partition visible for the user, while transferring the issued coins back to her own wallet on another network partition. After the network partitions were resolved, the attacker still owns her coins and the product she actually did not pay for.

Partition-based Attacks. Partition-based attacks can be successfully performed in public-permissionless DLT designs with probabilistic finality (e.g., Bitcoin and Ethereum). In such DLT designs, forks can be exploited to perform attacks on the distributed ledger's integrity. The most popular attacks are 51% attacks, balance attacks, and eclipse attacks. A ***51% attack*** can be successful in DLT designs with a consensus mechanism that relies on a majority decision among nodes (e.g., Nakamoto consensus). If the attackers control the majority of nodes, they can rewrite the transaction history, because their majority of nodes agrees on their desired (fraudulent) state of the distributed ledger. In DLT designs such as Bitcoin, where nodes can arbitrarily join and leave the network, mechanisms are required to prevent attackers from setting up a huge number of virtual nodes. Such mechanisms usually employ proof of work [47], where nodes must first do computational work before new data can be committed to the distributed ledger. A ***balance attack*** incorporates the process of transiently disrupting communications between subgroups of validating nodes with equal computational power, which is determined by, for example, the nodes' hashing rate [48]. While the communication is disrupted, transactions can be submitted to one subgroup while the attacker mines in another subgroup. The attackers' aim is to outweigh the blockchain branch they submitted transactions to with the blockchain branch they work on to raise the probability for successful double spending. As a result, the distributed ledger may be rewritten at any time the attackers prefers [48]. In an ***eclipse attack***, attackers target network partitions by delaying message forwarding (e.g., transactions) to nodes of the attacked network partition [49]. When messages are delayed, targeted nodes are isolated from the network. Such network partitions faciliate double spending (until the fork resolution rule is applied). For example, an attacking node would send a transaction (e.g., a payment) to the victim node. The victim node validates the transaction and is subsequently eclipsed from the network by the attacker. Then, the attacker issues another transaction to the entire network and spends the same assets again. Since all partition-based attacks target information asymmetry among nodes, partition-based attacks can be successfully performed in combination with ***routing attacks*** (i.e., Border Gateway Protocol hijack attack [50]). In routing attacks, attackers manipulate nodes or network operators to intentionally delay network messages [50].

Bribery Attack. During a bribery attack, attackers strive to create a new main branch by incentivizing validating nodes to work on a particular fork of the DLT design, which the attackers dominate [51]. The number of nodes that work on the attackers branch increases and the attackers branch may eventually catch up with the main branch of the blockchain and, finally, become the main branch.

Selfish-mining. Selfish-mining attacks describe a phenomenon where a set of nodes works on their own branch of a blockchain without publishing their blocks to the main branch until their branch would be chosen as future main branch by the fork resolution rule applied to the distributed ledger [16, 52]. A selfish-mining attack is carried out by attackers to obtain excessive rewards or waste the computing power of honest validating nodes [52]. It was found that a successful selfish-mining attack can be performed in Bitcoin if at least one-third of the validating nodes collude [16].

Long-range Attack. A long-range attack aims to rewrite the transaction history from the genesis block (the first block in a blockchain). A long-range attack is successful when the attackers have generated a fork that has become the main chain, which is similar to selfish-mining (or short-range attack) [53]. Predominantly, DLT designs that rely on proof of stake as a consensus mechanism, are prone to long-range attacks, the exhaustive use of PoW in consensus mechanisms (e.g., for leader election in Nakamoto consensus) employed in DLT designs requires too much computational effort to rewrite a transaction history beginning from the genesis block, which is why long-range attacks are considered impractical for PoW-based DLT designs [53].

Blockchain Anomaly. The blockchain anomaly refers to the fact that a blockchain cannot guarantee that a committed transaction is permanently included in a fork of a blockchain. Due to this, conditional transactions are hard to perform. In conditional transactions, a transaction t_{i+1} of a node n_1 should be committed after a certain condition is fulfilled (e.g., the commit of a previous transaction t_i issued by n_2) [27]. If the transactions have been issued by n_1 and n_2 to different network partitions, it is likely that t_i and t_{i+1} are included in different forks of the blockchain. The blockchain may finally decide for the fork not containing t_i but t_{i+1} and only commit t_{i+1}, which violates the conditional execution of t_i and t_{i+1}. The blockchain anomaly can occur in blockchains whose consensus mechanism does not ensure consensus safety and deterministic agreement between nodes [27].

Sybil Attack. In a sybil attack, the attacker sets up multiple (virtual) nodes to contribute the majority of actors in consensus finding to eventually rewrite the transaction history of the distributed ledger. To decrease the probability of successful sybil attacks, all nodes must perform a certain PoW, where each node must first finish a computationally hard task that can easily be evaluated by other nodes [47]. For example, in Bitcoin, Ethereum, and Nano, the block (or transaction) issuer must first guess a nonce with a corresponding hash value, which fulfills an easy-to-validate condition (e.g., starting with a defined minimum number of zeroes). Sybil attacks can isolate (honest) nodes within a distributed ledger's network by not relaying transactions of these nodes [54, 55]. The selective relaying of transactions can contribute to double spending [56].

2.3 Smart Contracts and Respective Vulnerabilities

Several distributed ledgers offer the possibility to deploy and execute customized business logic through smart contracts. Smart contracts are software programs. They can be developed in basic OP_CODE (e.g., in Bitcoin Script language) [57, 58] or in high-level programming languages (e.g., Java, Python, or Solidity), allowing for Turing completeness [22, 31, 59]. When Bitcoin was invented, the development of smart contracts was limited to the use of cryptographic functions such as hash-locks, time-locks, and multi-signatures. To give more flexibility to developers, the Ethereum foundation developed the Ethereum Virtual Machine (EVM), which allows for the execution of Turing-complete smart contracts that can be developed in high-level programming languages such as Solidity [22]. An Ethereum smart contract is contained in a transaction, which is sent to and eventually stored on the Ethereum blockchain. Ethereum smart contracts can receive and keep assets and issue transactions. Smart contracts can be called via their unique address to trigger methods [22]. If an Ethereum smart contract is triggered by a transaction, each node of the distributed ledger separately executes the smart contract. Smart contract computations are not restricted to the use of data already stored on the ledger (on-chain data), but can also retrieve data from external data feeds via oracles and outsource computation-heavy processes to off-chain computing resources (e.g., a cloud service) [60].

Smart contracts are of high interest in the field of DLT, because they considerably increase the range of applications on DLT. However, smart contracts leverage programming paradigms that developers are not yet used to (e.g., rollbacks of failed transactions due to out-of-gas conditions). Since Ethereum introduced Turing-complete smart contracts on *public-permissionless* DLT designs, the issue of how to prevent infinite loops became crucial to prevent system failure. As a solution, a pricing schedule, which requires an economic equivalent (referred to as 'gas' in Ethereum) to be paid for the execution of a particular smart contract, is applied in *public-permissionless* DLT designs [22]. As soon as the quantity of gas is no longer sufficient to execute a smart contract, its execution is cancelled (out-of-gas condition). Out-of-gas conditions always require appropriate error handling; otherwise, the respective smart contract is locked automatically and cannot be executed anymore [61, 62]. In the following, we briefly review the smart contract vulnerabilities relevant for this work.

Overflow/Underflow. Numbers in smart contracts, especially, those being executed in the EVM, are usually stored in variables of the datatype unsigned integer (*uint*). If the stored values exceed the maximum *uint* value (overflow), the value is set to zero. If the value of a *uint* variable becomes smaller than zero (underflow) it is set to its maximum value [61, 63]. Attacks can exploit over- and underflows for different purposes such as manipulation of payout values. To prevent overflow/underflow attacks, developers must consider whether the *uint* value could exceed its maximum or become less than zero.

Unbounded Loops. The most standard form of a gas-focused vulnerability is that of unbounded loops. Loops whose behavior is determined by user input could iterate too many times, exceed the block gas limit, become economically too expensive to perform, or lead to overflow or underflow. For example, a list could become a cause for an unbounded loop if users can add arbitrary entries and, thus, increment the number of iterations necessary to go through the list, where each iteration costs gas. This will commonly lead to a Denial of Service for all transactions that must attempt to iterate the loop [61].

Reentrancy. Atomicity and sequentiality of transaction execution require that non-recursive methods cannot be re-entered before their return values are committed to memory. The requirements for atomicity and sequentiality are by default not fulfilled in smart contracts and must be considered by smart contract developers. Recursive calls of smart contract functions (referred to as reentrancy) can occur when a single smart contract invokes itself or in a chained execution of smart contracts. Often, such recursive calls neglect the execution model underlying smart contracts (e.g., finite state machines), where each change in the smart contract's data represents a transition to a new state of the smart contract. The execution model allows for the execution of

Table 3. Overview of Prevalent DLT Research Streams

Stream	Description	Example
Description	Generation of structured descriptions and classifications of DLT designs	[14, 68]
Analysis	Measurement and report of dependencies between particular DLT characteristics	[11, 52]
Application	Development of prototypes and investigating the application of DLT designs in certain domains	[12, 69]
Guidance	Development of processes to guide practitioners when looking for a suitable DLT design for applications	[68, 70]

functions (e.g., withdraw functions) without changing the smart contracts internal state [63, 64]. In one of the most prominent incidents in the context of smart contracts, the attack on the Decentralized Autonomous Organization (the DAO) [64], reentrancy was exploited, which caused the hard fork of Ethereum into Ethereum Classic and Ethereum in 2016.

Wallet Griefing. A smart contract can cause unexpected errors when invoking external methods that may itself throw an out-of-gas exception [65]. In the EVM, transactions are, for example, issued to an account using the *<recipientAccount>.send(uint)* function. Using this function to transfer tokens can lock the smart contract if error handling is not properly implemented, because the execution of *<recipientAccount>.send(uint)* can produce out-of-gas conditions, where state changes durig the smart contract execution are potentially not completely rolled back [65]. Wallet griefing is also realistic when the smart contract should handle multiple clients without isolation and when a failure in sending transactions using *<recipientAccount>.send(uint)* occurs [61].

2.4 Prior Research on Trade-offs between DLT Characteristics

Viability and maintainability of applications on DLT heavily depends on the choice of a suitable DLT design. Maintainability refers to making an application easy to update and adaptable to changing requirements. Since DLT combines insights from several disciplines of computer science (e.g., distributed systems and cryptography) and economics (e.g., game theory), DLT designs are complex and implications for applications on the respective DLT design are not trivial to derive. Extant research on DLT can be distinguished into four research streams: *description, analysis, application*, and *guidance* (see Table 3).

The *description* research stream focuses on structured descriptions and classifications of DLT; for example, taxonomies of DLT characteristics. Characteristics of and differences between DLT designs are collected and consolidated into structured overviews (e.g., [14, 66–68]). However, dependencies between identified DLT characteristics are seldom investigated and the causes for the ever-increasing number of DLT designs remain unclear. Hence, the practical or technical use of identified DLT characteristics for a comprehensive understanding of functionalities and constraints of DLT designs is limited to the provision of a common understanding of selected DLT designs, while implications for application development remain unclear.

In the second research stream, *analysis*, dependencies between selected DLT characteristics are measured and individual dependencies between DLT characteristics are reported. For example, high performance of a DLT design mostly comes at the cost of its level of security [11, 52]. Extant research explains this trade-off in blockchains by the fact that various attacks result from an increased stale block rate, which is influenced by, among other things, the (mis-)configuration of block size and block creation interval in *public-permissionless* DLT designs [11, 52]. However, the application perspective is not considered in prior analysis research, because most of the research articles do not explain practical implications that result from the observed effects produced by configurations of DLT characteristics. Additionally, only few DLT characteristics have been included in these analyses (e.g., block size, throughput, or scalability). A holistic view on dependencies between DLT characteristics and resulting drawbacks is not presented in this research stream.

The third research stream focuses on the *application* of DLT in certain domains; for example, supply chain management, health IT, or the Internet of Things. Due to the novelty of DLT, the potential for and usefulness of applications on DLT in different domains is still under investigation. Several prototypes of applications on DLT have been developed, which reveal drawbacks of chosen DLT designs for the respective application. For example, the Ethereum blockchain is considered to have a low throughput [12, 69] and is costly [71] when used in the Internet of Things [69]. The Bitcoin blockchain cannot provide confidentiality and has an even lower throughput than Ethereum [72]. IOTA, which is predominantly designed for the use in the Internet of Things, is considered to be slow when handling a massive amount of data [73]. Hyperledger Fabric [29] and Ripple [74] come with high throughput but limited scalability in the number of validating nodes [75]. The practical drawbacks caused by DLT designs are often mentioned in the application research stream, but the causes of the respective drawbacks are not further investigated.

The fourth research stream, *guidance*, focuses on the development of processes to guide practitioners when looking for a suitable DLT design for applications. However, the presented processes are highly abstract and generic, they focus on questions related to whether a distributed ledger is useful at all. Some articles consider select DLT designs and compare them but hardly address causes for the viability of investigated DLT designs for applications (e.g., [17, 76, 77]). Other articles address the degree of decentralization (e.g., [68, 70]). The technical fundamentals of DLT that are crucial for the viability of a DLT design for an application are only sparsely discussed in the guidance research steam. Therefore, existing measures to evaluate suitability of an underlying DLT design for an application cannot be effectively used and the assessment of drawbacks of applications on a particular distributed ledger remains unclear.

These four research streams provide valuable contributions in general and, in particular, for identifying trade-offs. Although some trade-offs have been identified in prior research, these research streams are disjunct, which is why it is hard to obtain a holistic overview of the implications of a DLT design for an application on DLT. More comprehensive analyses of trade-offs in DLT in extant research are limited to the context of electronic health records and consider only blockchains [9, 78]. The findings in extant research on dependencies between DLT characteristics should be synthesized to identify trade-offs and support the development of viable applications on DLT for various use cases. This is the objective of our work.

3 METHOD

We applied a three-step research approach to answer our research question: *How do trade-offs between DLT characteristics impact the viability of applications on DLT?* First, we identified prevalent DLT characteristics by conducting a descriptive literature review [79–81] and surveying DLT experts. Second, we analyzed the identified DLT characteristics in detail to uncover trade-offs in DLT designs. Finally, we consolidated the identified trade-offs into archetypes and derived implications for and drawbacks of applications on DLT.

