Consensus in Enterprise and Financial Blockchains: Assumptions and Challenges

Henry F. Korth
Dept. of Computer Science and Eng.
Lehigh University
Bethlehem, PA 18015 USA
hfk2@lehigh.edu

ABSTRACT

While blockchain critics often point to the excessive energy consumption of Bitcoin-style proof-of-work, the rapidly growing world of enterprise-class blockchain databases has a set of assumptions and requirements that differs greatly from those of a public, fully decentralized blockchain like Bitcoin. In such settings, message-based consensus is feasible due to the ability of a permissioned chain to have admission control for nodes and possibly to make stronger assumptions about node behavior. Traditional Byzantine consensus [21] does not fully meet the needs of these environments due to performance issues as well as due to the dynamic arrival and departure of nodes. Practical Byzantine Fault Tolerance [9] addresses the expected-case issue for these environments, though the worst-case performance is bounded by the classic formal results of [12]. The much more recent Stellar protocol [17] addresses the dynamic arrival and departure of nodes via a clever definition of quora for consensus. The Algorand agreement protocol [14] addresses performance even in public chains. Another set of issues arise in cross-chain and off-chain transactions. This talk will review the assumption set driving the design of blockchain-consensus algorithms as they are or are likely to be applied in financial systems and as part of larger enterprise information systems.

PVLDB Reference Format:
Henry F. Korth. Consensus in Enterprise and Financial Blockchains: Assumptions and Challenges. *PVLDB*, 12(xxx): xxxx-yyyy, 2019.
DOI: https://doi.org/10.14778/xxxxxxx.xxxxxxx

1. INTRODUCTION

The subject of distributed consensus is a long-studied problem, going back to the classic two-phase commit (2PC) protocol [16]. Subsequent work sought to provide improved solutions, by seeking either to improve performance or to overcome shortcomings of prior work. Three-phase commit (3PC) [28] addressed the possibility of long-duration blocking in the event of an untimely coordinator crash in 2PC. The distributed commit problem was addressed in a more general distributed-consensus setting in Paxos [15] and subsequent improvements such as Raft [20]. Byzantine consensus [21] considered a weaker assumption regarding node failure. Rather than using the non-malicious *fail-stop* assumption of 2PC and 3PC, Byzantine consensus is based on the assumption that failure may consist of multiple malicious nodes conspiring to subvert the consensus protocol. However, classic work on Byzantine agreement assumes a fixed number n of nodes in the system.

Public blockchain systems, in which new nodes may join without limitation, need a Byzantine-style failure assumption, but face the additional threat of a *Sybil attack*, in which an adversary creates an arbitrarily large number of new nodes in the system for the purpose of subverting the consensus protocol. This threat led the developer(s) of Bitcoin (under the pseudonym Satoshi Nakomoto) to use a *proof-of-work* consensus protocol [19] that requires a would-be Sybil attacker to control more than half of the entire network's computation power (referred to as "51% attack"). That protocol, referred to as *Nakomoto consensus*, is effective from a security standpoint but extraordinarily consumptive of energy.[1]

These and various other consensus protocols present a range of options for an enterprise seeking to deploy a blockchain system. Such enterprises face a choice of using an existing blockchain, cloning an existing open-source blockchain system, or building a new system from scratch. Many factors go into such a choice, with the consensus protocol being only one. This paper's focus, however, is on consensus protocols and their underlying assumptions, and their impact on enterprise-blockchain design choice. For a deeper discussion of issues beyond consensus, see [4, 26].

We begin by reviewing criteria for choosing an approach, starting with the question that should come first in any planning for an enterprise blockchain, "Why not just use a relational database?". After identifying needs that suggest a blockchain approach, we identify criteria for choosing a specific blockchain solution. Next, we attempt to categorize blockchain systems by their approach to consensus, first with a taxonomy in Section 3, and then by reviewing a representative set of systems in Section 4. Because enterprise blockchains do not exist in isolation, we consider consensus across blockchains and between blockchains and

Proceedings of the VLDB Endowment, Vol. 12, No. xxx
ISSN 2150-8097.
DOI: https://doi.org/10.14778/xxxxxxx.xxxxxxx

[1]Most recent estimates show Bitcoin consuming about 0.29% of the world's electricity, roughly that of Switzerland or 5.9 million average U.S. households [digiconomist.net, June 2019].

"off-chain" systems in Section 5. Section 6 discusses choice of an approach from an enterprise perspective.

2. CRITERIA FOR CHOOSING A CONSENSUS PROTOCOL

A growing number of businesses are implementing blockchain databases or at least considering doing so. Quite often these are *distributed ledgers* in which a group of organizations jointly maintain a record of financial transactions. More generally, a group of organizations may seek joint maintenance of a shared, distributed database. In any such case, one must first address the obvious alternative of a traditional distributed database along with its centrally managed consensus protocol (Section 2.1). Once the choice between a blockchain system and a traditional system is made, one can them move on to the choice of a specific approach (Section 2.2). Finally, the performance of the specific implementation needs to be considered. Benchmarking of blockchain systems on realistic benchmarks is considered in [11].

2.1 Blockchain versus Traditional Database

Traditional database systems are well-established, offer high functionality and performance, and likely are already installed in the enterprise. For all participants to share in a singly-administered database based on a traditional database system, all must trust the organization administering the database and furthermore be confident that this trust will persist indefinitely into the future. Blockchain systems, though much newer and in-flux, offer the following four main properties not provided by traditional database systems:

1. **Decentralization:** Limited or no central control. Unlike a distributed system, in which multiple nodes cooperate under a unified administrative framework, a decentralized system offers greater (or total) autonomy to nodes.
2. **Tamper Resistance:** It is infeasible for any participant to change committed data. Note that this is not the same as *durability* in the ACID properties of database transactions, since in a traditional database there is unconditional trust of the database system itself and its administration. In a blockchain, there is as assumed mutual *dis*trust among the participants.
3. **Irrefutability:** Participants' updates are cryptographically signed, allowing all participants to verify the source of data and transactions.
4. **Anonymity:** Participants' identifiers on the blockchain may be kept separate from real-world identifiers, though there remains the threat of de-anonymization using off-chain data.

