Factors that Impact Blockchain Scalability

Peter W. Eklund
Cyber Security Research and Innovation Centre (CSRI)
Deakin University
Geelong, Australia
peter.eklund@deakin.edu.au

Roman Beck
European Blockchain Center
IT University of Copenhagen
Copenhagen, Denmark
romb@itu.dk

ABSTRACT

Blockchain systems (more precisely Distributed Ledger Technologies (DLTs)) represent a different digital ecosystem compared with traditional computer systems. One major difference are the performance and scalability factors which will be discussed and analytically investigated in this paper. In doing so, we provide guidance for defining a research agenda focusing on the investigation of the crucial role of scalability for DLT systems. System performance – measured in terms of (1) consensus response time (blockchain network latency or time to convergence/agreement); (2) number of transactions per second or throughput, and (3) computing (and power) resources consumed – can be understood by considering the design dimensions of a DLT system, namely: (i) the type of DLT system needed from a requirements perspective which in turn determines; (ii) the complexity of the consensus protocol used; (iii) the topography of the anticipated traffic flow on the network; (iv) the performance and complexity of the domain-specific language that implements smart contracts; and (v) by the anticipated growth in size and complexity of the distributed ledger itself.

CCS Concepts

•**Computing Methodologies** → **Distributed Computing Methodologies;** •**Networks** → *Network Performance Evaluation;*

Keywords

distributed ledger; blockchain; survey of system performance; distributed network scalability.

1. INTRODUCTION

Blockchain systems or Distributed Ledger Technologies (DLTs) became widely known in the form of implementations for cryptocurrencies, most prominently Bitcoin [Nakamoto 2008] and Ethereum [Buterin 2014]. Accompanying the ups and downs of Bitcoin (and other cryptocurrency) prices, as well as the mixed experiences investors have had with Initial Coin Offerings (ICOs), many more innovative business opportunities have emerged around the use of DLT systems in supply chain, finance, logistics, and other business verticals, that are promise to add economic value. Thus, while DLT systems where started life in cryptocurrencies, many of the interesting innovations actually take place outside of that field. For this reason, non-cryptocurrency applications of DLTs are the starting point for our study. These systems elicit different requirements – and exhibit different characteristics – to those used in cryptocurrencies – which are on the whole better understood through their practical use for tokenizing de-centralized peer-to-peer financial transactions.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

MEDES 2019 November 12–14, 2019, Limassol, Cyprus
© 2019 ACM. ISBN 978-1-4503-4895-9…$15.00
DOI:

1.1 DLTs and Blockchains in the IoT World

While applications that use crypto-secure distributed ledgers are attractive, the technology confronts significant hurdles that are interesting to investigate and challenging to fully understand. Performance and scalability hurdles are key among these. The original implementations of public crypo-secured distributed ledgers – Bitcoin and Ethereum are neither space nor time-scalable – in their pure form – for high-frequency transaction-based real-time systems such as those used in the Internet of Things (IoT). This is unsurprising because that is not their design purpose.

There are two technical reasons why the architecture of permissionless public blockchains is unsuited to IoT applications. The first is that the ledger (i.e. a database) is distributed to all participants in the network, imposing storage and latency costs for every node in the network. The second is that the permissionless public blockchain model that delivers consensus in the network involves assigning an ever-diminishing reward to incent participants to validate transaction blocks by computing the solution of computational hard crypto-puzzles, so called proof-of-work consensus [Vukolić 2016] (a.k.a. "moderately hard functions" [Abadi et al. 2005, Dwork and Naor 1993]).

This approach is intentionally computationally expensive, in an attempt to thwart miscreant players and transactions. The commitment by network participants to computational work to validate the blockchain is a surrogate for commitment to the stability and trust in the distributed ledger as a whole. The trade-off between being deliberately slow to agree is strong transactional-trust between the participants.

Specific features of blockchain systems are their peer-to-peer distributed (often decentralized) trust network, their tamper-resistant record of events, and the ability – in DLTs of the second generation and higher – to execute domain-specific languages within the distributive ledger architecture itself: implementing a form of distributed deductive database where the deductive rules are framed by the term "smart contracts".