3.1 Identification of DLT Characteristics

Our descriptive literature review [82] was guided by extant recommendations for literature reviews [83–85]. To identify publications addressing DLT characteristics, we searched scientific databases that cover the top computer science conferences and journals: ACM Digital Library, EBSCOhost, IEEE Xplore, ProQuest, and ScienceDirect. To cover a broad set of publications, we searched each database with the following string in title, abstract, and keywords: *(blockchain* OR ("distributed ledger*"))*. We limited our search to peer-reviewed articles to ensure a high quality of articles. Our search in June 2018 identified 1,144 articles. To identify and filter articles, we first checked the relevance of each article by analyzing title, abstract, and keywords. If any indication for relevance appeared, the article was marked for further analysis. We excluded articles that were duplicates (62), grey literature (i.e., editorials, work-in-progress, dissertations) or books (18), not applicable to our study (56), or not available in English (31). This first relevancy assessment resulted in a sample of 977 potentially relevant articles. Afterwards, a fine-grained relevance validation was made by accessing and reading the article abstracts, resulting in a final sample of 191 relevant articles. In this second relevance assessment, we excluded non-research articles (76) and articles that did not relate to viability of DLT designs for applications on DLT (710).

After the literature search was completed, we carefully read and analyzed the 191 articles to identify DLT characteristics. For each extracted DLT characteristic, we recorded a name, a description, and the original source [86]. In total, 277 DLT characteristics were extracted. A list of master variables was created to aggregate the identified DLT characteristics. A master variable is an aggregation of similar DLT characteristics consisting of a master variable name and a master variable description [86]. If an identified DLT characteristic fits into an existing master variable, we assigned it accordingly; otherwise, a new master variable was created. For example, we aggregated the DLT characteristics *immutability* and *tamper-resistance* to the master variable *integrity*. Since different people often put the same labels on different things and vice versa, we considered semantic ambiguities (e.g., different terms for the same characteristic) during our data analysis [87]. To improve readability of this article, we use the term DLT characteristic for the identified master variables in the remainder of this manuscript, because master variables represent aggregations of similar DLT characteristics. To ensure that we identified a reliable set of master variables, we aimed to reach theoretical saturation [88, 89] with respect to the emerging DLT characteristics. Since no new master variable emerged in the last 27 articles identified in our literature review, we are confident to have reached theoretical saturation.

DLT characteristics are further grouped into DLT properties (e.g., performance or security) under consideration of their influence on the DLT design. For instance, DLT characteristics were grouped into the DLT property *security* if they were related to common security topics such as *confidentiality, integrity*, and *availability*.

To consolidate and critically evaluate the derived DLT characteristics and DLT properties and their respective definitions, we set up an online survey to obtain feedback. We sent 68 requests for feedback via email to DLT experts who had at least three years of experience in dealing with DLT in a business or private context. Thirty-five DLT experts participated in the survey, and we received 113 comments on the generated DLT properties and DLT characteristics. We revised the DLT characteristics and DLT properties and their definitions according to the obtained feedback. For example, the DLT characteristic cost was split into *resource consumption* and *transaction fee*.

3.2 Uncovering Trade-offs between DLT Characteristics

We extracted dependencies between DLT characteristics described in the examined research articles. For research articles discussing relevant trade-offs, we coded trade-offs between DLT characteristics (e.g., [9, 11, 78]). In addition, we analyzed the identified dependencies between DLT characteristics (e.g., more replications of the stored data increase availability) and abstracted trade-offs (e.g., more replications increase the latency until consistency among all nodes is reached).

We evaluated the derived trade-offs on seven DLT designs including Bitcoin, Ethereum, and Hyperledger Fabric representing the blockchain concept; RChain and soteriaDAG representing the BlockDAG concept; and IOTA and Nano representing the TDAG concept. In particular, we discussed the occurrence of the identified trade-offs in intensive group discussions by two authors and two PhD students having profound knowledge and experience in the domain of DLT. Prior to the group discussions, the four participants individually rated each trade-off for the selected ledgers based on their knowledge and experience. In addition, each participant studied available

Table 4. Identified DLT Properties

DLT Property	Description
Flexibility	The degrees of freedom in deploying applications on and customizing a distributed ledger
Opaqueness	The degree to which the use and operation of a distributed ledger cannot be tracked
Performance	The accomplishment of a given task on a distributed ledger under efficient use of computing resources and time
Policy	The ability to guide and verify the correct operation of a distributed ledger
Practicality	The extent to which users of a distributed ledger can achieve their goals with respect to social and socio-technical constraints of everyday practice
Security	The likelihood that functioning of the distributed ledger and stored data will not be compromised

documentation and white papers for the selected DLT designs. Individual rating results were consolidated and then actively discussed by participants until all conflicts were resolved and consensus (total finality!) was reached.

3.3 Configuration of DLT Archetypes

To make the derived trade-offs between DLT characteristics more tangible and evaluate their impact on applications on DLT, we jointly configured DLT archetypes for each DLT property. These archetypes describe how to configure DLT characteristics to achieve a certain DLT property (e.g., flexibility) while considering underlying trade-offs. To identify the archetypes, we reviewed identified DLT characteristics for each DLT property and selected trade-offs corresponding to DLT characteristics first. We then decided what DLT characteristic is preferred over another to achieve the DLT property with respect to each trade-off. For example, an adequate block size outweighs transaction fees (see trade-off G.1) to achieve the DLT property flexibility. Finally, for each archetype and its corresponding DLT property, we highlight drawbacks for applications on DLT that are caused by the underlying DLT design. We assumed that all characteristics that are more positively associated with the DLT property assigned to the archetype should have a high value and accounted for the respective trade-offs. For the *performance archetype*, we assumed, for example, high scalability and throughput and analyzed the effects of this configuration on DLT characteristics of other DLT properties (e.g., *availability* within *security*).

4 TRADE-OFFS BETWEEN DLT CHARACTERISTICS

4.1 DLT Characteristics and Trade-offs between DLT Characteristics

The literature review revealed 40 DLT characteristics that are relevant for the assessment of a DLT design's viability for an application on DLT. The 40 DLT characteristics are briefly presented and defined in Table 5. The grouping of the 40 DLT characteristics resulted in a final set of 6 DLT properties, which are presented in Table 4. In the following, we will discuss derived trade-offs between DLT characteristics. Table 6 lists identified trade-offs.

A Flexibility vs. Performance

A.1 Turing-complete Smart Contracts vs. Resource Consumption

The use of external services (e.g., external data feeds) in smart contracts via oracles enables more flexibility in defining the conditions that must be fulfilled before the smart contract issues transactions. If an external service is requested from a smart contract, the oracle that manages the communication between the smart contract and the external service as well as the external service itself receives requests from every node, because every node needs to execute the smart contract. An oracle and the corresponding external service can become a performance bottleneck, because their bandwidth may not be sufficient to handle the amount of (almost) simultaneous requests by nodes.

Table 5. Identified DLT Characteristics

DLT Property	DLT Characteristic	Description
Flexibility	Interoperability	The ability to interact between distributed ledgers and with other external data services
	Maintainability	The degree of effectiveness and efficiency with which a distributed ledger can be kept operational
	Turing-complete Smart Contracts	The support of Turing-complete smart contracts within a DLT design
	Token Support	The possible uses of tokens within a distributed ledger (e.g., security token, stable coin, or utility token)
	Transaction Payload	The size of the payload in a transaction
Opaqueness	Traceability	The extent to which transaction payloads (e.g., assets) can be traced chronologically in a DLT design
	Transaction Content Visibility	The ability to view the content of a transaction in a DLT design
	User Unidentifiability	The difficulty of mapping senders and recipients in transactions to identities
	Node Controller Verification	The extent to which the identity of validating node controllers is verified prior to joining a distributed ledger
Policy	Auditability	The degree to which an independent third party (e.g., state institution, certification authority) can assess the functionality of a distributed ledger
	Compliance	The alignment of a distributed ledger and its operation with policy requirements (e.g., regulations or industry standards)
	Degree of Decentralization	A distributed ledger's degree of decentralization refers to the number of independent validating node controllers reduced by the number of controllers that control more than average validating nodes divided by the total number of node controllers in the DLT network.
	Incentive Mechanism	A structure in place to motivate node behavior that ensures viable long-term operation of a distributed ledger (e.g., by contributing computational resources)
	Liability	The existence of a natural or legal person that can be subjected to litigation with respect to the distributed ledger
Performance	Block Creation Interval	The time between the creation of consecutive blocks (only in DLT designs using blocks)
	Block Size Limit	The value of a fixed maximum storage size of a block (only in DLT designs using blocks)
	Confirmation Latency	The time span between the inclusion of a transaction in a ledger and the point in time where enough subsequent transactions have been included in the ledger so that the likelihood of future manipulations of the initial transaction becomes negligible
	Resource Consumption	The computational efforts required to operate a distributed ledger (e.g., for transaction validation, block creation, or storing the distributed ledger)
	Propagation Delay	The time between the submission of a transaction (or block) and its propagation to all nodes
	Scalability	The capability of a distributed ledger to efficiently handle decreasing or increasing amounts of required resources (e.g., of transactions per second or number of validating nodes)
	Stale Block Rate	The number of blocks that have been generated in a period of time but not appended to the main chain of the distributed ledger (only in forkable DLT designs using blocks)
	Throughput	The maximum number of transactions that can be appended to a distributed ledger in a given time interval
	Transaction Validation Latency	The time required for validating a transaction by validating nodes

(Continued)

Table 5. Continued

DLT Property	DLT Characteristic	Description
Practicality	Transaction Fee	The price transaction initiators can or must pay for the processing of transactions
	Ease of Node Setup	The ease of configuring and adding a new (or previously crashed) node to the distributed ledger
	Ease of Use	The simplicity of accessing and working with a distributed ledger
	Support for Constrained Devices	The extent to which devices with limited computing capacities (e.g., sensor beacons) can participate in a distributed ledger
Security	Atomicity	The state where transactions are either completely executed or not executed
	Authenticity	The degree to which the correctness of data that is stored on a distributed ledger can be verified
	Availability	The probability that a distributed ledger is operating correctly at any point in time
	Censorship Resistance	The probability that a transaction in a distributed ledger will be intentionally aborted by a third party or processed with malicious modifications
	Confidentiality	The degree to which unauthorized access to data is prevented
	Consistency	The absence of contradictions across the states of the ledger maintained by all nodes participating in the distributed ledger
	Durability	The property that data committed to the distributed ledger will not be lost
	Fault Tolerance	The constant maximum proportion of failed, malicious, or unpredictable nodes a distributed ledger can compensate while operating correctly
	Integrity	The degree to which transactions in the distributed ledger are protected against unauthorized (or unintended) modification or deletion
	Isolation	The property that transactions do not impact each other during their execution
	Non-Repudiation	The difficulty of denying participation in transactions
	Reliability	The ability of a system or component to perform its required functions under stated conditions for a specified time
	Strength of Cryptography	The difficulty of breaking the cryptographic algorithms used in the DLT design

A.2 Turing-complete Smart Contracts vs. Transaction Fee

In Bitcoin, no Turing-complete smart contracts can be developed and time complexity for processing a transaction (e.g., for multi-signature transactions) equals $k * N$ at maximum. Where (a transaction payload incorporates N bytes and a constant factor k). Due to the limited flexibility in Bitcoin smart contracts, there is no need to apply a mechanism to interrupt potential infinite loops (e.g., like *gas* in Ethereum). In Bitcoin, transaction fees are employed to incentivize validating nodes to prefer the validation of a transaction with higher transaction fees over transactions where the sender is only willing to pay a smaller transaction fee.

Turing completeness (e.g., in Ethereum) adds more flexibility to smart contracts but also increases complexity and vulnerabilities. Turing completeness allows for the use of loops in smart contract code, which may even result in infinite loops and, eventually, distributed denial-of-service (DDoS) attacks. The (automated) detection of infinite loops is not possible due to the halting problem [90]. To cope with potential infinite loops in permissioned DLT designs, timeouts are often applied (e.g., in Hyperledger Fabric). Such timeouts, however, limit flexibility in smart contract development, because an upper boundary of time is defined to kill the execution of a smart contract. To overcome this limitation, a pricing schedule is applied in various DLT designs (e.g., *gas* in Ethereum; see Section 2.3) to incentivize validating nodes to execute smart contracts. Users must pay a certain charge for smart contract execution proportional to the smart contract's computational operations. In such distributed ledgers, transaction fees are thus both an incentive mechanism for nodes to process smart contracts and a security mechanism to prevent potential vulnerabilities stemming from Turing-completeness in smart contracts.

A.3 Turing-complete Smart Contracts vs. Transaction Validation Speed

Support for more expressive programming languages (i.e., C++, Java, or Solidity) enables the development of smart contracts that offer a broad range of functionality. The more functionality is added to a smart contract, the higher becomes its complexity. Ultimately, this impedes performance because of the increased execution time for complex smart contracts. Consequently, the time required for transaction processing and validation increases [91].

B Flexibility vs. Security

B.1 Maintainability vs. Availability

To secure DLT designs, the software client of individual nodes must be maintainable and remain compatible with the majority of nodes in the network. Updates of the client protocol of a DLT design must be performed on each node. This is why maintainability of DLT designs decreases with an increasing number of independent nodes due to additional efforts when negotiating and applying software client updates. For example, in Bitcoin and LiteCoin, it has taken weeks to agree on updates such as the adoption of Segregated Witness (SegWit) and SegWit2x [92]. However, an increasing number of nodes increases the ledger's redundancy due to increasing replications. The dependency between maintenance-related cost (e.g., time and money) and the degree of decentralization of the distributed ledger is also derogatorily refered to as blockchain bloat or DLT bloat [93].

B.2 Maintainability vs. Integrity

To allow for efficient maintenance of a distributed ledger, the coordination of update procedures should be facilitated by a low number of (independently controlled) nodes (see trade-off B.1). However, a decrease in the number of independently maintained nodes (hence, a decrease in the DLT design's degree of decentralization) impedes the integrity of DLT designs due to reduced absolute fault tolerance regarding the number of tolerable, malicious nodes.