The key motivation for choosing a blockchain database over a traditional database solution is the degree to which the participating organizations

- need to access data, especially to perform updates, without having to submit requests to a centrally controlled system;
- need a substantial degree of autonomy from centralized control;
- lack complete mutual trust and thus would benefit from a trustless or reduced-trust blockchain environment;
- seek to limit or eliminate the role of a central organization's control.

It is not sufficient to consider these issues just at the present time but also how they may change as the relationships among the organizations evolve. At some point in the future, might an organization have a self-interest in removing or altering updates it had added previously?[2] Is there a need for external data visibility, perhaps for audit purposes? These, and other considerations, if answered in the affirmative, point towards the desirability of choosing a blockchain database of some type. The specific choice then depends on the degree to which each of the four basic blockchain properties must be supported.

2.2 Choosing a Blockchain Solution

Assuming that there is at least a modest degree of current or potential mutual distrust among the participating organizations, a blockchain-based solution may be deemed desirable. The following sections review some of the feature- and policy-based considerations in choosing a blockchain-based approach.

2.2.1 Governance

The first issue is the appropriate governance model for the blockchain. Despite some level of distrust, there may be agreement to have an organization (or a small number of organizations) serve as a *permissioning authority* to control membership in the blockchain system. For example, consider a supply-chain database for a large enterprise. Members of the supply chain may not all trust each other. Furthermore, they may not trust the large enterprise they are supplying not to modify data in the future. Despite such mistrust, the large enterprise does indeed control who its suppliers are, and, as a consequence, it may be a reasonable compromise of autonomy to allow that large enterprise to control membership in the blockchain system. Allowing such a compromise allows greater options in the choice of a consensus protocol because the permissioning authority can protect against Sybil attacks, thus relieving the consensus protocol of that responsibility.

One could choose a totally trustless governance model such as is used by public blockchains. Running on top of a public blockchain creates exposure to volatility of the underlying cryptocurrency as well as contention for commitment of transactions, but provides the start-up and operational savings of not having to run the underlying blockchain. The popularity of ERC-20 (and similar) tokens on top of the Ethereum blockchain is evidence of the perceived merits of this approach. Alternatively, one could create a private clone of a public open-source system. This latter choice might seem like a good compromise, but given the likely relatively modest number of participants, the potential of a 51% attack is high enough as to be unacceptable unless there is a relatively strong degree of trust; and if there is

[2]Consider, for example, a supply chain in which a calamitous recall situation occurs. Contributors to the supply chain for the recalled product suddenly have a strong incentive to cover up responsibility.

such trust, a permissioned solution likely will prove to be more cost-effective and offer a higher level of performance.

2.2.2 *Privacy*

Once the trust/governance model is determined, the next issue is to identify the level of data privacy needed. Certain data, such as transaction IDs and cryptographic signatures, must be accessible by all participants, but the transaction payload may have restricted access. Continuing the supply-chain example, a supplier may wish to keep the price of the supplied items secret between itself and the enterprise controlling the blockchain (and possibly also an auditor). Protection of secrets using public-key cryptography and/or zero-knowledge proofs is discussed elsewhere and is outside the scope of this paper's discussion, as it is not materially influenced by the consensus protocol in place.

2.2.3 *Performance*

Performance is influenced to a significant degree by the choice of consensus protocol. As is the case for any transaction system, performance can be measured by:

- **throughput:** measured in transactions per second;
- **average latency:** measured by average time for a transaction of standard size to be considered *final*. Finality in a blockchain system differs slightly from commit in a traditional database system, as we have discussed;
- **tail latency and latency bounds:** measured by high-percentile latency, with worst-case latency particularly relevant to systems with hard real-time deadlines.

2.2.4 *Data Integration / Cross-system Transactions*

A complete system design must address additionally the matter of data integration between the legacy enterprise database and the blockchain database. Many of these issues have been studied in other contexts of federated databases, but we shall discuss issues specific to cross-system transactions later, in Section 5.

3. CONSENSUS PROTOCOL TAXONOMY

In this section, we list a set of design choices for consensus protocols and discuss their relative advantages and disadvantages. Our focus will be on those protocols most relevant to an enterprise setting and thus on message-based protocols. Then, in the subsequent section, we shall place several existing consensus protocols within this taxonomy.

3.1 Consensus and Coordinator Type

At the highest level, consensus protocols fall into one of 3 categories:

- **[CC1]:** Proof of Work (PoW)
- **[CC2]:** Proof of Stake (PoS)
- **[CC3]:** Message-based

PoW is typified by the energy-consumptive Nakomoto-consensus protocol of Bitcoin. However, alternative PoW approaches have been proposed that benefit from large main memories. An attack is still of prohibitive cost, but the energy costs of a large main memory is much less that that of computation-intensive mining.

PoS assigns a probability of being chosen to add the next block to the chain based on the amount of currency held. The exact meaning of *held* varies. It may be total held, total placed in escrow, or a product of time and holdings. To avoid centralization, a probabilistic choice is typically made using a PoW scheme among the top stakeholders so as to avoid the largest stakeholder having absolute conrol. The relatively small number of nodes involved in this PoW scheme allows for a much less energy-intensive process as compared with a pure PoW solution.