These attributes make the technology appealing to applications of edge/fog computing and the Internet of Things (IoT), but the performance of most DLT frameworks for IoT applications is costly, both in terms of hardware infrastructure as well as network latency. IoT-DLT implementations in realistic scenarios with high-frequency transactions are slow in practice. In addition, the volume of data being produced by IoT devices means the size of a distributed ledger would overwhelm the limitations of most IoT devices. At the same time computational demands on devices to compute consensus — even via proof-of-stake [Iddo et al. 2014] – would cost the scarcest IoT resource of all, power on the device itself.

Before we begin, we need to ask, what is the compelling reason for using DLT systems from a requirements perspective? Why use DLTs such as blockchain and when do you need to use blockchain? And what is the reason one would use blockchain for IoT applications?

1.2 When to use a Distributed Ledger System

According to Wüst and Gervais [Wüst and Gervais 2017], the dimensions of distributed ledgers and centralized data management systems are the degree of public verifiability, the transparency (or otherwise) of the update process, privacy (both in terms of exposure to certain data fields, as well as participant-identity), integrity (namely tamper-resistance), the degree of data redundancy and the presence/absence or degree of authority given to a trust anchor (self-sovereignty).

What results from their study is a decision system for when a blockchain system should (or should not) be used. In short, if the database state is not to be stored and there are not multiple writers, a traditional DBMS is a better and more efficient choice than a distributed ledger. Furthermore, if the participants are willing to use a trusted third-party, a centralized trust-authority, a blockchain is likewise unnecessary. What remains is Fig. 1, a summary of the decision space for the selection of the type of blockchain solution.

Permissionless public blockchains, the best-known being Bitcoin and Ethereum, allow anyone to interact with the ledger, and additionally anyone to participate in consensus. On the other hand, a permissioned public DLT system is one where both participation and consensus-building are managed, the result being that permissioned blockchains have no requirement for brute-force computing power to reach consensus, because all participants are credentialed.

Permissioned blockchain-systems, where all participants are credentialed, typically use proof-of-stake [King and Nadal 2012]/proof-of-activity [Liu et al. 2017] consensus mechanisms. These consensus methods are simpler than those used in public permissionless blockchains . In these cases, consensus is more about sharing the burden of consensus-building and distributing consensus to minimize the risk of network failure (we discuss consensus protocols later in Section 3).

From Fig. 1 we can see that where it is important for the ledger's state to be publicly verifiable then a permissionless public blockchain system solution may be selected. A permissionless public blockchain system might need to be publicly verifiable in a business ecosystem where there is a regulator, such as a Government tax authority. If public

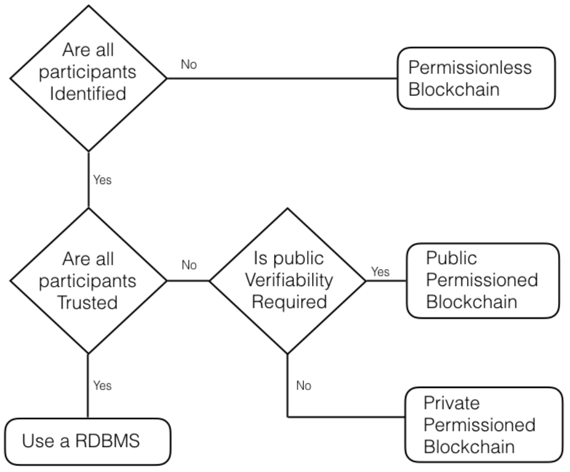
Are all participants Identified
No
Permissionless Blockchain
Yes
Are all participants Trusted
No
Is public Verifiability Required
Yes
Public Permissioned Blockchain
Yes
No
Use a RDBMS
Private Permissioned Blockchain

Figure 1: A decision chart on which blockchain is the appropriate technical solution to solve a problem, adapted from Wüst and Gervais [Wüst and Gervais 2017].

verifiability is not a criterion, then a permissioned public or private DLT system could be sufficient.

Read and write permissions of a fully-private blockchain are controlled by a single organization, while consortium or public blockchains distribute read/write permissions across a permissioned consortium in an attempt to add the extra security of decentralization. Thus, while the consensus mechanism for measuring blockchain scalability and performance is important, so too whether it is a permissionless-public or permissioned private or public blockchain. The intended application requirements of a DLT system can therefore – along these requirements analysis lines – determine the complexity and performance of a blockchain system implementation.