However, a high level of a distributed ledger's integrity also impacts maintainability of applications on DLT [94]. For achieving a high integrity of distributed ledgers, smart contracts are tamper-resistant: Smart contracts must always be redeployed and initialized with the state of the obsolete version whenever the smart contract should be updated. In addition, the new address of the updated smart contract must be adapted in any module of the corresponding applications and the chained smart contracts that reference the deprecated smart contract. Hence, tamper-resistance and resulting integrity of a distributed ledger increases efforts for maintenance. However, by relaxing integrity, thus, tamper-resistance of smart contracts, the idea of an inevitable and automated enforcement of agreements becomes vulnerable to malicious behavior.

B.3 Turing-complete Smart Contracts vs. Confidentiality

Use of smart contracts threatens confidentiality in three ways. First, it is publicly visible which account's transactions triggered a smart contract [95]. Second, the compiled smart contract code is also visible to the public and smart contracts can be decompiled to human readable source code. Thus, the current state of the smart contract and even values of variables that are declared private in the smart contract can be inferred due to the open smart contract code and transactions [95]. Hence, the common ways of using smart contracts do not support confidentiality. Nevertheless, there are new approaches for private smart contracts. For example, as proposed in the HAWK framework [95], smart contracts can be divided into a private and a public part. The private part determines the payout distribution among involved parties; the input data (e.g., a number of coins) is kept private and is protected using zero-knowledge proofs [95]. As a result, no participant knows the input data other participants sent to the smart contract.[3] Third, oracles and external services might have insight into data that are exchanged via smart contracts. Such oracles or services are often centralized instances that forward certain data, for example, in Provable (formerly known as Oraclize) [97] or TownCrier [85] this is the case. The use of external services in DLT requires at least one trusted party that stores the requested data. Thus, the oracle provider and also the provider of the external data feed can have insights into data flows that are made by users who trigger a smart contract. Although requests may be (partially) executed in a protected Software Guard Extensions (SGX) enclave[4] (e.g., Town Crier [85]), there is at least the risk of a leaked key that can be used to decrypt the respective data and there is a risk of failures in the centralized architecture.

C Opaqueness vs. Performance

C.1 User Unidentifiability vs. Resource Consumption

To achieve unidentifiability among users of a distributed ledger, additional computational resources are required such as computational power, storage space, and runtime [98]. For example, additional data structures can be used to increase unidentifiability but require additional storage size [72, 98, 99] or zero-knowledge proofs, which, for example, require various message exchanges between two parties to validate instead of one [100]. Thus, unidentifiability comes with the cost of high resource consumption.

C.2 User Unidentifiability vs. Throughput

The less a network is controlled by a central authority and the more nodes participate in the network (i.e., given a high degree of decentralization), the more vague is the identity of nodes. Therefore, *public-permissionless* distributed ledgers promise higher user unidentifiability than permissioned ones due to typically a higher number of nodes and a higher degree of decentralization. In contrast, a smaller, *permissioned* distributed ledger with verified and identifiable nodes allows for higher throughput, because faster consensus mechanisms can be used (e.g., PBFT). Nevertheless, unidentifiability can be improved by applying additional processes like mixing and the use of new keypairs for each transaction [101]. Yet, these processes create overhead due to preprocessing of each transaction, which results in extended transaction validation speed and hence decreases throughput.

D Opaqueness vs. Practicality

D.1 Node Controller Verification vs. Ease of Node Setup

Verification of node controllers and their node permissions (e.g., permissions to read data or to validate and commit new transactions) is required in permissioned DLT designs [68]. After permissions are granted to a node, the node can participate in (mostly, voting-based) consensus mechanisms (e.g., PBFT, PoA, or PoET; see Table 1). A public key infrastructure (PKI) with a trusted certification authority is often used to verify the nodes' identities and issue certificates to nodes [102]. However, a PKI produces additional efforts to obtain a certificate for the public-private key pair and leads to the dependency on a trusted certificate authority. Consequently, it becomes more complex to set up a node and participate in a permissioned distributed ledger compared to *public-permissionless* DLT designs (e.g., Bitcoin or Ethereum), which only require to install an open-source software client (e.g., *geth* or *parity* for Ethereum).

[3]The public portion of a HAWK smart contract is composed of three parts: publicly executed code, privately executed code, HAWK manager code. While the publicly executed code is executed on each node of the distributed ledger, the privately executed portion of the smart contract code is only executed by users who sent transactions to the smart contract. The HAWK manager is a trusted party who runs the HAWK manager code in an Intel SGX enclave [96]. Thus, the HAWK manager must be trusted to not disclose private data of a smart contract, which is sent for the execution of the smart contract.

[4]Intel SGX is a set of central processing unit instruction codes that allows user-level code to allocate private regions of memory (referred to as enclaves). These enclaves are protected from processes running at higher privilege levels [96].

E Performance vs. Performance

E.1 Block Creation Interval vs. Stale Block Rate

Forkable and block-based DLT designs (e.g., Bitcoin or Ethereum) enable smaller block creation intervals. With smaller block creation intervals, transactions can be faster appended to the distributed ledger; however, other nodes may not be aware of newly created blocks fast enough and keep working on already deprecated blocks. Ideally, each node would stop working on their blocks as soon as a new block is announced to save computational resources. However, nodes might receive newly created blocks too late, due to propagation latency, and keep working on a stale block. Consequently, smaller block creation intervals increase stale block rate [103] and cause computational inefficiency.

F Performance vs. Policy

F.1 Block Creation Interval vs. Degree of Decentralization

In DLT designs where mining is performed, a long block creation interval decreases the frequency of reward payouts and decreases the likelihood of rewards for individual miners. High variance in payments for miners makes it more likely that mining nodes will join mining pools[5] to increase the probability of receiving rewards [104, 105]. However, the formation of mining pools decreases the degree of decentralization of a distributed ledger due to collusion of miners in a mining pool [105].

F.2 Confirmation Latency vs. Degree of Decentralization

In *public-permissionless* DLT designs, participation of a high number of independent nodes (i.e., a high degree of decentralization) in consensus finding is required to protect the distributed ledger from malicious behavior and other Byzantine failures (see Figure 2). In turn, the number of nodes participating in the consensus mechanism negatively impacts the confirmation latency, because agreement (e.g., *all nodes choose the same block*) and termination (e.g., *all nodes eventually choose a block*) in consensus finding cannot be reached at the same time in asynchronous systems [68, 106]. As a solution, consistency and synchronicity must be relaxed in *public-permissionless* DLT designs to achieve liveness in distributed ledgers with a high degree of decentralization.

F.3 Throughput vs. Degree of Decentralization

One major technological drawback inherent to current *public-permissionless* blockchains (e.g., Bitcoin or Ethereum) is a low throughput [107]. Such DLT designs are run by Thousands of nodes. These nodes are operated by potentially malicious node controllers, which is why various DLT designs apply consensus mechanisms (e.g., Nakamoto consensus) that reach consistency across multiple nodes with a comparably high (Byzantine) fault tolerance. However, most consensus mechanisms that allow for highly decentralized distributed ledgers only provide probabilistic finality to increase throughput by decreasing message complexity (see Table 1) [108]. In contrast, consensus mechanisms with total finality (e.g., PBFT [40] or PoA [109]) can only include a comparatively small set of validating nodes due to their extensive communication overhead [106, 108, 110]. Hence, consensus mechanisms providing fast finality are commonly applied in permissioned DLT designs. Nevertheless, some consensus mechanisms for *public* DLT designs decrease the degree of decentralization to achieve an increase in throughput; for example, GoChain's PoR, which builds upon PoA and allows only selected organizations to run a validating node [35]. Meanwhile, there are consensus mechanisms that apply special derivates of PBFT to *public-permissionless* DLT designs (e.g., Tendermint [30] or EOS [31]), where a set of nodes is randomly chosen to reach consensus. However, this still centralizes decision making to a subset of nodes, decreasing the ledger's degree of decentralization. Thus, increased throughput comes at the cost of the degree of decentralization.

[5] A mining pool is a coalition of miners who share mining rewards if one of these nodes receives the mining reward. Thus, the probability to receive some coins increases for each node.

G Performance vs. Practicality

G.1 Block Size vs. Transaction Fee

Bitcoin has no mandatory transaction fees but allows for optional transaction fees (see trade-off A.2). However, miners could create huge blocks to receive transaction fees from as many transactions as possible, which eventually inhibits the distributed ledger from operating correctly. Generation of such huge blocks is prevented by introducing a maximum block size limit (e.g., in Bitcoin 1 MB per block or in Bitcoin Cash 2 MB per block). Nevertheless, such fixed block size limits reduce flexibility in distributed ledgers, because only limited data can be included in blocks. In contrast, Ethereum has no fixed maximum block size in favor of more flexibility (especially, when using smart contracts). However, Ethereum needed to solve the issue of potentially huge blocks, which is why transaction fees must be paid by any user of the Ethereum network.

H Performance vs. Security

H.1 Confirmation Latency vs. Fault Tolerance

Fault-tolerant consensus mechanisms come with an inherent trade-off between responsiveness and robustness [111]. Although enabling better responsiveness, allowing for forks in DLT reduces robustness of a distributed ledger. To minimize the number of forks and to strengthen security, the targeted block creation interval must be set to a value that is large enough to minimize the stale block rate and small enough to confirm sufficient transactions within a certain period. If the block creation interval is set too short, the number of tolerable, malicious nodes decreases due to too many forks, because new blocks are created before all nodes received the last valid block committed to the ledger; however, new blocks are faster confirmed for some nodes. However, if the block creation interval is set too long, confirmation latency increases, because it takes more time to append a sufficient number of blocks so that it can be assumed that a transaction is committed to the ledger.

To cope with crashed nodes, weak synchronicity [111, 112] is often applied, where the system designer makes timing assumptions on network delays to guarantee that the system will respond within a defined timeframe. A node is assumed as failed if it did not respond within this timeframe (e.g., in PBFT and consensus mechanisms that adapt PBFT) [112]. In DLT designs such as Bitcoin and Ethereum [23], where the number of nodes is unknown, the timing assumption is expressed by the targeted average block creation interval [38], which prevents nodes from working too long on already stale blocks. Due to the assumption of weak synchronicity in the consensus mechanism, the targeted block creation interval strongly depends on the assumed block propagation time [113]. Timing assumptions (or block creation intervals) must be well balanced. If the timing assumption is too short, too many nodes would be considered as failed, which weakens robustness (e.g., fault tolerance) of the underlying security model. If the timing assumption is too long, responsiveness decreases [111].

H.2 Throughput vs. Consistency

For the DLT concept blockchain, it was found that an increased block size can increase throughput, because more transactions can be included in a block [11]. An increased size of data packets (i.e., blocks or transactions) comes with a longer propagation delay [11, 68, 114], which results in a longer state of inconsistency between nodes in a distributed ledger [71, 115]. For Bitcoin and Ethereum, it was found that the percentage of created blocks that are successfully committed to the blockchain's main chain becomes low as the block size (and consequently the block propagation delay) increases [11]. Consequently, the stale block rate increases and nodes have inconsistent views on the ledger until the forks are resolved. Such inconsistent states facilitate successful attacks (see Section 2.2). Forkable DLT designs based on PoW can only improve throughput by degrading consistency, thereby, and increasing vulnerability [116].

In BlockDAGs and TDAGs, throughput and scalability are usually much higher than in blockchains, because the number of transactions per second is not bouned by the block size, the block creation interval (due to relaxed consistency), or the requirement of being eventually added to the main chain. Instead, BlockDAGs and TDAGs (e.g., IOTA, RChain, or soteriaDAG) require blocks or transactions to be linked to previous blocks or transactions so that individual nodes do not have to store an identical version of the ledger. Scalability in terms of increasing or decreasing throughput is theoretically infinite. However, such systems are much more complex than blockchains, because they often aim to be fully asynchronous and the process of converging toward a consistent state among all nodes is mostly non-deterministic (e.g., in IOTA).

H.3 Throughput vs. Fault Tolerance

In blockchains, the requirement for high throughput is predominantly met by applying consensus mechanisms with finality where a small set of nodes participates in the transaction and block validation process (e.g., PBFT [40]). In such consensus mechanisms, the number of nodes n determines the message complexity for the synchronization of the node states, which often equals $O(n^2)$. Due to the exponential increase in message complexity in contrast to probabilistic consensus mechanisms, finality comes at the cost of the degree of decentralization of the DLT design and its fault tolerance. In the case of *public-permissionless* DLT designs for the use of cryptocurrencies, fault tolerance is, for instance, predominantly prioritized above all other DLT properties characteristics such as *performance* and *flexibility*. For example, the Bitcoin blockchain achieves an average throughput of only seven transactions per second and a block takes an average of 10 min to be appended to the ledger [23]. However, Bitcoin is Byzantine fault tolerant up to 50% of fraudulent validating nodes [23]. In contrast, PBFT achieves a moderate throughput of 3K transactions per second but tolerates only: less than one third ($f \leq \frac{|N|-1}{3}$) of fraudulent nodes in a set of validating nodes in the distributed ledger (N) [40].

H.4 Throughput vs. Integrity

Increased block size can increase throughput, because more transactions can be included in a block [11]. Bandwidth [99] and the current size of a block strongly influence the block propagation delay. Thus, the increased throughput comes with longer block propagation delays, because more transactions are included in a block. However, longer block propagation delays increase the probability of forks [68], which threaten integrity and facilitate successful partition-based attacks on the distributed ledger [117] (e.g., selfish-mining [103, 118], long-range attacks [53], bribery attacks [51]; see Section 2.2). To preserve integrity, the block creation interval must be adjusted in concert with the block size because a longer block creation interval mitigates the occurrence of forks and resulting attacks in blockchains [53]. Nevertheless, long block creation intervals also decrease the number of blocks issued, ultimately decreasing throughput. An increase of the block size to include more transactions per block will increase the message propagation delay and, thus, the number of forks in the distributed ledger.

Furthermore, highly varying loads on the distributed ledger caused by variations in transaction frequency result in block size variations and variations in the block propagation delay [119]. Variations in the block propagation delay increase the probability of successful selfish-mining attacks, thereby threatening integrity [16, 52].