Message-based consensus allows a specific number of nodes to reach agreement despite the presence of some bounded number of nodes being "faulty" or "adversarial." As a result, most message-based protocols rely either on some externally-sourced concept of trust (at least to some limited extent) among certain nodes or on some membership-control authority. Algorand avoids this by a stake-based weighting of node votes in the messaging protocol. Some message-based protocols utilize a central coordinator, possibly with a means to replace that coordinator in the event of actual or suspected failure. This leads us to split CC3 into subcategories as follows:

- **[CC3.1]:** fully decentralized, either no coordinator or all nodes may be coordinator.
- **[CC3.2]:** decentralized with constraints based on trust and or quora.
- **[CC3.3]:** elected and replaceable coordinator.
- **[CC3.4]:** irreplaceable coordinator chosen for each transaction.
- **[CC3.5]:** single, irreplaceable system-wide coordinator.

This subdivision of message-based consensus focuses largely on the degree of centralization of consensus coordination. Greater centralization generally means reduced overhead as measured by message rounds or total number of messages. Centralization comes at a cost in terms of trust (about which we say more in the next section) and possibly in terms of blocking (as in the case of an untimely coordinator failure in 2PC).

3.2 Trust and Membership

Blockchains may have no membership restrictions, as is the case for public chains for which anyone can download the code and become a node. Others are strictly controlled:

- **[TM1]:** Public, open to all without restriction and with no trust requirements.
- **[TM2]:** Public partnership, all partners participate in consensus, but consensus requires some level of mutual trust among subsets of nodes.
- **[TM3]:** Private, a single owner or small group control membership and manage consensus. The nodes corresponding to the controlling group must be trusted to perform their specified tasks.
- **[TM4]:** Fixed pre-determined membership set. No trust requirements.

Bitcoin and Ethereum are the two best-known blockchains that are fully open, though there are many more, including their respective forks Bitcoin Cash, Bitcoin SV, and Ethereum Classic. XPR (Ripple) and XLM (Stellar) are public but validation has a trust requirement. The IBM-Maersk TradeLens shipping blockchain, based on Hyperledger, was originally controlled by IBM and Maersk, though other "TrustAnchors" have been added as validators[2]. At Lehigh University, the HawKoin prototype replacement for the current GoldPlus student purchasing system uses a centrally managed implementation of Hyperledger due to the need to prevent student-to-student transactions.[3]

3.3 Failure-Mode Assumption for Nodes

Node failure modes range from the simplistic *fail-stop* model in which nodes fail only by stopping and never do anything wrong. At the other extreme, in Byzantine failure, failed nodes become omniscient about the state of other nodes and of the network and collude in an arbitrary manner to disrupt the system.

- **FND1:** Fail-stop. Nodes may cease to execute but never operate incorrectly.
- **FND2:** Byzantine failure. Nodes may fail arbitrarily and no assumptions may be made as to the nature of that failure. Safety thus mandates a pessimal assumption. For correctness, the number of failed nodes must be bounded.
- **FND3:** Infrequent Byzantine failure, with the expected number of failed nodes well below the upper bound.
- **FND4:** Limited Byzantine failure with the limited extent of failure defined by the trust model.

Addressing the most general and malicious types of failure provides maximum safety, but necessitates most costly consensus protocols.

3.4 Failure-Mode Assumption for the Network

Here, we treat the network as being a connection medium whose nodes or routers are not part of the blockchain system and thus subject to an independent set of assumptions.

Network failures may materialize as failure to deliver messages or as an arbitrarily long delay in delivering them. A blockchain node cannot distinguish the non-transmission of information from delayed delivery unless delays are bounded. This latter assumption imposes a degree of synchrony on the network. Bounded delays can be an assumed property of the network or a feature implemented by ensuring that late messages are ignored. The latter can be done by cryptographically secured timestamps within messages (assuming clock synchronization) or by periodically selecting new cryptographic keys (using an assumed secure key-distribution mechanism).

Networks may partition such that the blockchain nodes in a single partition may all communicate but none may communicate with other partitions. Partitioning may interact adversely with other assumptions to lead to wrong inferences by nodes unless care in taken in the protocol.[4]

[3]University systems, like Lehigh's GoldPlus, allow students to spend money provided by parents at university-run or university-approved vendors. Allowing student-to-student transactions would risk enabling a university-internal black market.

Protocol	CC	TM	FND	FNT
2PC	CC3.4	TM3	FND1	FNT1a/2b/3a
3PC	CC3.3	TM3	FND1	FNT1a/2a/3a
BFT	CC3.1	TM4	FND2	FNT1b/2a/3a
PBFT	CC3.3	TM4	FND3	FNT1b/2a/3a
BTC	CC1	TM1	FND2	FNT1a/2b/3b
ETH	CC1,2	TM1	FND2	FNT1a/2b/3b
XRP	CC3.2	TM2	FND3	FNT1a/2b/3b
XLM	CC3.2	TM2	FND3	FNT1a/2b/3b
Algorand	CC3	TM1	FND3	FNT1a/2a/3a
Iota	CC1	TM1	FND2	FNT1a/2b/3b
Hyperledger	CC3	TM3	FND2	FNT1a/2b/3b

Table 1: Protocol Classification

Network communication can lead to data errors, but these are easily dealt with at a lower level via error-detecting or error-correcting codes. We thus do not address messages whose content has been altered within the network, nor messages for which the identity of the sender is incorrect.

Finally, messages may be intercepted by nodes that were not the intended recipient. This enables not only theft of information but also a *suppress-replay* attack, in which an adversary captures a message and resends it at a later time aiming to cause the consensus algorithm to fail. The Byzantine node-failure assumption takes this into account. Data security issues for the payload of non-protocol messages can be managed via encryption.