1.3 Paper Structure & Overview

This paper is structured as follows. In Section 2 we look at the topography of the distributed network and in particular pay attention to a characterization of traffic over the network, in terms of volumes of transactions and whether the network traffic exhibits properties more (or less) similar to scale-free networks [Barabási et al. 1999]. In Section 3 we take a look at some of the consensus algorithms used in popular blockchain implementations, since these are an important contributor to blockchain complexity and scalability. In Section 4 we examine the design features and execution complexity of the smart contract programming language, here there are two ways of thinking about smart contracts, the first is in terms of programming language design, and in particular the language ability to secure programming patterns, and secondly in terms of the execution environment for the language. Because every transaction is recorded in the blockchain, in Section 5 we examine so called 'blockchain bloat' (a condition when the data stored in the blockchain achieves very large sizes due to increasing numbers of users and recorded transactions) and consider ways that blockchain implementations have overcome bloat, by delegating the responsibility for storing the entire history to miners or by compressing or pruning the blockchain data structure.

2. NETWORK TOPOLOGY ANALYSIS

Research focused on the Bitcoin blockchain illustrated that the Bitcoin traffic shows that it is a scale-free network [Barabási et al. 1999]. In a similar style of research, but more nuanced, Chen et al. [Chen et al. 2018] use a graph analysis approach for the analysis of money-flows and smart contract distributions on Ethereum. The authors look at three different dimensions of Ethereum's logical network distribution, namely the flow of money (Ether) transfers (so called Money Flow Graph (MFG)), the patterns for the creation of smart contracts (Contract Creation Graph (CCG)) and the sequences of smart contract invocation (Contract Invocation Graph (CIG)).

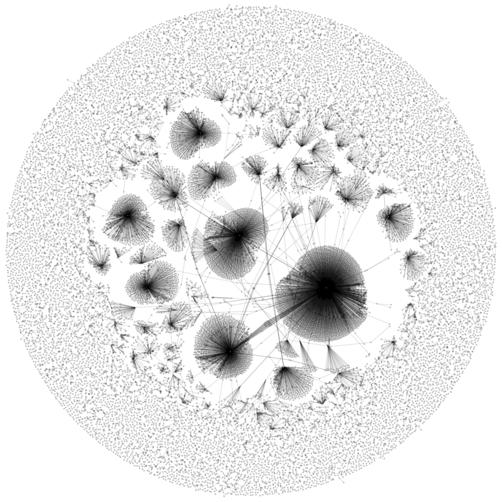

Figure 2: A selection of 20,000 edges from the Money Flow Graph show clear scale-free network features (reprinted with permission from Chen et al. [Chen et al. 2018])

Analysis is then conducted on the three graphs to measure the node degree distribution (the number of connections or edges the node has to other nodes). Nodes can be accounts (or wallets) or smart contracts.

In the MFG, the degree of an account represents the number of accounts trading with a node. In the case of CCG, the degree of account represents the number of contracts owned by it. The node degree of the CIG is the number of smart contracts invoking it and the number of smart contracts it itself invokes.

If the structures are considered as digraphs then the in-degree of nodes in the MFG represent the number of accounts sending money, and the out-degree, the number of accounts being transferred to. Most Ethereum users prefer direct transfer of Ether rather than using smart contracts and generally speaking smart contracts are under-used.

The in-degree of both the CCG and CIG for a account is zero but for a smart contract it is 1 in CCG and CIG, a single smart contract can be created by only one account. The out-degree in CCG and CIG is the number of contracts created/invoked by it.

The results show that for the MFG graph, the node degree in Ethereum shows that nearly 93% of accounts is less than 6, and that the MFG outliers are small degree accounts that transfer large sums of Ether. For the CCG, the distribution of degree follows the power law, namely a small number of accounts create a large number of contracts.

When connected pathways from accounts to contacts are considered in either direction, if an account creates a lot of contracts, the created contracts also create many other smart contracts. It is evident from the analysis, that a single account on Ethereum created 21% of all its smart contracts and it is apparent that only a small number of developers created most of the smart contracts on Ethereum.