I Policy vs. Flexibility

I.1 Degree of Decentralization vs. Maintainability

Distributed ledgers and applications on DLT require efficient maintenance to allow adaption to changing requirements and to increase security of the distributed ledger [120]. *Public-permissionless* DLT designs enable a high degree of decentralization, thereby, supporting unidentifiability, because nodes do not need to be verified before joining the distributed ledger. Anybody is allowed to create new accounts. However, updates of software clients (e.g., geth) for a DLT design resulting from protocol changes must be accepted by the majority of nodes in the whole network to keep compatibility among the nodes after a hard fork and to prevent successful malicious behavior of nodes (see Section 2.1) [121]. The usually large number of nodes in *public-permissionless* distributed ledgers inhibits the introduction of mechanisms enforcing that nodes are kept up to date, which decreases maintainability of a DLT design [68]. Thus, *public-permissioned* DLT designs come with the costs of lower maintainability. In contrast, *private-permissioned* DLT designs are better maintainable, because each node is verified and node controllers can be contacted directly.

J Policy vs. Security

J.1 Degree of Decentralization vs. Integrity

As the network size increases in *public-permissionless* DLT designs, it becomes unlikely that participating node controllers have the same (malicious) intentions or even know each other. Hence, the degree of decentralization increases (see Figure 2) and the presence of a group of nodes with shared interests that takes control of the distributed ledger becomes unlikely (e.g., by gaining a majority of, for example, 51% of the overall hashing power). This strengthens the integrity of the respective DLT design.

In contrast, *private* DLT designs typically incorporate a small number of identifiable (trusted) nodes operated by verified node controllers, thus, decreasing the degree of decentralization and threatening integrity. Each node of such a *private* DLT design has an increased influence in the distributed ledger, which increases vulnerability, for example, with respect to the blockchain anomaly (see Section 2.2) [27]. Thin nodes, which only store parts of the distributed ledger, must assume that validating nodes verify all blocks and follow an effective incentive mechanism when creating blocks. Otherwise, thin nodes risk to accept invalid transactions [122]. End-users who only retrieve data from the distributed ledger and do not verify the distributed ledger's integrity on their own (e.g., using simple payment verification) cannot be sure that the distributed ledger's transaction history has not been tampered with [123].

In *permissioned* DLT designs, where only a subset of nodes is permitted to validate transactions and issue new blocks, the degree of decentralization of a distributed ledger decreases. However, *permissioned* DLT designs (and small *private-permissionless* distributed ledgers) can make use of consensus mechanisms that preserve total finality (e.g., PBFT). After total finality has been reached among the validating nodes, committed transactions cannot be retroactively changed. Hence, the trade-off between degree of decentralization and integrity predominantly refers to DLT designs that make use of probabilistic finality.

K Policy vs. Practicality

K.1 Degree of Decentralization vs. Transaction Fee

The degree of decentralization comes at the cost of higher transaction fees due to the applied consensus mechanism. In DLT designs that employ consensus mechanisms with probabilistic finality and rely on leader election based on PoW (e.g., Nakamoto consensus), the degree of decentralization is important to ensure integrity of the stored data. To achieve a high degree of decentralization, *permissionless* DLT designs are characterized by extreme openness for new nodes. Arbitrary nodes can join the distributed ledger to participate in consensus finding and to validate transactions—without requiring permissions. As computations on blockchains are performed on each node, the total computational effort for the distributed ledger increases with an increasing number of nodes while the average transaction rate is constant. To compensate computational efforts, such DLT designs apply an economic incentive mechanism that rewards nodes for their share of resources [68]. The economic rewards require a pricing structure that usually expects transaction issuers to pay transaction fees for the transaction processing and respective computational efforts.

In contrast, various voting-based consensus mechanisms (e.g., PBFT or EOS's PoS) do not require transaction fees but allow only for a low degree of decentralization, since they are unsuitable

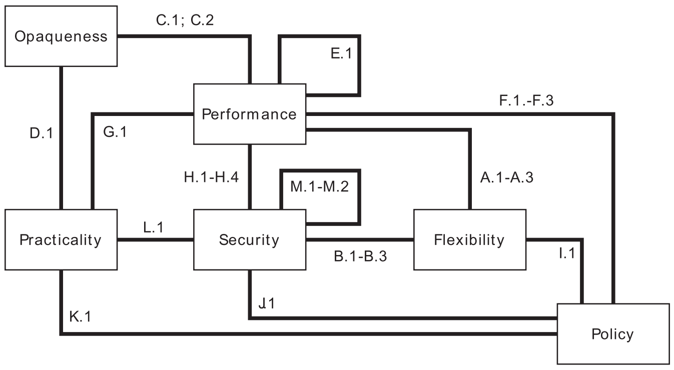

Fig. 3. Identified trade-offs between DLT properties.

for a large number of validating nodes compared to, for example, Nakamoto consensus. The low degree of decentralization results from the fact that the applied consensus mechanisms require each node to agree on a certain state to reach total finality before a new transaction is committed to the distributed ledger. In addition, such finality-preserving consensus mechanisms are usually monetarily less costly than PoW-based consensus mechanisms due to lower consumption of computational resources [106].

L Security vs. Practicality

L.1 Strength of Cryptography vs. Support for Constrained Devices

The strength of cryptography of a DLT design is dependent on the degree of security reached by algorithms for the generation of public-private key pairs (e.g., to secure authentication), for content encryption (e.g., to protect confidentiality), and for hash value calculation (hashing). For public key encryption, it is important that the key pairs are unique and cannot be guessed. The algorithm's time complexity is important for encrypting/signing and decrypting/verifying data. In addition to time complexity of public key encryption, the applied hash algorithm yields a likelihood for collisions [124]. Low collision likelihood is desirable, which is why more secure hashing and key generation approaches are required (e.g., more bits for the hash). However, an increased strength of cryptography requires more computational resources, such as random access memory and storage memory [125]. Thus, constrained devices such as microcontrollers can hardly handle resource-intensive cryptography [125, 126].

M Security vs. Security

M.1 Confidentiality vs. Integrity

To improve confidentiality, DLT designs are often implemented in a private network, where only select nodes can join (i.e., *private* DLT designs); for example, a private Ethereum or Hyperledger Fabric blockchain. However, a small number of known nodes makes it easier to have detailed information on the network topology. Access to a detailed network topology facilitates initiation of targeted delays in the communication between nodes, because the data flow is known [48]. Thus, the probability for successful partition-based attacks [48] increases in private, forkable DLT designs such as a private Ethereum blockchain, which increases the likelihood for violations of a distributed ledger's immutability. Increased vulnerability for immutability violations reduces the integrity of a distributed ledger.

M.2 Consistency vs. Availability

Distributed systems theory reveals a trade-off between consistency and availability—the CAP Theorem [114, 127]. This trade-off also persists in the field of DLT and is caused by latency in block propagation, for example, due to big block sizes or network failures. The larger the number

Table 6. Overview of Identified Trade-offs between DLT Characteristics for the Generated Archetypes and Exemplary DLT Designs

Trade-Offs between DLT Characteristics			**Archetypes**						**Exemplary DLT Designs**						
A	**B**	**ID**	**Flexibility**	**Opaqueness**	**Performance**	**Policy**	**Practicality**	**Security**	**Bitcoin**	**Ethereum**	**Hyperledger Fabric**	**RChain**	**soteriaDAG**	**IOTA**	**Nano**
Block Size (+)	Transaction Fee (-)	*G.1*	A	-	-	-	B	A	B	A	A	AB	B	-	-
Block Creation Interval (+)	Degree of Decentralization (+)	*F.1*	B	B	A	-	A	B	B	B	A	AB	AB	-	-
	Stale Block Rate (-)	*E.1*	-	-	B	-	A	B	B	A	A	A	AB	-	-
Confidentiality (+)	Integrity (+)	*M.1*	B	A	B	B	B	A	B	B	A	AB	B	-	B
Consistency (+)	Availability (+)	*M.2*	B	-	A	A	B	B	B	B	A	B	B	B	B
Confirmation Latency (-)	Fault Tolerance (+)	*H.1*	A	-	A	B	A	B	B	B	A	A	A	A	AB
	Degree of Decentralization (+)	*F.2*	B	B	A	A	A	B	B	B	A	A	A	A	AB
Degree of Decentralization (+)	Integrity (+)	*J.1*	A	A	B	B	B	A	A	A	B	A	A	B	AB
	Maintainability (+)	*I.1*	B	A	B	B	B	B	A	A	B	A	A	A	A
	Transaction Fee (-)	*K.1*	A	A	B	-	B	A	A	A	B	AB	A	B	AB
Strength of Cryptography (+)	Support for Constrained Devices (+)	*L.1*	B	A	-	A*	B	A	A	A	-	A	A	B	B
Maintainability (+)	Availability (+)	*B.1*	A	-	A	A	B	B	B	B	A	B	B	B	B
	Integrity (+)	*B.2*	A	-	A	A	B	B	B	B	A	B	B	X	B
Node Controller Verification (+)	Ease of Node Setup (+)	*D.1*	B	B	A	A	B	A	B	B	A	B	B	B	B
Turing-complete Smart Contracts (+)	Confidentiality (+)	*B.3*	A	B	-	-	A	B	-	A	AB	AB	X	X	X
	Resource Consumption (-)	*A.1*	A	-	B	-	B	-	-	A	AB	AB	B	B	B
	Transaction Fee (-)	*A.2*	A	-	B	-	B	-	B	A	AB	A	B	B	B
	Transaction Validation Speed (+)	*A.3*	A	-	B	-	B	-	B	A	A	A	B	B	B
Throughput (+)	Consistency (+)	*H.2*	A	-	B	B	A	B	B	B	A	AB	AB	A	AB
	Degree of Decentralization (+)	*F.3*	A	B	A	-	A	B	B	B	A	AB	AB	AB	AB
	Fault Tolerance (+)	*H.3*	A	-	A	B	A	B	B	B	A	AB	A	-	AB
	Integrity (+)	*H.4*	A	-	A	B	A	B	B	B	A	AB	AB	A	AB
User Unidentifiability (+)	Resource Consumption (+)	*C.1*	B	A	-	-	B	A	-	-	X	B	B	B	B
	Throughput (+)	*C.2*	B	A	B	B	B	B	A	A	B	B	A	B	B
									Blockchain			BlockDAG		TDAG	

A: *DLT characteristic A outweighs DLT characteristic B*
AB: *DLT characteristic A and B are both achieved (trade-off avoided by other means)*
-: *Trade-off not applicable*
(+): *DLT characteristics is aimed to be high*
B: *DLT characteristic B outweighs DLT characteristic A*
X: *Neither DLT characteristic A nor B are achieved (neither characteristic seems to be a design goal for the ledger)*
*: *only signatures*
(-): *DLT characteristic is aimed to be low*

of nodes that must receive new transactions, the longer the distributed ledger is in an inconsistent state. The larger the number of nodes of a distributed ledger, the more time it takes until each node has received the new block. However, many replications of the data stored on the distributed ledger increases availability. Thus, there is a trade-off between high availability and fast consistency.

4.2 DLT Design Archetypes for Applications on DLT

We introduce six archetypes of DLT designs to illustrate and to consolidate the previously presented trade-offs. Figure 3 illustrates the identified trade-offs on the DLT-property layer. The archetypes indicate benefits and drawbacks for applications on DLT that result from the choice and configuration of a DLT design that is optimized toward a certain DLT property. Table 6 gives an overview of the identified trade-offs between DLT characteristics for the archetypes and exemplary DLT designs.

Flexibility Archetype. The *flexibility archetype* is designed to achieve high degrees of freedom in deploying applications on and customizing a DLT design. The flexibility archetype is predominantly characterized by the following five DLT characteristics: support for *Turing-complete smart contracts*, high *interoperability* with various external systems, a high *degree of maintainability*, a high *degree of decentralization*, and high *throughput*. Turing-complete smart contracts allow for the development of complex applications on DLT. Furthermore, such expressive smart contracts are often required to enable interoperability between distributed ledgers, for example, by using simple payment verification [66]. In addition, the flexibility archetype must be efficiently maintained to allow for fast bug fixes or updates. In turn, efficient maintainability also requires an efficient change management, thus, governance mechanisms. Governance mechanisms pose a challenge in DLT designs with a high degree of decentralization (see trade-off I.1). However, distributed ledgers of the flexibility archetype should be capable of a high degree of decentralization to allow nodes to arbitrarily join and leave the distributed ledger; yet, the flexibility archetype should still achieve high throughput to be applicable in a variety of use cases, which may require high performance (e.g., sensor-based real-time monitoring of a production process).

The use of Turing-complete smart contracts in a distributed ledger with a high degree of decentralization comes at the cost of resource consumption (see trade-off A.1) and a slower transaction validation speed (see trade-off A.3). No fixed block size (or transaction size) should be introduced to allow for the deployment of smart contracts at any size. Accordingly, the flexibility archetype is likely to make use of transaction fees. The introduction of transaction fees also supports a high degree of decentralization, because node controllers will receive rewards for their share of computing resources. The high degree of decentralization should be supported by a seamless ease of node setup. Ledgers of the flexibility archetype neglect integrity in favor of availability (see trade-off B.1) and maintainability (see trade-off B.2), which suggests that *public-permissioned* DLT designs could be employed to support efficient maintainability while ensuring a high availability. In the flexibility archetype, low resource consumption is preferred over user unidentifiability (see trade-off C.1) to allow for the use of devices with constrained computational resources.

The focus on low resource consumption, while supporting Turing-complete smart contracts, suggests a need for thin nodes. Smart contracts should only be executed by full nodes but not by thin nodes. Furthermore, ledgers of the flexibility archetype are likely to apply sharding to achieve parallel execution of smart contracts and to achieve high throughput and a high degree of decentralization. A DLT design that aligns with the flexibility archetype has already been proposed for the *Serenity* update in the Ethereum protocol [128], which will divide the network into three distinct shards: transaction processing shard (*Main Chain*), consensus shard (*Beacon Chain*), and smart contract execution shard (*Sharding Chain*) [128]. Furthermore, DLT designs that strongly relax their consistency assumptions while still supporting Turing-complete smart contracts (e.g., RChain) align well with the flexibility archetype [25]. Applications deployed on ledgers of the flexibility archetype may become expensive to use if most of the logic is performed by smart contracts. In the development of applications on ledgers of the flexibility archetype, a balance must be maintained between the use of smart contracts and traditional programs to express and enforce program logic. A balance between use of smart contracts and traditional programs is also important for maintainability of the respective application. For example, functionalities that are updated frequently should not be stored on the distributed ledger or in a smart contract, because when the smart contract needs to be updated, it is reset and starts from zero. If a smart contract is used, data stored in it (for example, a list of user accounts) should be kept in another smart contract that is only used for data storage and retrieval. By doing so, smart contract functionality can be maintained while still keeping its current state.