Thus, for network failure modes, we list not a set of categories but rather a set of three binary choices leading to $2^3 = 8$ resulting categories:

- **FNT1a:** fail-stop / **FNT1b:** reliable delivery (i.e., no, or only bounded-time, failures).
- **FNT2a:** robust / **FNT2b:** partitionable.
- **FNT3a:** highly synchronous / **FNT3b:** highly asynchronous.

The final item on the above list is not truly an all-or-none choice. Some protocols assume highly probably synchrony in which a substantial fraction of the messages are delivered within a stated time bound. Pure asynchrony is provably uninteresting as an option due to the theorem of [12] stating that no completely asynchronous consensus protocol can tolerate even a single unannounced node failure. We thus use the terminology "highly {a}synchronous."

4. TAXONOMY: EXISTING PROTOCOLS

In Table 1 we list several protocols and their classification. In this section, we discuss how we classified each protocol and note key features not fully captured in the classification

4.1 Classic 2PC and 3PC

We do not discuss classic distributed database consensus in detail here. See [27] for textbook coverage, [28] for

[4]Historically, this was one of the factors leading to the non-adoption of 3PC in practice.

the original 3PC paper, and [16] for the original 2PC paper. These protocols both assume a single administrative authority controlling the distributed system and a fail-stop model of failures. 2PC uses a single coordinator for a transaction (though distinct transactions may have different coordinators). This leads to blocking if the coordinator fails at an inopportune time. 3PC alleviates this by adding a messaging round (thus the added phase), to ensure distribution of the knowledge that nodes are able to commit if so instructed before the protocol makes a final commit/abort decision. This allows a failed coordinator to be replaced and blocking behavior to be avoided. The classic version of 3PC does not tolerate network partitions since it must be able to distinguish node failure from network failure.

4.2 Classic Byzantine

The classic work on Byzantine consensus, represented here by [21], assumes a given set of n nodes of which f are faulty with $n >= 3f + 1$. A total of $f + 1$ rounds of messaging is required. In each round, every node transmits to every other node for a total of $n^2 - n$ messages. Neither the value of n and f nor the identity of the nodes themselves can be changed once the protocol is running. The network is assumed "fail-safe and of negligible delay." Thus to get the full level of fault tolerance, we must assume, given n that f is the maximum allowed value. This inflexibility does not satisfy enterprise blockchain applications because:

- Most nodes are nonfaulty most of the time. Nevertheless, the full $f + 1$ rounds of messaging are needed in all cases to achieve the full power of the protocol.

- The fixed set of members for the protocol results in a need to quiesce the entire consensus activity in the system in order to change membership.

4.3 Practical Byzantine

The protocol of [9] achieves its "practicality" by incorporating a central coordinator (referred to as the *primary*) into the protocol. This primary chooses the ordering of operations (transactions). With the ordering decided centrally, a decision on an operation can be made by all nonfaulty nodes using a three-phase protocol (reminiscent of the 3PC algorithm of [28]). This entails significant message-cost savings since the number of rounds is a constant independent of the number of nodes, and the messages are such that they can be multicast to all nodes. At the end of the protocol, agreement is reached when $2f + 1$ nodes have replied with the same value.

All messages include a view number so as to enable a *view change* mechanism that protects against failure of the primary. View change is invoked when a node cannot execute a requested operation (e.g. fails). This action leads to a choice of a new primary with a new view number larger than that of prior views and a three-phase process to confirm the new "view" and its view number. Processing in the new view can then proceed.

If many nodes invoke a view change and do so often and maliciously, the message complexity of this protocol rises rapidly. Since such behavior is indeed faulty behavior, at most f nodes can behave in this manner, thus in the worst case create $f + 1$ rounds of messaging to execute an operation, making this no better than the classic protocol. However, if we do not expect nodes to be faulty very often and seek only to guard against the (remote) possibility of as many as f nodes failing, normal protocol operation results in many fewer messaging rounds than the classic algorithm.

Practical Byzantine fault tolerance retains the flaw, from a blockchain-consensus perspective, that it has a fixed set of members. Thus there is a need to quiesce the entire consensus activity in the system in order to change membership.

We classify this protocol as FNT1b, reliable delivery, but note that the protocol itself specifies the needed level of reliability and its implementation. We classify it as FNT3a although it does tolerate some asynchrony. As the degree of synchronization decreases, the likelihood of a view change being invoked increases.

4.4 Leading Public Coins, BTC and ETH

We discuss only the two leading public blockchains, Bitcoin and Ethereum as representatives of this class.

4.4.1 Bitcoin

Bitcoin is the most well-known implementation of proof-of-work (PoW), but its approach is not a viable large-scale enterprise consideration due to the energy costs and high transaction latency. The protocol is self-adjusting so as to add a block in approximately 10-minute intervals, Since PoW includes a risk of not only malicious but also nonmalicious forks, one must wait for several blocks to be added after the block containing one's transaction to have confidence that the transaction is indeed durable (in the sense of 'D' in the ACID properties). Since the *commit* of a transaction onto the chain is only tentative at first, the term *finality* is used to indicate the point where the transaction is durable with an extremely high level of probability. The accepted standard in Bitcoin is to wait for 6 blocks, which results in a time-to-finality of an hour. Such latency is unacceptable for such financial transactions as consumer-purchases and stock trades. An alternative in such cases is to use off-chain accelerators such as Lightning (see Section 5). Another alternative is to rely on trusted intermediaries that promise not to double-spend, but that, of course, deviates from the trustlessness assumption.