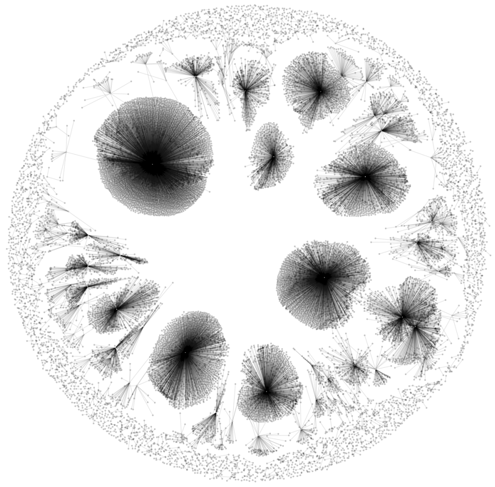

Figure 3: A selection of 20,000 edges from the Contract Creation Graph show clear scale-free network features (reprinted with permission from Chen et al. [Chen et al. 2018])

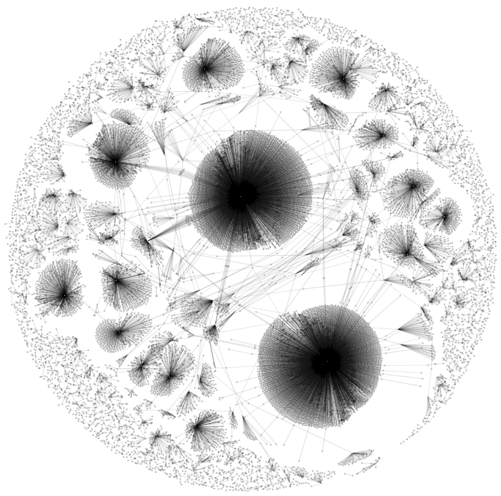

Figure 4: A selection of 20,000 edges from the Contract Invocation Graph show clear scale-free network features (reprinted with permission from Chen et al. [Chen et al. 2018])

3. CONSENSUS ALGORITHMS

3.1 Slow builds trust

Permissionless public blockchain systems such as Bitcoin and Ethereum have well-known latency and throughput limitations. For example, Bitcoin takes up to 10 minutes on average to reach block consensus – *latency* – and accommodates only 3-7 transactions per second – *throughput.*

Ethereum is faster, it can process up to 30 transactions per second and consensus is achieved in less than 10 minutes. However, when compared to traditional commercial distributed systems, this is very slow. An often used comparison is to compare blockchain system latency and throughout to Visacard transactions, Visa being able to process 1,700 credit card transactions per second on average, with a peak performance of up to 24,000 transactions per second, several orders of magnitude faster.

This sluggishness is explained in two ways: (i) that a block on the Bitcoin blockchain consists of about 2,400 transactions and in practice the transactions need to arrive before mining to add a block can start and (ii) by the trust mechanism (Nakamoto consensus) that requires participating nodes to prefer longer transaction blockchains – Bitcoin uses 6 confirmation blocks per transaction, Ethereum uses 12. Nakamoto consensus – preferring longer blockchains – relies on passage of time in order for miners to produce the longer transaction blockchains that then compete for consensus. The result is that the comparatively slow processing time also makes for higher transaction fees as well, since the transaction fee determines which transactions miners include in the 2,400 transactions used to compose a block. In Ethereum the transaction fee is called Gas.

The goal of crypto-currencies such as Bitcoin is to sequentially order transactions on a distributed ledger, hence the name, blockchain. There are many different mechanisms to reach consensus about how a block is added to the blockchain and the complexity of these different approaches have a significant impact on convergence of the blockchain and the overall network latency.

Consensus algorithms are used within blockchains to ensure that participating nodes in the distributed network agree with the state of the blockchain when new blocks are added [Svetinovic 2017]. Blockchain consensus algorithms are called “Byzantine Fault Tolerant” [Lamport et al. 1982], meaning that no one machine can succeed in a malicious attack on the distributed system without being ‘checked’ or ‘detected’ by other nodes in the distributed network. The first implementation of a blockchain for Bitcoin in 2009 uses the consensus algorithm “proof-of-work”[Nakamoto 2008].

Since then, various consensus algorithms have been developed and used, such as “proof-of-activity” [Iddo et al. 2014] and “proof-of-authority” [Angelis et al. 2017]. In the following, we will discuss a few consensus mechanisms, some of which are shown in Figure 3.