Opaqueness Archetype. The *opaqueness archetype* is specialized to prevent use and operation of a DLT design to be tracked. This archetype is concerned with the achievement of a high degree of *user unidentifiability*, high *confidentiality*, a high *degree of decentralization*, and the absence of *node controller verification*. Confidentiality and user unidentifiability are the main requirements to be fulfilled by the opaqueness archetype. A high degree of decentralization is desired to support user unidentifiability. Node controller verification contradicts with the opaqueness archetype. Therefore, more seamless ease of node setup is preferred (see trade-off D.1).

Although ledgers of the opaqueness archetype are geared towards *ease of node setup* instead of *node controller verification*, a high *ease of node setup* and *ease of use*, in general, may not be achieved in ledgers of the opaqueness archetype, because the use of additional anonymization mechanisms is often recommended (e.g., use of the TOR network in Zcash). Such additional anonymization mechanisms may not be easy to comprehend or apply for users and can pose a risk for user anonymity. To achieve user unidentifiability, additional processing of transactions is necessary (e.g., mixing or zero-knowledge proofs). These processes are time-consuming and require additional computational power, which slows down performance and can hardly be performed on constrained devices (e.g., microcontrollers or sensors). A high degree of decentralization increases unidentifiability, but increases confirmation latency, thus, impeding consistency. From a policy point of view, auditability is impaired, because transactions are not traceable and issuers and recipients of a transaction are not easily identifiable. The opaqueness archetype is likely to not offer Turing-complete smart contracts, because smart contracts pose a threat to confidentiality and make it easier to identify transaction senders and receivers and to monitor their interactions (see trade-off B.3) [129]. Applications with a strong requirement for opaqueness should handle most of their advanced business logic off-chain, because the opaqueness archetype will probably provide poor performance and flexibility. Due to the immutability of stored data, there is a threat of revealing encrypted content as technology evolves.

The opaqueness archetype aligns well with *public-permissionless* ledgers where multiple cryptographic techniques are applied (e.g., zero-knowledge proofs) to make it as hard as possible to assign transactions to their senders and receivers or to reveal transaction contents. Popular representatives for the opaqueness archetype are Dash [36], Monero [130], and Zcash [58]. In Dash, additional fees must be paid if a transaction should be issued privately. Dash still allows to view the transaction recipient. In Monero, ring signatures are applied to obfuscate the identity of involved parties [131]. However, Monero has been criticized for vulnerabilities that make transactions eventually traceable [132]. Although Zcash does not obfuscate IP addresses of clients, it is currently considered the most confidentiality-preserving DLT design (especially, when using it over the TOR network).

Performance Archetype. The *performance archetype* is focused accomplishing a given task on a distributed ledger under most efficient use of computing resources and time constraints. The performance archetype is characterized by high *throughput*, low *confirmation latency*, low *resource consumption*, and a high *maintainability*. High throughput and confirmation latency can be achieved by keeping the number of validating nodes small (e.g., by using a *private* DLT design). A small number of validating nodes supports maintainability (see trade-off I.1) and can accelerate consistency among all nodes [114, 127]. When deciding for a high degree of decentralization, consistency assumptions would need to be relaxed to achieve high throughput and scalability (e.g., in IOTA or RChain) [114, 127].

DLT designs that align well with the performance archetype are not likely to support user unidentifiability or Turing-complete smart contracts to decrease the transaction processing time. Due to the short-targeted confirmation latency, the performance archetype will have lower fault tolerance. To accomplish fast confirmation latency with total finality, the performance archetype can be instantiated as a *private*(*-permissioned*) DLT design. Such *private* DLT designs come at the cost of low availability [114, 127], user identifiability (see trade-off C.2), a low degree of censorship resistance, low fault tolerance (see trade-off H.3), and low integrity (see trade-off J.1). In private instantiations of the performance archetype, node controller verification is required, which decreases the ease of node setup (see trade-off D.1). If ledgers of the performance archetype should also scale to a huge number of validating nodes a *public-permissionless* DLT design can be designed based on the blockDAG or the TDAG DLT concept, where consistency assumptions are relaxed compared to blockchains that require a certain average block creation interval for synchronization. Such DLT concepts predominantly levereage probabilistic consensus mechanisms that often have higher fault tolerance, but less integrity.

Multiple DLT designs targeting high performance that follow a *private(-permissioned)* approach in blockchains (e.g., Hyperledger Fabric) or rely on BlockDAGs (e.g., RChain) or TDAGs (e.g., IOTA), have been developed. To increase performance (especially, scalability of blockchains), sharding is applied; that is, multiple distributed ledgers exist in parallel and are connected with each other (e.g., in Zilliqa [133] or Wanchain [134]) [135, 136]. Sharding requires interoperability between the DLT designs, which brings more complexity to the distributed ledger but also better maintainability of the particular distributed ledger. Applications requiring high-performance DLT designs have a limited degree of decentralization or increased complexity due to sharding. However, new consensus mechanisms are under development (e.g., ε-differential agreement) that scale proportional to the number of nodes in the network (e.g., seele [137]).

Policy Archetype. The *policy archetype* aims to offer a variety of abilities to govern and verify the correct operation of a DLT design. Thus, ledgers of the policy archetype are likely to make use of *node controller verification* to better *govern, maintain*, and *audit* the appropriate setup of their nodes. For efficient governance, various mechanisms are provided to users of ledgers of the policy archetype (e.g., standard smart contracts for voting). High maintainability of distributed ledgers in the policy archetype allows for the introduction of updates, which makes the ledgers more flexible and capable to apply changes to the protocol to achieve compliance with targeted regulations or standards. To check compliance, auditability is important in the policy archetype. To audit data in the distributed ledgers, fast *consistency*, high *integrity*, and *non-repudiation* are of particular importance in the policy archetype, just as well as, *transaction content visibility* and *traceability*. Fast consistency among nodes contributes to less contradictions between the statements represented in the data stored on the nodes, which facilitates the auditing process. In addition, *integrity*—in particular, *tamper-resistance*—of once-stored data helps to trace the history of logs (e.g., transfer of assets between users), which increases the reliability of audits. Finally, *non-repudiation* is important to be able to reliably map such logs to users in audits or governance.

The specialization of ledgers regarding the policy archetype predominantly comes at the cost of opaqueness-related DLT characteristics (i.e., *traceability, transaction content visibility*, or *user unidentifiability*) and, additionally, *confidentiality* (due to transaction content visibility) and *throughput* (see trade-off H.2). New regulations and standards are often introduced, and distributed ledgers must adapt to them to achieve compliance. Due to the high targeted level of integrity, the ex post adaptation of a distributed ledger to reach compliance becomes challenging. For example, it is not possible to become compliant with the requirements imposed by the General Data Protection of the European Union (GDPR) [138] when personal data are stored on a distributed ledger because GDPR demands for a possibility to completely delete personally-identifiable user data. To increase flexibility to adapt applications on DLT to future regulations or standards, developers must carefully determine which data should be stored on-chain or off-chain [138, 139]. For now, it remains unclear how to provide flexibility to become compliant with future regulations or standards and achieve a high level of integrity at the same time (e.g., [138]). Therefore, sensitive data should be predominantly stored off-chain. Nevertheless, off-chain data stores are controlled by at least one trusted third party, which lowers the degree of decentralization of applications on DLT. In addition, external data need to be kept confidential and available for the distributed ledger. Thus, reliable interoperability of DLT designs with oracles becomes important for the policy archetype [60]. Furthermore, the oracles themselves must also be compliant with the same laws and regulations.

Due to its strong requirement for transaction content visibility and traceability, instantiations of the policy archetype are likely to be found as a *private* DLT design. All users are identifiable and no unknown user is allowed to view the data stored on the distributed ledger, which increases confidentiality. Furthermore, *private* DLT designs allow for better maintainability (see trade-off I.1) and faster consistency (see trade-off M.2) compared to *public-permissionless* DLT designs.

Practicality Archetype. Ledgers of the *practicality archetype* are designed to allow their users to achieve their goals with respect to social expectations on technology in everyday practice [152]. Thus, DLT designs that align well with the practicality archetype offer high *throughput* and a low *confirmation latency* to achieve low response time (under consideration of transaction finalization), high *support for constrained devices*, and low *transaction fees*. In addition, the practicality archetype also provides Turing-complete smart contracts to allow for interoperability with other distributed ledgers [66] or non-DLT systems [60].

Despite the advantages of the practicality archetype, there are several drawbacks. Various *public-permissionless* DLT designs incentivize nodes to share resources with monetary mechanisms to reach a high *degree of decentralization* (see trade-off K.1). A high *degree of decentralization* and openness for new users of the distributed ledger is also targeted in ledgers of the practicality archetype. To make applications on DLT easily usable by a large number of users that interact with the distributed ledger via a broad variety of devices including constrained devices, such as sensors, a full replication of the ledger on each device should be avoided and ledgers of the practicality archetype should allow for the use of thin nodes. The requirements for high *scalability*, high *throughput*, and *fast confirmation latency* indicate that ledgers of the practicality archetype likely have poor *fault tolerance* (see trade-off H.3) and that the *consistency* assumptions need to be relaxed (see trade-off H.2). Furthermore, the pragmatism comes at the cost of *confidentiality* (see trade-off M.1) and *user unidentifiability* (see trade-offs C.1 and C.2).

Ledgers of the practicality archetype ensure that users do not need to have sound knowledge of DLT before using it, while allowing them to easily interact with the distributed ledger. Therefore, users of ledgers of the practicality archetype will usually not host their own node but will be offered other gateways to interact with the ledger. The ledgers are operated by a consortium as (*private- or public-*)*permissioned* distributed ledgers. Since a private key cannot be recovered in a decentralized infrastructure, users can no longer access their assets if they lose their private key. Therefore, the management of a public-private key pair should be made easy and secure for users, which is why the provision of secure tools for the organization of users' public and private keys is crucial. Exemplary DLT designs that aligns with the practicality archetype are Hyperledger Fabric and EOS (see Table 1).

Security Archetype. The *security archetype* aims to ensure a high likelihood that the functioning of the distributed ledger and stored data will not be compromised. To achieve this goal, DLT designs are optimized toward high *availability*, high *fault tolerance*, high *integrity*, and high *confidentiality*, which may even include *user unidentifiability* to inhibit the mapping of data to identities. To achieve strong integrity and fault tolerance, the *degree of decentralization* should be high.

High availability can be achieved by adding numerous, physically distributed nodes to the distributed ledger, each maintaining a replication of the ledger. While large network size is comparably easy to achieve, achieving a high degree of decentralization is more challenging. However, the degree of decentralization is a focal requirement in the security archetype (e.g., trade-offs *B.1* or *J.1*). The degree of decentralization does not merely result from the DLT protocol. Instead, it predominantly depends on socio-technical phenomena, such as (ad-hoc) consortia of validating node controllers (e.g., mining pools; see Figure 2). Avoiding such consortia poses a particular challenge in the instantiation of the security archetype. Due to its high requirements for confidentiality, the security archetype is likely to require additional techniques that make the identification of users difficult (see opaqueness archetype). Such mechanisms come at the cost of increased resource consumption and less support for constrained devices (see trade-off C.1). The use of anonymization techniques (e.g., zero-knowledge proofs) may cause serious security issues, because it is hard to audit the distributed ledger due to decreased traceability (e.g., by applying mixing) and decreased transaction content visibility (e.g., by encryption).

Due to trade-offs inherent to the security archetype (e.g., trade-offs M.1 and M.2), DLT designs that correspond to the security archetype are likely to either achieve *security through decentralization* or *security through permission*. The first aims to solicit a huge number of independent node controllers, which increases absolute fault tolerance (tolerable number of malicious nodes). The most prominent DLT design that follows the approach of *security through decentralization* is the Bitcoin blockchain [23]. The Bitcoin blockchain is highly available due to its high number of nodes; and it hardly exposes potential for flawed smart contracts. Nevertheless, Bitcoin does not fulfill all the security-related DLT characteristics (e.g., confidentiality). To increase confidentiality, Zcash applied zero-knowledge-succinct-non-interactive-arguments-of-knowledge (zk-SNARKs) to obfuscate transaction senders (and receivers) and to impede transaction traceability. However, the added complexity of zk-SNARK has already caused a counterfeiting vulnerability for Zcash coins [140], which indicates that such techniques also allow for new vulnerabilities. *Security through permission* aims at limiting access to the distributed ledger to known users, which increases maintainability of the distributed ledger at the cost of its degree of decentralization (see trade-off I.1).

5 DISCUSSION

5.1 Principal Findings

Our research reveals 24 trade-offs (see Table 6) based on 40 identified DLT characteristics (see Table 5), which we grouped into 6 DLT properties (see Table 4). The diversity of the identified DLT characteristics from purely technical (e.g., *strength of cryptogra*phy in *security*) to social (e.g., *degree of decentralizati*on in *policy*) highlights the complexity of DLT. Among the abstracted trade-offs, the DLT properties *performance* and *security* each exhibit the most trade-offs (11) (see Figure 3 and Table 6). Purely performance- or security-oriented DLT designs appear to be challenging and may even be impossible to be developed due to trade-offs between DLT characteristics within the respective DLT property (see *performance* and *security archetype*).

The consolidation of DLT deigns based on the identified trade-offs between DLT characteristics into archetypes of DLT designs elucidates that it is not possible to develop a one-size-fits-all DLT design that fulfills all requirements of each application. Thus, application designers will have to wisely choose a DLT design when aiming to develop viable applications on DLT. Nevertheless, the derived archetypes partially support each other (e.g., opaqueness and security), while others contradict (e.g., *performance* vs. *security*). This phenomenon can also be seen as an indicator of compatibility between DLT designs. The preference of one DLT characteristic over another in a trade-off is critical. For example, the security archetype can benefit from certain features of DLT designs leveraging the opaqueness archetype; these DLT designs could even benefit from each other if they were combined. To jointly use DLT designs that have made contradicting decisions in the trade-offs, interoperability between DLT designs is required, because otherwise they could not synchronize.