Bitcoin has faced throughput issues resulting from the upper bound on block size. This matter has led to serious disputes in the Bitcoin community and controversy over proposed hard forks to remedy the issue. While we don't elaborate here on Bitcoin governance, the matter is one that would be of possible concern to an enterprise. See [1] for an explanation of the *segregated witness* Bitcoin protocol upgrade.

Aside from matters of consensus, a major factor limiting Bitcoin as an enterprise solution is its limited scripting language, which is not Turing-complete.

4.4.2 Ethereum

The Ethereum blockchain was designed in part to target financial and enterprise applications. It has a Turing-complete language for scripts that are referred to as "smart contracts," supported by a virtual machine (the *Ethereum Virtual Machine* (EVM) into whose instruction set higher level code can be compiled (the standard high-level language is *Solidity*). As we focus here on consensus, we refer to the many online and book sources for an introduction to Ethereum and Solidity such as [5].

Ethereum consensus is a PoW mechanism that, though performing at a higher level than Bitcoin, faced many of the same issues Bitcoin faces. An impending fork will implement a switch to a proof-of-stake (PoS) mechanism, named *Casper* in which validators propose new blocks in proportion to their stake of ETH. The planned upgrade includes sharding of the chain so as to allow parallelism in the processing of new blocks.

The process of switching Ethereum consensus protocols illustrates the form of governance in the public Ethereum blockchain. Ultimately, a majority must agree to any hard fork that is proposed, but the leadership in finalizing the specific proposal comes from Ethereum's founder, Vitalik Buterin, and the Ethereum Foundation. The strength of this leadership was demonstrated in the followup to a bug in the smart-contract code of an Ethereum-based decentralized autonomous organization in 2016 that resulted in the theft of ETH worth tens of millions of US dollars. Despite concerns about the principle of immutability, it was agreed that there would be a hard fork of the Ethereum blockchain to refund the stolen funds. This controversial move led some miners to continue to mine off the old chain, resulting a new cryptocurrency, Ethereum Classic (ETC).

Ethereum has established itself as a major platform for organizations wishing to issue their own currencies without having the create and maintain the underlying infrastructure. By issuing tokens based on the ERC-20 standard, organizations can take advantage of the rich ERC-20 support infrastructure, including wallet software for users. All processing is done on the Ethereum blockchain, eliminating the need to develop a critical mass of miners (for PoW) or validators (for other consensus protocols).

4.5 Currency-Exchange: XRP and XLM

Both Ripple and Stellar are in the business of intermediating international foreign-currency exchange. The legacy SWIFT system still dominates this business, but even in its most modern SWIFT GPI variant, there is a manual component that limits overall performance, with response times at least several minutes. Ripple and Stellar claim transaction times that are only several seconds. Ripple and Stellar each use a cryptocurrency as the intermediate currency in the exchange process: XRP for Ripple and XLM (Lumens) for Stellar.

Both XRP and XLM achieve Byzantine consensus using a quorum-based approach. The use of quora in distributed agreement was first proposed by Gifford [13] in the context of replicated databases. Ripple and Stellar expand considerably on this concept to adapt it to the needs of a blockchain system and the need to linearize blockchain operations.

In Stellar, each node determines its own mini-quorum, called a *quorum slice* and shares that set with all nodes. This slice is independent of the number of nodes in the system. A node selects its quorum slice from among nodes it trusts. That trust is defined by the node's administrator and may be based on various considerations, such as business relationships or public reputation. Thus instead of placing a premium on wealth, as in PoS, Stellar allows, for example, low-net-worth non-profits to be in the business of participating in the XLM network for the purpose of keeping the large institutions honest.

Stellar expects every pair of quorum slices to have non-empty intersection. It is robust to a Byzantine failure of a set B of nodes if this intersection property survives the deletion of B from the node set and from every quorum slice. A three-phase agreement protocol ensures that all "correct" nodes agree on the ordering. Correctness in Stellar is a complex concept based on not just the node itself but whether it may be "befouled" by receiving so much incorrect input that it accepts a wrong value. Correct nodes exhibit the safety property that no two nodes externalize different values for the same operation slot. They also exhibit the liveness property that they can externalize new values without the participation of any failed nodes. Note that the use of "can" and not "must" allows for a situation where consensus is possible for a node but never happens. This somewhat weak liveness property is necessitated by the impossibility result of [12].

The XRP blockchain [10], like Stellar, allows a dynamic membership set and allows each node to choose a set of nodes with which it is willing to exchange consensus messages. Such a set is called a *unique node list*. Various versions of the protocol have had differing requirements for the overlap among unique node lists, depending on the model used for potential failure modes. Overlap must be 90% for a Byzantine-style assumption, though lower overlap rates (on the order of 60%) are sufficient for stronger reliability assumptions.

4.6 Algorand

Unlike other consensus protocols we have discussed, Algorand makes use of the power of randomization in consensus protocols, as first shown in [7] and [8]. Algorand allows a selected subset of the nodes, called a *committee* to make the selection of the next block. Members of the committee are chosen via a randomized process in which voting power is weighted by the currency stake held. The distinctive feature membership selection is the use of *verifiable random functions* [18] in combination with public/private key pairs to allow committee members to be identified in a cryptographically verifiable manner without need for any distributed consensus process. Because random selection can select malicious members for a committee, committees must be large enough to make the probability of malicious control virtually zero. In [14], a probability of 5×10^{-9} is deemed acceptable, and in that case, a committee size of roughly 2000 suffices if 80 percent of the total holdings are held by honest users. The committee size must be larger if a lower percentage is honest, and, in any case at least 2/3 of the holdings must belong to honest users. Interestingly, the committee size bound is independent of the overall number of nodes/holdings, meaning that for a very large-scale network, committees are small, though not of insignificant size (for calibration, note that Ethereum has about 10000 nodes).[5]

The consensus process under normal operation then requires only a constant number of messaging rounds for voting. However, that depends on strong synchrony. Periods of asynchrony can lead to forks in the chain that are then resolved when synchrony is restored by the same manner of consensus used to propose new blocks. While this can, in theory, lead to worst-case behavior of repeated forks and resolution of those forks, but the likelihood of this is provably negligibly small.