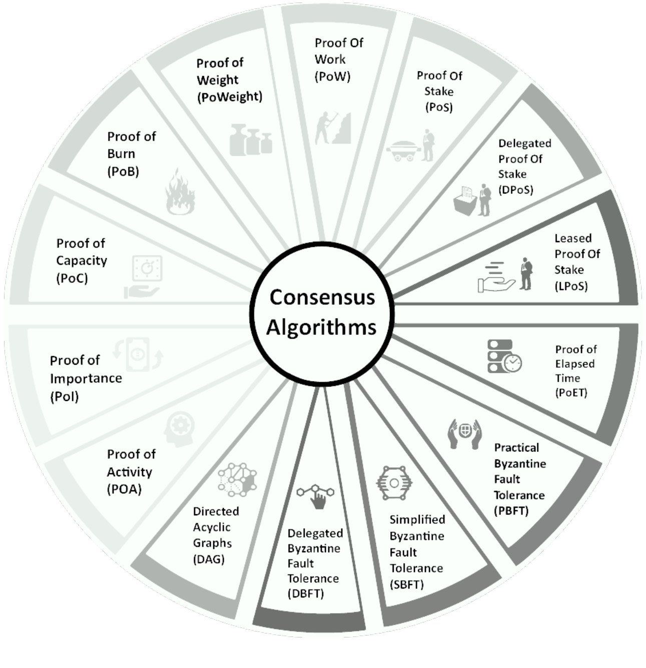
Consensus Algorithms
Proof Of Work (PoW)
Proof Of Stake (PoS)
Delegated Proof Of Stake (DPoS)
Leased Proof Of Stake (LPoS)
Proof of Elapsed Time (PoET)
Practical Byzantine Fault Tolerance (PBFT)
Simplified Byzantine Fault Tolerance (SBFT)
Delegated Byzantine Fault Tolerance (DBFT)
Directed Acyclic Graphs (DAG)
Proof of Activity (POA)
Proof of Importance (PoI)
Proof of Capacity (PoC)
Proof of Burn (PoB)
Proof of Weight (PoWeight)

Figure 5: This figure provides a non-exhaustive list of blockchain consensus algorithms, adapted from 101Blockchains (https://101blockchains.com/)

3.2 Proof-of-work Consensus

The proof-of-work consensus algorithm used in Bitcoin is based on one-way functions to enforce a brute-force method to prove consensus, this in turn requires a large amount of computation (and therefore energy), resulting in slow transaction times, the time it takes for the blockchain to synchronize and update, called blockchain-convergence. The specific proof-of-work algorithm that Ethereum uses is called “ethash”, GPU friendly but designed to require more memory to make it harder to mine using ASICs (Application Specific Integrated Circuit) which are specialized mining chips that are now the only profitable way of mining Bitcoin. Ethereum aims to transition from proof-of-work mining to ‘proof of stake’ – which we discuss below – so buying an ASIC might not be a smart option since they likely won’t prove useful for very long. Kadena uses an PoW-consensus protocol called ‘Chainweb’. It achieves consensus by processing a block through a network of chains that it calls a Chainweb. Each node must validate block headers of some number of pre-specified micro-chains in order to produce its own new block. The effect is a parallel PoW-consensus, merging the micro-chains as a complete integrated blockchain and distributing the work across the entire network.

3.3 Proof-of-stake Consensus

“Proof-of-stake” is a less expensive alternative to proof-of-work. Here, no brute-force computing is required to achieve consensus. Instead, the next node to create a block is selected proportional to some “stake” in the blockchain. In crypto-currency blockchains, the stake maybe the number of coins a node holds combined with how long they have been held [Iddo et al. 2014]. In non-crypto-currency blockchains, the stakes typically resemble the investment one has made into the DLT system. Depending on the consensus mechanism, validator nodes are selected proportionately to their stake or some other algorithm and are allowed to add blocks to the blockchain.

3.4 Proof-of-authority Consensus

In proof-of-authority, representatives are elected how are allowed to add blocks to the blockchain. In contrast to the previous consensus mechanisms, the individual or institution behind the node needs to be known, as the penalty for misbehaving is a decline in reputation and authority. Validators are usually selected via a ballot or lottery [Unterweger et al. 2018].