Several identified trade-offs are inherent to distributed systems; for example, those related to the CAP theorem [10, 114] (see trade-off M.2) or the FLP impossibility [141] (see trade-off H.1). However, the implications of these trade-offs on DLT designs differ from commonly used distributed databases, where mostly a known number of nodes is employed, and consensus mechanisms are predominantly crash fault-tolerant but not Byzantine fault–tolerant (e.g., Paxos [142] or Raft [143]). In DLT, implications of such design decisions do not only impede consistency of the distributed ledger but may have serious (financial) consequences that result from inconsistencies (e.g., form forks) and successful attacks that make use of inconsistencies to weaken a distributed ledger's integrity (e.g., double spending).

Extant literature shows a trend preferring *private* DLT designs over *public* DLT designs for industrial applications. The shift from the incipient idea of pure decentralization through a high degree of openness (e.g., Bitcoin and Ethereum) to more centralization seems largely motivated by anticipated improvements in performance due to the employment of faster consensus mechanisms and enhanced confidentiality due to restricted access to the ledger. There is much criticism on this shift, specifically, because it contradicts with DLT's original philosophy. Similarly, this shift aligns with our observation that most trade-offs between DLT characteristics are related to the DLT characteristics *security* or *performance. Private* DLT designs come with various advantages compared to *public* DLT designs with respect to practicality but tend to reach a low degree of decentralization compared to public DLT designs, they basically ignore the foundation of DLT [23]. Furthermore, *private* DLT designs are dependent on *interoperability* with other DLT designs or external services to prevent being caught on a blockchain silo or a "*blockchain island*" [144–147]. Furthermore, there are various DLT designs of the DLT concepts BlockDAG and TDAG that incorporate rather loose structures based on DAGs (e.g., IOTA, Nano, or RChain). These DLT designs relax consistency assumptions in favor of high throughput and consensus algorithms that consume less resources compared to established blockchains based on Bitcoin or Ethereum. Nevertheless, support of Turing-complete smart contracts is not yet widespread in extant DLT designs, which limits flexibility of several DAGs.

DLT designs such as Nano and RChain advance the use of DAGs in DLT by relaxing the concept of replicated state machines in favor of less storage consumption. In Nano, only personal transactions are, for example, stored on the terminal device. In RChain, DLT is strongly connected with peer-to-peer file sharing, which is also targeted in other projects such as Ethereum's Swarm. The integration of such external data allows for the extensive use of (unreliable) data sources and may announce the birth of a new generation of peer-to-peer-governed information infrasturucures, where DLT might take an important position by allowing for asset exchanges and tamper-resistant proofs.

5.2 Main Lessons to Be Learned

Our work provides diverse contributions to research and practice. Regarding the latter, practitioners can obtain deep insights into viability of DLT designs for applications on DLT and their possible impacts on organizations. Our work supports the decision making for selecting a DLT design and its later configuration to use for applications on DLT, under consideration of application requirements and DLT characteristics. The overview of DLT characteristics and DLT properties supports practitioners in defining requirements for DLT designs that must be considered in the requirements engineering process to ensure viability of applications on DLT. The derived trade-offs between DLT characteristics and the generated archetypes are useful to be aware of potential benefits and drawbacks for applications on DLT, which can be assessed before starting to develop the application (see Table 6). Such assessments eventually facilitate avoidance of unsuitable DLT designs and consequent waste of resources. To understand causes of such drawbacks for applications on DLT, the described trade-offs between DLT characteristics provide explanations for the dependencies between DLT characteristics (see Section 4.1). Careful DLT design selection and application development becomes crucial to ensure that DLT's unique advantages can be achieved, ultimately pushing DLT from a hype to a critical information infrastructure [148] for future businesses and societies.

Our synthesis of the four prevalent research streams on DLT (*description, analysis, application*, and *guidance*) bridges the dsiparate research on DLT, thereby, contributing to a more holistic view of DLT. The *description* research stream contributes to this work by outlining (currentlly perceived) application domains for DLT. Our classification of DLT characteristics can be used to generate a common understanding of important terms in the field of DLT and their technical dependencies across research fields, such as, economics or computer science. Our findings support the development of comprehensive models and simulations of DLT designs [149, 150]. The results of such *analyses* (e.g., formalization of dependencies between DLT characteristics) will elucidate assessments of the influence of the identified trade-offs between DLT characteristics on applications on DLT. Research on the application of DLT is supported as this work forms a foundation for decision making for a suitable DLT design for a certain application. Within the application research stream, for example, research on business process innovation with DLT can draw on the trade-offs and archetypes to discuss possible negative effects of using certain DLT designs or DLT in general. Furthermore, we contribute to research on software engineering and requirements engineering in distributed systems since a holistic view on non-functional requirements can be obtained from the presented DLT characteristics and their dependencies. Finally, we support the DLT research stream *guidance* by introducing the archetypes of DLT design. The archetypes of DLT design form a fundament for a preselection of DLT designs for applications and can support the selection of an appropriate DLT design (e.g., [70]) to make the selection of a DLT design more efficient.

5.3 Limitations

Nevertheless, our study comes with limitations. DLT characteristics and DLT properties were identified in a literature review in the field of DLT. Analyzed DLT concepts are limited to already published scientific articles and mainly focused on blockchain. We limit our overview of DLT characteristics to those of particular interest in extant research on DLT for the development of applications. The DLT characteristics and related trade-offs are also corroborated by multiple whitepapers of DLT designs such as Bitcoin [31], Ethereum [35], or soteriaDAG [26] (see Table 6). Most of the analyzed research articles in the *application* research stream developed applications on Bitcoin, Ethereum, or Hyperledger Fabric. This makes our work—the trade-offs, in particular—only partially generalizable to other DLT designs. We tried to overcome this limitation by including feedback from several DLT experts in a survey and illustrate the generalizability of the identified trade-offs between DLT characteristics by applying them to DLT designs of all three DLT concepts known so far. There are preliminary approaches for the analysis and formalization of DLT concepts for the development of frameworks for the simulation of DAGs (e.g., [150]). However, we could not identify trade-offs between DLT characteristics that are specific for BlockDAGs and TDAGs based on the reviewed literature. While we analyzed dependencies between DLT characteristics, we predominantly focused on potential negative effects and resulting trade-offs. We acknowledge that dependencies might also lead to synergistic and positive effects.

5.4 Future Research

We identified multiple influential conditions that impact the strength of certain dependencies or even the presence of trade-offs between DLT characteristics such as the applied consensus mechanism or the use of additional services, such as mixing (e.g., with respect to the trade-off *unidentifiability* vs. *throughput* (see trade-off C.1). Thus, researchers of the *analysis* research stream should conduct measurements to quantify the identified trade-offs between DLT characteristics. The analysis should include DLT concepts beyond blockchain to reveal dependencies between DLT characteristics for different DLT concepts. Such analyses would support quantification of the dependencies between DLT characteristics, which would be very useful to quantify the influence of the trade-offs on the affected DLT characteristics. Quantified trade-offs would support the development of decision support systems for the selection of DLT designs for applications in the *guidance* research stream. Based on a quantified model of the trade-offs, monitoring-systems for distributed ledgers can be developed, which can use the trade-offs to predict the behavior of a distributed ledger. Researchers of *applications* on DLT can further investigate how to design decision-support and monitoring applications for DLT.

As the identified archetypes inhibit simultaneous optimization of all DLT characteristics at the same time due to DLT-inherent trade-offs, interoperability between DLT designs (cross-ledger technology) turns out an important avenue for future research in the field of DLT to overcome prevalent issues in DLT (e.g., scalability, throughput, or lack of smart contracts [66, 151]). Research on cross-ledger technology is still in its infancy and is, for example, concerned with the transfer of assets from one distributed ledger to another [144]. Cross-ledger technology can increase flexibility of DLT designs and might help to mitigate the inherent trade-offs through multi-chain networks, which leverage the benefits of one DLT design while avoiding the drawbacks of others through clever cross-chain technology.

5.5 Conclusion

There cannot be a one-size-fits-all DLT design due to dependencies and consequent trade-offs between DLT characteristics. Since it is at least difficult if not complex to consider all the trade-offs and their particular impact at once, this manuscript introduces archetypes of DLT designs and illuminates 24 prevalent trade-offs in DLT designs. The archetypes of DLT design support practitioners in understanding causes of benefits and drawbacks for particular applications on DLT. The trade-offs and their consolidation into archetypes make the challenges inherent in the configuration of a DLT design more transparent for developers and are useful to prevent wrong decisions before choosing a DLT design. Beyond blockchain, our survey article suggests that the true potential of DLT might lie in decentralization of applications that are not as restrictive as Bitcoin transactions while still empowering the individual (e.g., as intended in the Solid project; see https://solid.mit.edu/). Public-permissioned DLT designs appear paricularly prominsing for the future decentraliized internet. We wrote this survey article in the hope that it will be helpful to successfully navigate this future transformation.

ACKNOWLEDGMENTS

We thank all participants of the empirical studies that contributed to this work. In particular, we would like to thank Mikael Beyene and Konstantin Pandl for taking the time to discuss the trade-offs based on the archetypes and selected DLT designs.

REFERENCES

[1] Steven R. Kursh and Natalia A. Gold. 2016. Adding fintech and blockchain to your curriculum. *Bus. Educ. Innov. J.* 8, 2 (2016), 6–12.

[2] Feng Tian. 2016. An agri-food supply chain traceability system for China based on RFID & blockchain technology. In *Proceedings of the 13th International Conference on Service Systems and Service Management.* 1–6.

[3] Filip Caron. 2018. The evolving payments landscape: Technological innovation in payment systems. *IT Profess.* 20, 2 (2018), 53–61.

[4] Gaby G. Dagher, Jordan Mohler, Matea Milojkovic, and Praneeth Babu Marella. 2018. Ancile: Privacy-preserving framework for access control and interoperability of electronic health records using blockchain technology. *Sustain. Cities Soc.* 39 (2018), 283–297.

[5] Tomi Lehikoinen. 2018. Food supply chain this summer, fishing in Finland means food traceability on the menu. *IBM Blockchain Blog.* 4 (2019). Retrieved from https://www.ibm.com/blogs/blockchain/2018/07/this-summer-fishing-in-finland-means-food-traceability-on-the-menu.

[6] Xueping Liang, Sachin Shetty, Deepak Tosh, Charles Kamhoua, Kevin Kwiat, and Laurent Njilla. 2017. ProvChain: A blockchain-based data provenance architecture in cloud environment with enhanced privacy and availability. In *Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid'17).* 468–477.

[7] Ingo Weber, Vincent Gramoli, Alex Ponomarev, Mark Staples, Ralph Holz, An Binh Tran, and Paul Rimba. 2017. On availability for blockchain-based systems. In *Proceedings of the IEEE 36th Symposium on Reliable Distributed Systems.* 64–73. Retrieved from http://ieeexplore.ieee.org/document/8069069/.

[8] Igor Zikratov, Alexander Kuzmin, Vladislav Akimenko, Viktor Niculichev, and Lucas Yalansky. 2017. Ensuring data integrity using blockchain technology. In *Proceedings of the 20th Conference of Open Innovations Association.* 534–539.

[9] Niclas Kannengießer, Sebastian Lins, Tobias Dehling, and Ali Sunyaev. 2019. What does not fit can be made to fit! Trade-offs in distributed ledger technology designs. In *Proceedings of the 52nd Hawaii International Conference on System Sciences.*

[10] Eric Alan Brewer. 2000. Towards robust distributed systems (abstract). In *Proceedings of the 19th ACM Symposium on Principles of Distributed Computing (PODC'00).* 7.

[11] Johannes Göbel and Anthony E. Krzesinski. 2017. Increased block size and bitcoin blockchain dynamics. In *Proceedings of the 27th International Telecommunication Networks and Applications Conference.* 1–6.

[12] Nejc Zupan, Kaiwen Zhang, and Hans-Arno Jacobsen. 2017. Hyperpubsub: A decentralized, permissioned, publish/subscribe service using blockchains: demo. In *Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference: Posters and Demos.* 15–16.

[13] Jude Nelson, Muneeb Ali, Ryan Shea, and Michael J. Freedman. 2016. Extending existing blockchains with virtualchain. Retrieved from https://www.zurich.ibm.com/dccl/papers/nelson_dccl.pdf.

[14] Florian Glaser and Luis Bezzenberger. 2015. Beyond cryptocurrencies—A taxonomy of decentralized consensus systems. In *Proceedings of the 23rd European Conference on Information Systems.* 1–18.

[15] Karl Wüst and Arthur Gervais. 2017. Do you need a blockchain? Retrieved from https://eprint.iacr.org/2017/375.pdf.

[16] Ittay Eyal and Emin Gün Sirer. 2014. Majority is not enough: Bitcoin mining is vulnerable. In *Financial Cryptography and Data Security*, Nicolas Christin and Reihaneh Safavi-Naini (eds.). Retrieved from http://arxiv.org/abs/1311.0243.

[17] Qassim Nasir, Ilham A. Qasse, Manar Abu Talib, and Ali Bou Nassif. 2018. Performance analysis of hyperledger fabric platforms. *Secur. Commun. Netw. (Sept. 2018).* 1–14.

[18] J. P. Morgan Chase & Co. 2018. J. P. Morgan interbank information networksm expands to more than 75 banks. Retrieved from https://web.archive.org/save/https://www.jpmorgan.com/country/US/en/detail/1320570135560.

[19] Kaiwen Zhang and Hans-Arno Jacobsen. 2018. Towards dependable, scalable, and pervasive distributed ledgers with blockchains. In *Proceedings of the IEEE 38th International Conference on Distributed Computing Systems.* 1337–1346.

[20] Leslie Lamport, Robert Shostak, and Marshall Pease. 1982. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.* 4, 3 (1982), 382–401.

[21] Ali Sunyaev. 2019. Distributed ledger technology. In *Internet Computing: Principles of Distributed Systems and Emerging Internet-based Technologies* (1st ed.). Springer, 265–292.

[22] Vitalik Buterin. 2018. Ethereum whitepaper. Retrieved from https://github.com/ethereum/wiki/wiki/White-Paper/f18902f4e7fb21dc92b37e8a0963eec4b3f4793a.

[23] Satoshi Nakamoto. 2008. Bitcoin: A peer-to-peer electronic cash system. Retrieved from https://bitcointalk.org/bitcoin.pdf.