[5]See ethernodes.org for a current count.

The power of randomization in Algorand allows relatively efficient consensus without any need for trust.

4.7 Iota

Iota [23] has two distinctive features:

- The "blockchain" is not a linear linked list but is instead a directed, acyclic graph (DAG).
- The unit of commit is not a block but rather an individual transaction

More specifically, each transaction is a node in the DAG (called a *tangle* in Iota parlance). An edge (T_1, T_2) means that T_1 *approves* T_2 directly. Paths represent indirect, or transitive, approval and thus the strength of approval for a transaction T is the number of distinct nodes (transactions) T' for which there is a path in the DAG from T' to T.

A new transaction submitted by a node must approve two existing transactions. Nodes are responsible for validating transactions for which they provide direct approval. This lightweight approach to adding a transaction is targeted at Internet of Things (IoT) applications that are run by large numbers of relatively low-powered computers.

Adding a transaction requires solution of cryptographic puzzle (as protection against Sybil attacks). Iota has rules for choosing transactions to approve that ensures approval of leaf nodes (*tips*) in a timely manner. The approval protocol also addresses possible attacks in which the attacker ignores the approval protocol and approves in an adversarial manner. In such situations, conflicting transactions may appear in the tangle and an honest node must choose among them. This cannot be done based solely on prior approvals since they may have come from an attacker. Instead, the honest nodes runs the tip-selection algorithm repeatedly to see which of the conflicting nodes would be most likely to be approved by an honest transaction. An important benefit of this approach is that the work in adding a transaction depends only on computations on the locally stored copy of the tangle rather than the passing of messages over the network. The network is used to communicate updates in a gossip-protocol style but not as part of the consensus process. This is important for low-connectivity IoT nodes.

4.8 Hyperledger

Hyperleder[3] is a Linux Foundation project spanning a variety of open-source, distributed-ledger projects. The most widely known and used framework is Hyperledger Fabric, which has strong support from IBM.

Applications that submit transactions must register with a "Certificate Authority" that manages access to the permissioned blockchain. Fabric takes a distinctive approach to transaction processing that is related in concept to validation-based protocols in database concurrency control. This approach executes transactions before they are ordered in the blockchain, but requires that they be validated before being finally added to the permanent state of the blockchain. In contrast, typical blockchains first choose an ordering of transactions and then validate and execute them. In Fabric, a set of "endorsing peers" execute the transaction and generate and cryptographically sign the transaction's read and write sets. The application that submitted the transaction collects the responses from the endorsing peers and forwards that to the ordering service, which is implement on a (relatively small) subset of the nodes. Once the transaction is placed in the order, the read set is checked to see that data values have not changed and that the endorsement policy was followed. Then the transaction is disseminated to all nodes and it is validated and added to the chain. For further details see [25].

This framework leaves the exact policy as a parameter to the framework, allowing distinct enterprise deployments of Fabric free to choose a mechanism appropriate to the application domain. This flexibility includes not only the specific endorsement mechanism but also the number of peers required for endorsement.

By separating in a clean manner the various system components for transaction processing, the Fabric framework achieves a flexibility that broadens its applicability. Of particular note is that the set of trust assumptions for endorsement can be different from that used for ordering. Of course, any specific deployment includes a specific set of choices of policy and those must be considered carefully by designers of that specific enterprise deployment.

4.9 Notable Omissions

We have not included settlement tokens such as JPM Coin and Libra. JP Morgan's JPM Coin is based on JP Morgan's Quorum blockchain, whose implementation is based very closely upon the Ethereum blockchain source code. Libra, the recent proposed, but not yet deployed, network from a Facebook-led consortium, the Libra Association, is a permissioned blockchain using Byzantine fault tolerance. Libra claims a plan to evolve from being permissioned to being public. The technical approach is presented in [6], but much of the evolution of Libra prior to launch is likely to be driven by regulatory and political considerations, rather than technical ones.[6]

5. CROSS-CHAIN AND OFF-CHAIN TRANSACTIONS

A transaction may need to span more than just a single blockchain for two main reasons:

1. The use of an off-chain system to aggregate transactions into one, larger transaction to be placed on the blockchain. This aggregation is done by an off-chain accelerator that is designed to process transactions more quickly and without the need for distributed consensus. Off-chain transactions take advantage of a situation where 2 or more users have a high degree of trust and therefore are willing to enter into an off-chain channel offering higher performance and lower cost.

2. The need to run a transaction that affects more than just one blockchain. This occurs if a trade is to be made of one cryptocurrency for another without using a fiat currency like the U.S. Dollar as an intermediary. There needs to be a separate transaction on the blockchain of each currency involved but all of these as

[6]This view is similar to that expressed by U.S. Federal Reserve Chair Jerome Powell in his July 2019 testimony to the U.S. Congress, in which he cited the systemic risk that the Libra plans appear to present.

a whole must be integrated into a single global transaction. In an enterprise setting, this is the blockchain version of the legacy problem of federated databases.

Neither of these needs can be met with the standard methods of committing blockchain transactions since each chain's consensus mechanism is internal to that chain.

We discuss briefly representative existing systems that address these issues.