3.5 Proof of elsapsed time (PoET) Consensus

Hyperledger Sawtooth uses Proof of Elsapsed Time (PoET) as concensus mechanism. Hyperledger is a blockchain consortium under the Linux Foundation. In Hyperledger Fabric, consensus is reached by the election of a leader node who coordinates the broadcast of the blocks to be updated to all the consensus participating nodes on the network. These then respond with a hashcode to the leader who needs to ensure than more than 2/3 of the consensus nodes agree on the same hashcode before the blockchain update is committed. This is a consensus mechanism for permissioned blockchain networks, where any prospective participant must identify themselves before joining the network. In PoET, every node is equally likely to be a validator, PoET distributes validation fairly across the network participants. Each validator is required to wait a randomly chosen time period, and the first to complete the designated wait time earns validation rights. The PoET concept was invented during early 2016 and is part of Intel's Secure Guard Extensions. It's reliance on the Intel platform for the validators limits its relevance to permissioned DLT systems.

3.6 Practical byzantine fault tolerence (PBFT)

Multichain uses a variant of of Practical Byzantine Fault Tolerence (PBFT) similar to Hyperledger Fabric except that only a single node is selected to compute the blockchain update and these nodes are selected in a round-robin or polled fashion, depending on the validator participation historically in the creation of blocks on the blockchain. This is intended to maintain 'validating diversity'. A validator is selected deterministically but based on the sequence of blocks historically created by them on the blockchain, which in practical terms makes it impossible for any one miner to anticipate selection and in so doing impossible to game the system by preemptively seizing control of a given miner.

3.7 Time-based Voting Protocols

Quorem uses yet another variation of consensus for its DLT system. In QuoremChain, a time-based voting protocol is implemented. Nodes on the network alternate from being exclusively *voters* or *makers*. Voters elect the head block by a majority and once there is consensus on the block, the maker nodes signs the block. The partitioning of consensus nodes into non-overlapping roles, and the time-varying allocation of voter/maker roles, mean that there would need to be massive coordination over the network between consensus nodes in order to deceive or tamper with the network. Tendermint is composed of two protocols, a peer-to-peer networking protocol that ensures that only nodes that are active will be obliged to participate in the consensus and a consensus algorithm that assigns a vote or stake. Tendermint polls through block proposers or block validators. Validators with more stake are more likely to be elected as leaders. Once a validator is chosen the protocol proceeds in a similar way to PBFT which we previously discussed.

3.8 Leader Consensus Protocols

The Red Belly blockchain uses a leader-based consensus protocol based on the notion of epochs in which leaders are selected to propose block values and consensus is time-limited. This is called Democratic Byzantine Fault Tolerant (DBFT) where a leader proposes a value v that is multi-cast to all functional nodes on the network. The leader then participates in binary consensus with each responding node until there is a majority in agreement or, if there is not, the process times out and new leader and epoch is selected. The effect is a geographic emphasis on consensus since consensus is made among nodes with lower latency that are reachable first, the consensus is effectively asynchronous in this model.

In NEO consensus works as follows. Anyone owning NEO can vote for a delegate (delegates are called book-keeping nodes). The majority of NEO holders are ordinary nodes that only transfer or exchange assets. Delegates represent validating nodes. They verify each block – generated by the ordinary nodes – written to the blockchain. To become a delegate certain requirements must be met, special equipment, dedicated Internet connections and a certain of amount of GAS (the NEO token) is expected. When it is time to validate a block, it is randomly assigned to a delegate from the pool of delegates. The selected delegate proposes a block and its hash-key. The non-selected delegates decide if the block and its hash-key match their own and confer with the other delegates to verify correctness. If 66% percent of the delegates agree that the proposed block and hash-key is correct, the block is added.

3.9 Other protocols

There are other approaches to consensus not covered here. "Proof-of-Burn" is an idea to address the PoW-overhead of Bitcoin. In this approach, miners show proof that they burned some coins. This is an expensive approach from any point of view. Although it consumes no power, or processor resources, to burn the coins, all proof-of-burn implementations to date work by burning crypto-currencies that have already been mined (at great computing expense) using PoW consensus.