[24] Garrick Hileman and Michel Rauchs. 2017. Global Blockchain Benchmarking Study (September 22, 2017). Available at SSRN: https://ssrn.com/abstract=3040224 or http://dx.doi.org/10.2139/ssrn.3040224.

[25] Ede Eykholt, Lucius Gregory Meredith, and Joseph Denman. 2017. RChain architecture documentation - release 0.8.1. Retrieved from https://buildmedia.readthedocs.org/media/pdf/rchain-architecture/stable/rchain-architecture.pdf.

[26] Soteria Lab. 2019. soteriaDAG - Soteria DAG Project. Github. Retrieved from https://github.com/soteria-dag/soterd/blob/master/docs/README.md.

[27] Christopher Natoli and Vincent Gramoli. 2016. The blockchain anomaly. In *Proceedings of the IEEE 15th International Symposium on Network Computing and Applications.* 310–317.

[28] GoChain Foundation. 2019. Official GoChain documentation. Retrieved from https://web.archive.org/web/20190516083054/https://github.com/gochain-io/docs.

[29] Hyperledger. 2017. Hyperledger-fabricdocs documentation release v0.6. Retrieved from https://buildmedia.readthedocs.org/media/pdf/hyperledger-fabric/v0.6/hyperledger-fabric.pdf.

[30] All In Bits, Inc. 2019. Tendermint documentation. Retrieved from https://tendermint.com/docs/introduction/.

[31] EOS.IO. 2018. EOS.IO Technical White Paper v2. Retrieved from https://github.com/EOSIO/Documentation/blob/master/TechnicalWhitePaper.md.

[32] Colin LeMahieu. 2018. Nano: A feeless distributed cryptocurrency network. Retrieved from https://nano.org/en/whitepaper.

[33] Cody Born. 2018. Ethereum proof-of-authority on Azure. *Microsoft Azure.* Retrieved from https://web.archive.org/web/20190501134839/https://azure.microsoft.com/en-us/blog/ethereum-proof-of-authority-on-azure/.

[34] Intel Corporation. 2015. PoET 1.0 Specification. Retrieved from https://sawtooth.hyperledger.org/docs/core/releases/1.0/architecture/poet.html.

[35] GoChain. 2018. Proof of reputation. Retrieved from https://medium.com/gochain/proof-of-reputation-e37432420712.

[36] Evan Duffield and Daniel Dia. 2019. Dash: A privacy-centric cryptocurrency. Retrieved from https://whitepaperdatabase.com/wp-content/uploads/2017/09/Dash-Whitepaper.pdf.

[37] Serguei Popov. 2018. The tangle. Retrieved from https://assets.ctfassets.net/r1dr6vzfxhev/2t4uxvsIqk0EUau6g2sw0g/45eae33637ca92f85dd9f4a3a218e1ec/iota1_4_3.pdf.

[38] Rafael Pass and Elaine Shi. 2017. The sleepy model of consensus. In *Proceedings of the Advances in Cryptology Conference (ASIACRYPT'17).* 380–409.

[39] Kenji Saito and Yamada Hairoyuki. 2016. What's so different about blockchain? — Blockchain is a probabilistic state machine. In *Proceedings of the IEEE 36th International Conference on Distributed Computing Systems Workshops.* 168–175.

[40] Miguel Castro and Barbara Liskov. 1999. Practical byzantine fault tolerance. In *Proceedings of the 3rd Symposium on Operating Systems Design and Implementation (OSDI'99).* 173–186.

[41] Demiro Massessi. 2018. *Public vs private blockchain in a Nutshell.* Retrieved from https://medium.com/coinmonks/public-vs-private-blockchain-in-a-nutshell-c9fe284fa39f.

[42] Paige Cabianca. 2018. What's the difference between public, private, and permissioned blockchains? Retrieved from https://medium.com/nakamo-to/whats-the-difference-between-a-public-and-a-private-blockchain-c08d6d1886a0.

[43] Zibin Zheng, Shaoan Xie, Hong Ning Dai, Xiangping Chen, and Huaimin Wang. 2018. Blockchain challenges and opportunities: a survey. *Int. J. Web Grid Serv.* 14, 4 (2018), 352.

[44] J. P. Morgan Chase & Co. 2016. Quorum Whitepaper. Retrieved from https://github.com/jpmorganchase/quorum-docs/blob/master/Quorum%20Whitepaper%20v0.1.pdf.

[45] ARK.io. 2019. ARK Ecosystem Whitepaper. Retrieved from https://ark.io/Whitepaper.pdf.

[46] Tom Rodgers. 2019. Ethereum classic price roaring just weeks after 51% attack. Retrieved from https://www.forbes.com/sites/tomrodgers1/2019/04/08/ethereum-classic-price-roaring-just-weeks-after-51-attack/#2906e2a6f7ef.

[47] Markus Jakobsson and Ari Juels. 1999. Proofs of work and bread pudding protocols. In *Proceedings of the IFIP TC6/TC11 Joint Working Conference on Secure Information Networks: Communications and Multimedia Security (CMS'99).* 258–272.

[48] Christopher Natoli and Vincent Gramoli. 2017. The balance attack or why forkable blockchains are ill-suited for consortium. In *Proceedings of the 47th IEEE/IFIP International Conference on Dependable Systems and Networks.* 579–590.

[49] Ethan Heilman, Alison Kendler, Aviv Zohar, and Sharon Goldberg. 2015. Eclipse attacks on bitcoin's peer-to-peer network. In *Proceedings of the 24th USENIX Conference on Security (SEC'15).* 129–144.

[50] Maria Apostolaki, Aviv Zohar, and Laurent Vanbever. 2017. Hijacking bitcoin: Routing attacks on cryptocurrencies. In *Proceedings of the IEEE Symposium on Security and Privacy.* 375–392.

[51] Joseph Bonneau. 2016. Why buy when you can rent? Bribery attacks on bitcoin-style consensus. In *Proceedings of the International Conference on Financial Cryptography and Data Security.* 19–26.

[52] Johannes Göbel, Paul Keeler, Anthony E. Krzesinski, and Peter G. Taylor. 2016. Bitcoin blockchain dynamics: The selfish-mine strategy in the presence of propagation delay. *Perform. Eval.* 104 (2016), 23–41.

[53] Evangelos Deirmentzoglou, Georgios Papakyriakopoulos, and Constantinos Patsakis. 2019. A survey on long-range attacks for proof of stake protocols. *IEEE Access* 7 (2019), 28712–28725.

[54] John R. Douceur. 2002. The Sybil attack. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS'01).* 251–260.

[55] Pim Otte, Martijn de Vos, and Johan Pouwelse. 2017. TrustChain: A Sybil-resistant scalable blockchain. *Fut. Gen. Comput. Syst.* (2017). https://www.sciencedirect.com/science/article/abs/pii/S0167739X17318988?via%3Dihub.

[56] Brian Neil Levine, Clay Shields, and N. Boris Margolin. 2005. A survey of solutions to the Sybil attack. Retreived on April 05, 2019 from https://allquantor.at/blockchainbib/pdf/levine2006survey.pdf.

[57] Patrick Dai, Neil Mahi, Jordan Earls, and Alex Norta. 2017. Smart-contract value-transfer protocol on a distributed mobile application platform. Retrieved from https://web.archive.org/web/20190506095324/https://qtum.org/user/pages/01.home/Qtum%20whitepaper_en%20v0.7.pdf.

[58] Daira Hopwood, Sean Bowe, Taylor Hornby, and Nathan Wilcox. 2018. Zcash protocol specification. Retrieved from https://whitepaperdatabase.com/wp-content/uploads/2018/03/z-cash-zec-whitepaper.pdf.

[59] Howard Shrobe, David L. Shrier, and Alex Pentland. 2018. Enigma: Decentralized computation platform with guaranteed privacy. In *New Solutions for Cybersecurity. Chapter* 15, 504.

[60] Jonathan Heiss, Jan Eberhardt, and Stefan Tai. 2019. From oracles to trustworthy data on-chaining systems. Retrieved from https://www.redaktion.tu-berlin.de/fileadmin/fg308/publications/2019/Heiss-et-al-oracles_preprint.pdf.

[61] Neville Grech, Michael Kong, Anton Jurisevic, Lexi Brent, Bernhard Scholz, and Yannis Smaragdakis. 2018. MadMax: Surviving out-of-gas conditions in ethereum smart contracts. *Proc. ACM Program. Lang.* 2 (Oct. 2018), 1–27.

[62] Maximilian Wöhrer and Uwe Zdun. 2018. Smart contracts: Security patterns in the ethereum ecosystem and solidity. In *Proceedings of the International Workshop on Blockchain Oriented Software Engineering*. 2–8.

[63] Nicola Atzei, Massimo Bartoletti, and Tiziana Cimoli. 2016. A survey of attacks on ethereum smart contracts. Retrieved from https://allquantor.at/blockchainbib/pdf/atzei2016survey.pdf.

[64] Xiangfu Zhao, Zhongyu Chen, Xin Chen, Yanxia Wang, and Changbing Tang. 2017. The DAO attack paradoxes in propositional logic. In *Proceedings of the 4th International Conference on Systems and Informatics*. 1743–1746. Retrieved from http://ieeexplore.ieee.org/document/8248566/.

[65] Peter Vessenes. 2016. Ethereum griefing wallets: Send w/Throw Is Dangerous. Retrieved from https://vessenes.com/ethereum-griefing-wallets-send-w-throw-considered-harmful/.

[66] Niclas Kannengießer, Michelle Pfister, Malte Greulich, Sebastian Lins, and Ali Sunyaev. 2020. Bridges between islands: Cross-chain technology for distributed ledger technology. In *Proceedings of the 53rd Hawaii International Conference on System Sciences*.

[67] Paolo Tasca and Claudio Tessone. 2019. A taxonomy of blockchain technologies: Principles of identification and classification. *Ledger* 4 (2019).

[68] Xiwei Xu, Ingo Weber, Mark Staples, Liming Zhu, Jan Bosch, Len Bass, Cesare Pautasso, and Paul Rimba. 2017. A taxonomy of blockchain-based systems for architecture design. In *Proceedings of the IEEE International Conference on Software Architecture*. 243–252. Retrieved from http://ieeexplore.ieee.org/document/7930224/.

[69] Seyoung Huh, Sangrae Cho, and Soohyung Kim. 2017. Managing IoT devices using blockchain platform. In *Proceedings of the 19th International Conference on Advanced Communication Technology*. 464–467.

[70] Morgen E. Peck. 2017. Blockchain world—do you need a blockchain? This chart will tell you if the technology can solve your problem. *IEEE Spectrum* 54, 10 (2017), 38–60.

[71] Andreas Unterweger, Fabian Knirsch, Christoph Leixnering, and Dominik Engel. 2018. Lessons learned from implementing a privacy-preserving smart contract in ethereum. In *Proceedings of the 9th IFIP International Conference on New Technologies, Mobility and Security*. 1–5.

[72] Merve Can Kus Khalilov and Albert Levi. 2018. A survey on anonymity and privacy in bitcoin-like digital cash systems. *IEEE Commun. Surv. Tutor.* 20, 3 (2018), 1–44. https://ieeexplore.ieee.org/abstract/document/8325269

[73] Bogdan Cristian Florea. 2018. Blockchain and internet of things data provider for smart applications. In *Proceedings of the 7th Mediterranean Conference on Embedded Computing*. 1–4.

[74] Ripple. 2017. Solution overview. Retrieved from https://whitepaperdatabase.com/wp-content/uploads/2017/09/Ripple-XRP-Whitepaper.pdf.

[75] Runchao Han, Vincent Gramoli, and Xiwei Xu. 2018. Evaluating blockchains for IoT. In *Proceedings of the 9th IFIP International Conference on New Technologies, Mobility and Security*. 1–5.

[76] Tien Tuan Anh Dinh, Ji Wang, Gang Chen, Rui Liu, Beng Chin Ooi, and Kian-Lee Tan. 2017. Blockbench: A framework for analyzing private blockchains. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD'17)*. 1085–1100.

[77] Florian Gräbe, Niclas Kannengießer, Sebastian Lins, and Ali Sunyaev. 2020. Do not be fooled: Towards a holistic comparison of distributed ledger technology designs. In *Proceedings of the 53th Hawaii Internantional Conference on System Sciences*.

[78] Odhran O'Donoghue, Anuraag A. Vazirani, David Brindley, and Edward Meinert. 2019. Design choices and trade-offs in health care blockchain implementations: Systematic review. *J. Med. Internet Res.* 21, 5 (2019), e12426.

[79] Jan vom Brocke, Alexander Simons, Kai Riemer, Bjoern Niehaves, and Ralf Platfaut. 2015. Standing on the shoulders of giants: Challenges and recommendations of literature search in information systems research. *Commun. AIS* 37, 9 (2015), 205–224.

[80] Barbara Kitchenham, O. Pearl Brereton, David Budgen, Mark Turner, John Bailey, and Stephen Linkman. 2009. Systematic literature reviews in software engineering. *Inf. Softw. Technol.* 51, 1 (2009), 7–15.

[81] Guy Paré, Marie Claude Trudel, Mirou Jaana, and Spyros Kitsiou. 2015. Synthesizing information systems knowledge: A typology of literature reviews. *Inf. Manag.* 52, 2 (2015), 183–199.

[82] Jane Webster and Richard T. Watson. 2002. Analyzing the past to prepare for the future. *MIS Quart.* 26, 2 (2002), xiii–xxiii.

[83] Gavin Wood. 2016. Polkadot. Retrieved from https://icowhitepapers.co/wp-content/uploads/PolkaDot-Whitepaper.pdf.

[84] Shilan Yang, Huaimin Wang, Wei Li, Wei Liu, and Xiang Fu. 2018. CVEM: A cross-chain value exchange mechanism. In *Proceedings of the International Conference on Cloud Computing and Internet of Things*. 80–85.

[85] Fan Zhang, Ethan Cecchetti, Kyle Croman, Ari Juels, and Elaine Shi. 2016. Town crier: An authenticated data feed for smart contracts. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS'16)*. 270–282.

[86] Mary C. Lacity, Shaji Khan, Aihua Yan, and Leslie P. Willcocks. 2010. A review of the IT outsourcing empirical literature and future research directions. *J. Inf. Technol.* 25, 4 (2010), 395–433.