5.1 Lightning

The key feature of the Lightning Network [22] is the ability of two users who conduct many transactions together to move their activity off the underlying blockchain (Bitcoin) and onto the Lightning Network in a private channel. Both users fund the channel initially and are able at any point to terminate the relationship and move their current balance in the channel back to the underlying blockchain. The secure maintenance of this "bail-out" feature is enabled by exchanging signed bail-out transactions that the other can the sign and submit to the blockchain. When a payment is made off-chain, not only are the off-chain channel balances updates, but also new bail-out transactions must be exchanged and the old ones invalidated.

The invalidation of the old transactions is tricky since nothing prevents a party from submitting the old bail-out transaction. Protection arises from the design of these transactions. Rather than paying both parties back immediately, the submitter's payback is delayed using the transaction time-lock feature of Bitcoin. During that time, the other party can take the entire channel's value if it knows a particular secret created by the submitted. These secrets are exchanged to invalidate the old bail-out transactions. With the secrets exchanged, no one has an incentive to submit the old bail-out transaction since doing so would allow the other party to walk away with the entire amount of currency in the channel. The cryptographic details are in [22]. Ethereum side-chains are discussed in [24].

The Bitcoin basis of Lightning limits its enterprise applicability, but the concept of low-overhead, high-performance transactions between trusting (or at least partially trusting) users is quite promising in enterprise settings. It allows scale-up to transaction rates closer to the norm in OLTP systems.

5.2 Loopring

The typical cryptocurrency exchange is a centralized service in which users submit their private cryptographic keys to the exchange and then the exchange can execute trades on behalf on their clients. If such an exchange is hacked (and several have been, often with major publicity in the media), stolen keys enable the perpetrator to steal cryptocurrency. Due to blockchain immutability, restitution is impossible.[7]

Loopring [29] is one example of trade-matching system that enables smart-contract-based cross-chain trades without users have to reveal their private keys to the exchange. Such exchanges, called *decentralized exchanges* or, equivalently, *non-custodial exchanges*, protect users by not requiring them to disclose their private keys. Loopring enables the creation of trade rings to fill submitted trade orders, and the secure implementation of those trades. Several wallets and exchanges are being constructed on this platform, one example of which is Dolomite.[8] Several other decentralized exchanges are based on Loopring or on competing platforms.

Enterprise blockchain databases may not need trading features such as these, but the secure cross-chain transaction systems created here have the potential of forming the foundation of interoperability between separately controlled blockchains, without requiring the owners/operators of those chains needing to build a trust relationship.

[7]Unless, of course, blockchain immutability is compromised by a hard fork.

6. REVIEWING THE OPTIONS

Returning to the criteria we discussed in Section 2.2, we see that the nature of governance and trust in the application domain plays a dominate role in choosing an appropriate blockchain solution. The willingness (or perhaps requirement) to have a centralized permissioning authority allows greater flexibility in the consensus mechanism used. Members of the blockchain, though approved by the permissioning authority, may nevertheless have some degree of mutual distrust. If there is no trust whatsoever, certain consensus mechanisms are immediately ruled out. However, where some trust exists (e.g., each node has several others that it trusts) it becomes possible to choose a possibly simpler or more performant consensus mechanism. Trust has another important influence on performance, even in systems without any trust assumption. Some protocols (e.g. Algorand) will perform better it situations where the expectation of honesty is high (and thus the rate of Byzantine failure likely to be low).

The range of performance options is substantial. At one extreme is Bitcoin where finality for a transaction takes about an hour. At the other extreme are highly controlled, permissioned chains, where the ordering process is fast, success occurs with very high probability, and time-to-finality is on the same order and initial addition of a transaction to the blockchain. In between these extremes are a variety of improved versions of public blockchains (such as the recent upgrades to Ethereum), the employment off off-chain transactions (as in the case of Lightning), as well as newer consensus protocols that provide strong security guarantees (including from Sybil attacks), while offering the potential for good performance (e.g. Algorand).

Beyond technical considerations, there are various non-technical business decisions regarding the choice of a blockchain solution:

- Maintenance and control of the underlying platform: What is the consensus mechanism for software updates? Does it follow the mechanism used for the blockchain?

- Long-term stability of the underlying platform: Is there strong reason to believe that the underlying chain (for public blockchains) or the software used will continue to be maintained?

- Resource contention, in the case of a public blockchain.

- Public sentiment regarding the platform: Unsavory uses of a specific platform, though unrelated to the

[8]https://dolomite.io

enterprise, may create a negative public image in the media that, though undeserved, represents a real cost.

- Evolution path: Is there a plan to move or a likelihood to move towards a more (or less) permissioned system, towards more (or less) trust, etc.? Several of the consensus mechanisms we have discussed have performance that may be influenced significantly by such evolutions, as we have seen. Hyperledger's ability to allow chances to the consensus mechanism may be an advantage in this regard.

7. CONCLUSION / FUTURE DIRECTION

Blockchain systems emerged from a strong assumption about privacy, anonymity, trustlessless, and freedom from all forms of central authority. The world's financial systems and business enterprises operate under much more stringent contraints. The adaptation of blockchain technology to this more constrained setting remains a work-in-progress both technically and in terms of regulatory oversight. As a result, an enterprise developing a blockchain database faces an environment much more fluid that even that of the early days of database systems. Will one platform (or a small number of platforms) emerge as a de-facto standard? Will there instead be a vast number of options arising from competition and/or a determination that the "one size fits all" concept does not apply?

Research questions remain open in many areas, not only the areas of consensus and performance on which we have focused here. Data storage is prohibitively expensive on many blockchains (notably Ethereum). Today's solution is decentralized, off-chain storage (such as Storj[9] and Sia[10]), but a linkage between a blockchain database and a relational database would appear more promising to an enterprise if proper transactional guarantees are provided.

While the exact future path is uncertain, the vast scope of enterprise applications for blockchain databases (see a list in [26]) suggests an exciting future for blockchain database research.

Acknowledgement

The author would like to thank Luke Bernick, Masoomeh Javidi Kishi, and Roberto Palmieri for many helpful comments on an earlier draft.

8. REFERENCES

[1] N. Acheson. What is SegWit? Technical report, Coindesk, https://www.coindesk.com/information/what-is-segwit, 2018.

[2] I. Allison. Ibm, Maersk finally sign up 2 big carriers for shipping blockchain. Technical report, Coindesk, https://www.coindesk.com/ibm-maersk-finally-sign-up-2-big-carriers-for-shipping-blockchain, 2019.

[3] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. D. Caro, D. Enyeart, C. Ferris, Y. Manevich, S. Muralidharan, C. Murthy, B. Nguyen, M. Sethi, G. Singh, K. Smith, A. Sorniotti, C. Stathakopoulou, M. Vukolić, S. W. Cocco, and J. Yellick. Hyperledger Fabric: A distributed operating system for permissioned blockchains. In *Proc. Eurosys'18*, April 2018.

[4] D. T. T. Anh, R. Liu, M. Zhang, G. Chen, B. C. Ooi, and J. Wang. Untangling blockchain: A data processing view of blockchain systems. *IEEE Transactions on Knowledge and Data Engineering*, 30(7):1366–1385, July 2018.

[5] A. M. Antonopoulos and G. Wood. *Mastering Ethereum: Building Smart Contracts and DApps.* O'Reilly Media, 2018.

[6] M. Baudet, A. Ching, A. Chursin, G. Danezis, F. Garillot, Z. Li, D. Malkhi, O. Naor, D. Perelman, and A. Sonnino. State machine replication in the Libra blockchain. Technical report, The Libra Association, 2018.

[7] M. Ben-Or. Another advantage of free choice: Completely asynchronous agreement protocols. In *Proc. 2nd ACM Symposium on Principles of Distributed Systems*, pages 27–30, August 1983.

[8] G. Bracha and S. Toueg. Asynchronous consensus and broadcast protocols. *J. ACM*, 32(4):824–840, October 1985.

[9] M. Castro and B. Liskov. Practical byzantine fault tolerance and proactive recovery. *ACM Trans. on Computer Systems*, 20(4):398–461, Novemeber 2002.

[10] B. Chase and E. MacBrough. Analysis of the XRP ledger consensus protocol. Technical report, Ripple Research, 2018.

[11] T. T. A. Dinh, J. Wang, G. Chen, R. Liu, B. C. Ooi, and K.-L. Tan. Blockbench: A framework for analyzing private blockchains. In *Proc. ACM SIGMOD Conference on the Management of Data*, pages 1085–1100, 2017.

[12] M. J. Fischer, N. A. Lynch, and M. S. Patterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, October 1985.

[13] D. K. Gifford. Weighted voting for replicated data. In *Proc. 7th ACM Symposium on Operating System Principles*, pages 150–162, December 1979.

[14] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich. Algorand: Scaling Byzantine agreements for cryptocurrencies. In *Proc. 26th ACM Symposium on Operating Systems Principles*, pages 51–68, March 2017.

[15] L. Lamport. The part-time parliament. *ACM Transactions on Computer Systems*, 16(2):133–169, May 1998.

[16] B. W. Lampson and H. E. Sturgis. Crash recovery in a distributed data storage system. Technical report, Xerox Palo Alto Research Center, 1979.

[17] D. Mazières. The Stellar consensus protocol: A federated model for internet-level consensus. Technical report, Stellar Development Foundation, 2016.

[18] S. Micali, M. O. Rabin, and S. P. Vadhan. Verifiable random functions. In *Proc. 40th IEEE Symposium on Foundations of Computer Science*, 1979.

[19] S. Nakomoto. Bitcoin: A peer-to-peer electronic cash system. Technical report, Bitcoin.org, 2008.

[9] storj.io
[10] sia.tech

[20] D. Ongaro and J. Ousterhout. In search of an understandable consensus algorithm. In *Proc. USENIX Annual Technical Conference*, pages 305–320, 2014.

[21] M. Pease, R. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *J. ACM*, 27(2):228–234, April 1980.

[22] J. Poon and T. Dryja. The Bitcoin Lightning Network. Technical report, lightning.network, 2016.

[23] S. Popov. The tangle. Technical report, The Iota Foundation, 2018.

[24] P. Robinson, D. Hyland-Wood, R. Saltini, S. Johnson, and J. Brainard. Atomic crosschain transactions for ethereum private sidechains. Technical report, University of Queensland, 2019.

[25] A. Sharma, F. M. Schuhknecht, D. Agrawal, and J. Dittrich. How to databasify a blockchain: the case of Hyperledger Fabric. Technical report, Universität Saarland, 2018.

[26] A. Silberschatz, H. F. Korth, and S. Sudarshan. *Database System Concepts, 7th edition*, chapter 26: Blockchain Databases. In [27], 2020.

[27] A. Silberschatz, H. F. Korth, and S. Sudarshan. *Database System Concepts, 7th edition*. McGraw Hill Education, New York, NY, 2020.

[28] D. Skeen. Non-blocking commit protocols. In *Proc. ACM SIGMOD Conference on the Management of Data*, pages 133–142, 1981.

[29] D. Wang, J. Zhou, A. Wang, and M. Finestone. Loopring: A decentralized token exchange protocol. Technical report, loopring.org, 2018.