[87] Mildred L. G. Shaw and Brian R. Gaines. 1989. Comparing conceptual structures: Consensus, conflict, correspondence, and contrast. *Knowl. Acquis.* 1, 4 (1989), 341–363.

[88] Juliet M. Corbin and Anselm L. Strauss. 2015. *Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory* (4th ed.). SAGE Publications.

[89] Barney G. Glaser and Anselm L. Strauss. 2009. *The Discovery of Grounded Theory: Strategies for Qualitative Research* (4th ed.). Routledge.

[90] Martin Davis. 1982. *Computability & Unsolvability*. Dover.

[91] Mingjun Dai, Shengli Zhang, Hu Wang, and Shi Jin. 2018. A low storage room requirement framework for distributed ledger in blockchain. *IEEE Access* 6 (2018), 22970–22975.

[92] NewsBTC. 2018. Increased SegWit Adoption for Bitcoin—Is Lightning Network Next? Retrieved from https://web.archive.org/web/20190410173449/https://www.newsbtc.com/2018/03/01/segwit-adoption-lightning-network-increases-bitcoin/.

[93] Deepak Puthal, Nisha Malik, Saraju P. Mohanty, Elias Kougianos, and Chi Yang. 2018. The blockchain as a decentralized security framework (future directions). *IEEE Consum. Electron. Mag.* 7, 2 (2018), 18–21.

[94] Michael Coblenz. 2017. Obsidian: A safer blockchain programming language. In *Proceedings of the 39th International Conference on Software Engineering Companion (ICSE-C'17)*. 97–99.

[95] Ahmed Kosba, Andrew Miller, Elaine Shi, Zikai Wen, and Charalampos Papamanthou. 2016. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In *Proceedings of the IEEE Symposium on Security and Privacy*. 839–858.

[96] Ittai Anati, Shay Gueron, Simon P. Johnson, and Vincent R. Scarlata. 2014. Innovative technology for CPU Based attestation and sealing. In *Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy*.

[97] Oraclize. 2019. Oraclize documentary. Retrieved from https://provable.xyz/.

[98] Xiaoqi Li, Peng Jiang, Ting Chen, Xiapu Luo, and Qiaoyan Wen. 2017. A survey on the security of blockchain systems. *Fut. Gen. Comput. Syst.* 107 (2017), 1–13. https://www.sciencedirect.com/science/article/abs/pii/S0167739X17318332.

[99] Matevž Pustišek and Andrej Kos. 2018. Approaches to front-end IoT application development for the ethereum blockchain. *Proc. Comput. Sci.* 129 (2018), 410–419.

[100] Ioannis Chatzigiannakis, Apostolos Pyrgelis, Paul G. Spirakis, and Yannis C. Stamatiou. 2011. Elliptic curve based zero knowledge proofs and their applicability on resource constrained devices. In *Proceedings of the IEEE 8th International Conference on Mobile Ad-Hoc and Sensor Systems*. 715–720.

[101] Jan Henrik Ziegeldorf, Fred Grossmann, Martin Henze, Nicolas Inden, and Klaus Wehrle. 2015. CoinParty: Secure multi-party mixing of bitcoins. In *Proceedings of the 5th ACM Conference on Data and Application Security and Privacy (CODASPY'15)*. 75–86.

[102] Micha R. Hoffman. 2018. Can blockchains and linked data advance taxation. In *Proceedings of the Web Conference (WWW'18)*. 1179–1182.

[103] Arthur Gervais, Ghassan O. Karame, Karl Wüst, Vasileios Glykantzis, Hubert Ritzdorf, and Srdjan Capkun. 2016. On the security and performance of proof of work blockchains. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS'16)*. 3–16.

[104] Jega Anish Dev. 2014. Bitcoin mining acceleration and performance quantification. In *Proceedings of the IEEE 27th Canadian Conference on Electrical and Computer Engineering*. 1–6.

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

[106] Du Mingxiao, Ma Xiaofeng, Zhe Zhe, Wang Xiangwei, and Chen Qijun. 2017. A review on consensus algorithm of blockchain. In *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics*. 2567–2572.

[107] Karim Jabbar and Pernille Bjørn. 2018. Infrastructural Grind: Introducing blockchain technology in the shipping domain. In *Proceedings of the ACM Conference on Supporting Groupwork (GROUP'18)*. 297–308.

[108] Tim Swanson. 2015. Consensus-as-a-service: A brief report on the emergence of permissioned, distributed ledger systems. Retrieved from https://www.ofnumbers.com/wp-content/uploads/2015/04/Permissioned-distributed-ledgers.pdf.

[109] Stefano De Angelis, Leonardo Aniello, Roberto Baldoni, Federico Lombardi, Andrea Margheri, and Vladimiro Sassone. 2018. PBFT vs proof-of-authority: Applying the CAP theorem to permissioned blockchain. Retrieved from https://eprints.soton.ac.uk/415083/.

[110] Harish Sukhwani, José M. Martínez, Xiaolin Chang, Kishor S. Trivedi, and Andy Rindos. 2017. Performance modeling of PBFT consensus process for permissioned blockchain network (hyperledger fabric). In *Proceedings of the IEEE 36th Symposium on Reliable Distributed Systems*. 253–255.

[111] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. 2016. The honey badger of BFT protocols. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS'16)*. 31–42.

[112] Dvora Dolev and H. R. Strong. 1982. Distributed commit with bounded waiting. Retrieved from https://www.cs.huji.ac.il/~dolev/pubs/dist-commit.pdf.

[113] Rafael Pass and Elaine Shi. 2017. Rethinking large-scale consensus. In *Proceedings of the IEEE 30th Computer Security Foundations Symposium*. 115–129.

[114] Seth Gilbert and Nancy Lynch. 2002. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *ACM SIGACT News* 33, 2 (2002), 51.

[115] Christian Decker and Roger Wattenhofer. 2013. Information propagation in the bitcoin network. In *Proceedings of the IEEE International Conference on Peer-to-Peer Computing (P2P'13)*. 1–10.

[116] Artem Barger, Yacov Manevich, Benjamin Mandler, Vita Bortnikov, Gennady Laventman, and Gregory Chockler. 2017. Scalable communication middleware for permissioned distributed ledgers. In *Proceedings of the 10th ACM International Systems and Storage Conference (SYSTOR'17)*. 23:1.

[117] Frank Hofmann, Simone Wurster, Eyal Ron, and Moritz Böhmecke-Schwafert. 2017. The immutability concept of blockchains and benefits of early standardization. In *ITU Kaleidoscope: Challenges for a Data-Driven Society*. IEEE, 1–8.

[118] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. 2015. The bitcoin backbone protocol: Analysis and applications. In *Proceedings of the International Conference on the Theory and Application of Cryptographic Techniques (EUROCRYPT'15)*. 281–310.

[119] ISO. 1989. Information processing systems — Open Systems Interconnection — Basic Reference Model — Part 2: Security Architecture. Retreived on June 10, 2018.

[120] Heiko Koziolek. 2011. Sustainability evaluation of software architectures: A systematic review. In *Proceedings of the Joint ACM SIGSOFT Conference - QoSA and ACM SIGSOFT Symposium - ISARCS on Quality of Software Architectures - QoSA and Architecting Critical Systems– (QoSA-ISARCS'11)*. 3.

[121] Jesse Yli-Huumo, Deokyoon Ko, Sujin Choi, Sooyong Park, and Kari Smolander. 2016. Where is current research on blockchain technology? *PLOS One* 11, 10 (2016), 1–27.

[122] Alin Tomescu and Srinivas Devadas. 2017. Catena: Efficient non-equivocation via bitcoin. In *Proceedings of the IEEE Symposium on Security and Privacy*. 393–409.

[123] Nitesh Emmadi and Harika Narumanchi. 2017. Reinforcing immutability of permissioned blockchains with keyless signatures' infrastructure. In *Proceedings of the 18th International Conference on Distributed Computing and Networking (ICDCN'17)*. 1–6.

[124] Adam Back. 1997. A partial hash collision based postage scheme. Retrieved from http://www.hashcash.org/papers/announce.txt.

[125] Lukas Malina, Jan Hajny, Radek Fujdiak, and Jiri Hosek. 2016. On perspective of security and privacy-preserving solutions in the internet of things. *Comput. Netw.* 102 (2016), 83–95.

[126] Djamel Eddine Kouicem, Abdelmadjid Bouabdallah, and Hicham Lakhlef. 2018. Internet of things security: A top-down survey. *Comput. Netw.* 141 (2018), 199–221.

[127] Daniel Abadi. 2012. Consistency tradeoffs in modern distributed database system design: CAP is only part of the story. *Computer* 45, 2 (2012), 37–42.

[128] The Cryptocurrency Consultant. 2019. Ethereum 2.0 - Consensys Publishes Roadmap to Serenity. *Medium*. Retrieved December 10, 2019 from https://medium.com/altcoin-magazine/ethereum-2-0-consensys-publishes-roadmap-to-serenity-e1ce76fa34f2.

[129] Qi Feng, Debiao He, Sherali Zeadally, Muhammad Khurram Khan, and Neeraj Kumar. 2019. A survey on privacy protection in blockchain system. *J. Netw. Comput. Applic.* 126 (2019), 45–58.

[130] Nicolas van Saberhagen. 2013. CryptoNote v 2.0. Retrieved from https://cryptonote.org/whitepaper.pdf.

[131] Nicolas van Saberhagen. 2013. CryptoNote v 2.0. Retrieved from https://whitepaperdatabase.com/wp-content/uploads/2017/09/Monero-whitepaper.pdf.

[132] Malte Möser, Kyle Soska, Ethan Heilman, Kevin Lee, Henry Heffan, Shashvat Srivastava, Kyle Hogan, Jason Hennessey, Andrew Miller, Arvind Narayanan, and Nicolas Christin. 2018. An empirical analysis of traceability in the monero blockchain. *Proc. Privacy Enhanc. Technol.* 2018, 3 (2018).

[133] The Zilliqa Team. 2017. The Zilliqa technical whitepaper. Retrieved from https://docs.zilliqa.com/whitepaper.pdf.

[134] Jack Lu, Boris Yang, Zane Liang, Ying Zhang, Demmon Shi, Eric Swartz, and Lizzie Lu. 2017. Wanchain. Retrieved from https://wanchain.org/files/Wanchain-Whitepaper-EN-version.pdf.

[135] Tien Tuan Anh Dinh, Rui Liu, Meihui Zhang, Gang Chen, and Beng Chin Ooi. 2018. Untangling blockchain: A data processing view of blockchain systems. *IEEE Trans. Knowl. Data Eng.* 30, 7 (2018), 1–20.

[136] Loi Luu, Viswesh Narayanan, Chaodong Zheng, Kunal Baweja, Seth Gilbert, and Prateek Saxena. 2016. A secure sharding protocol for open blockchains. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*. 17–30.

[137] Seele. 2018. Seele whitepaper. Retrieved from http://seele.hk.ufileos.com/Seele_White_Paper_English_v3.1.pdf.

[138] Cindy Compert, Maurizio Luinetti, and Bertrand Portier. 2018. Blockchain and GDPR—How blockchain could address five areas associated with GDPR compliance. Retrieved on December 11, 2018 from https://www.ibm.com/downloads/cas/2EXR2XYP.

[139] Ashiq Anjum, Manu Sporny, and Alan Sill. 2017. Blockchain standards for compliance and trust. *IEEE Cloud Comput.* 4, 4 (2017), 84–90.

[140] Josh Swihart, Benjamin Winston, and Sean Bowe. 2019. Zcash counterfeiting vulnerability successfully remediated. Retrieved from https://electriccoin.co/blog/zcash-counterfeiting-vulnerability-successfully-remediated/.

[141] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. 1985. Impossibility of distributed consensus with one faulty process. *J. ACM* 32, 2 (1985), 374–382.

[142] L. Lamport and M. Massa. 2004. Cheap paxos. In *Proceedings of the International Conference on Dependable Systems and Networks*. 307–314.

[143] Diego Ongaro and John Ousterhout. 2016. In search of an understandable consensus algorithm (extended version). Retrieved from https://raft.github.io/raft.pdf.

[144] Vitalik Buterin. 2016. Chain interoperability [white paper]. Retrieved from http://www.r3cev.com/s/Chain-Interoperability-8g6f.pdf.

[145] Hai Jin, Jiang Xiao, and Xiaohai Dai (Eds.). 2018. Towards a novel architecture for enabling interoperability amongst multiple blockchains. In *Proceedings of the IEEE 38th International Conference on Distributed Computing Systems*.

[146] Claudio Lima. 2018. Developing open and interoperable DLT/blockchain standards. *Computer* 51, 11 (2018), 106–111.

[147] Shijun Liu, Bedir Tekinerdogan, Mikio Aoyama, Liang-Jie Zhang, Liping Deng, Huan Chen, and Jing Zeng (Eds.). 2018. Research on cross-chain technology based on sidechain and hash-locking. Retrieved from https://link.springer.com/content/pdf/10.1007%2F978-3-319-94340-4.pdf.

[148] Tobias Dehling, Sebastian Lins, and Ali Sunyaev. 2019. Security of critical information infrastructures. In *Information Technology for Peace and Security: IT Applications and Infrastructures in Conflicts, Crises, War, and Peace*, Christian Reuter (Ed.). Springer, 319–339.

[149] Diego Marmsoler and Leo Eichhorn. 2018. Simulation-based analysis of blockchain architectures. Retrieved from http://rgdoi.net/10.13140/RG.2.2.19898.44481.

[150] Manuel Zander, Tom Waite, and Dominik Harz. 2018. DAGsim: Simulation of DAG-based distributed ledger protocols. *ACM SIGMETRICS Perform. Eval. Rev.* 46, 3 (2018), 118–121.

[151] Xiwei Xu, Cesare Pautasso, Liming Zhu, Vincent Gramoli, Alexander Ponomarev, An Binh Tran, and Shiping Chen Chen. 2016. The blockchain as a software connector. In *Proceedings of the 13th Working IEEE/IFIP Conference on Software Architecture*. 182–191.

[152] Youngjin Yoo. 2010. Computing in everyday life: A call for research on experiential computing. *MIS Quarterly* 34, 2 (2010), 213–231.

Received June 2019; revised December 2019; accepted January 2020