3.10 Consensus Mechanisms Summary

Table 1 is an updated version of the results published elsewhere [Buchman 2016, Cachin and Vukolic 2017]. The table provides us with a road-map to explore consensus protocols but blockchain performance claims can be somewhat misleading. For instance, throughput and response times with 10 validators running on 10 virtual machines in the same server rack will vary greatly from throughput measured on 10 virtual machines geographical dispersed across the globe. On this note, we rarely get concrete details on the resources that provision the performance tests, the nature of the traffic and load on the cloud-services, or the geographic spread of the cloud resources. It seems most of the tests are also conducted without the presence of deception on the network, so we have no sense of how corruptible the protocols are, and therefore no idea how robust or reliable the solutions – compared to one another – when prone to deception.

Framework Name	Consensus Algorithm	OpenSource	Throughput (tx/s)	Response time (secs)
Bitcoin	PoW	Y	3-5	> 500 seconds
Ethereum	PoW	Y	15-30	360
Ethereum Casper	PBFT/PoW hybrid - ethash	Y	≈ 5000	unknown
Ripple	PBFT	Y	50,000	4
NEO	Delegated-BFT	Y	10,000	15-20
Hyperledger Fabric	PBFT	Y	80,000	< 1
Hyperledger Sawtooth	Proof of Elapsed Time (PoET)	Y	more than 80,000	< 1
MultiChain	PBFT + MultiChain	Y	1000-1500	$5 - 10$
Qourum	PBFT + Quoremchain	Y	835	5
Tendermint	PBFT + Tendermint	Y	4,000-10,000	< 1
Red Belly	Democratic-BFT	N	660,000	$2 - 4$
Kadena	Scalable PoW-BFT	N	8,000	< 0.1

Table 1: Some blockchain performance claims, extended from [Buchman 2016, Cachin and Vukolic 2017].

4. SMART CONTRACTS

In blockchain systems, consensus protocols are constrained by the property of byzantine fault tolerance and while each consensus protocol that we surveyed claims this property, the degree of effort required to subvert the blockchain varies, but is largely unknown, untested or unreported.

While the blockchain as a data-structure looks to be an impregnable point of entry for attackers, DLT systems – by virtue of very different architecture to traditional centralized systems – are not immune to new and different types of vulnerability. On the contrary, the highly decentralized architecture for the execution of smart contracts exposes vulnerability at every consensus node that executes an Ethereum Virtual Machine (EVM) in Ethereum. It is worthwhile therefore to explore the execution environment for smart contracts because the performance and vulnerability of the execution languages can give insight to performance, scalability and security.

4.1 Smart Contract Programming Languages

Smart contracts are programs that run on the blockchain network. Smart contract languages (a.k.a. called contract-oriented programming languages) are programming languages that are used to write or specify smart contracts.

Solidity is the pre-eminent language for Ethereum and some other DLT platforms. Other programming languages can also run on the Ethereum Virtual Machine (EVM)[1]. These include Python, Go, Rust, Java Ruby etc. Solidity is believed to be the cause behind the Decentralized Autonomous Organization (DAO) hack in 2016. The DAO, built on Ethereum, was hacked but not through tampering with the Ethereum blockchain on which it ran, but rather through an exploit of the EVM programming language Solidity that fooled a smart contract into spending all its Ether. Solidity suffers from some design flaws including batch overflow – if you overflow a number it overflows silently and resets its value to 0, double-spending is also possible, and unauthorized function calls can be inserted into code.

Bitcoin Script language, the Bitcoin programming language, supports only conditionals, stack manipulation, hashing, and digital-signature verification operations but no loops, thus all programs halt and the language is not Turing complete. Ivy [Hanke et al. 2018] is a higher-level language that can compile to Bitcoin Script, the low-level language used by the Bitcoin protocol to determine whether a transaction is authorized, it is a stack-based language and limited to Bitcoin capabilities, Simplicity [O'Connor 2017] is a typed, combinator-based, functional language without loops and recursion, designed to be used for crypto-currencies and blockchain applications. Simplicity is meant to replace Bitcoin Script, and thus allow for developing abstract and expressive smart contract programming languages.

Flint [Schrans et al. 2018], is type-safe, capabilities-secure, contract-oriented programming language designed for writing robust smart contracts. Flint allows programmers to use caller capabilities to define access control on smart contract functions. To prevent vulnerabilities relating to the unintentional loss of currency, transfers of assets in Flint are performed through safe atomic operations. Formal modeling and verification of smart contracts - e.g. F* [Bhargavan et al. 2016] either using smart contract source code or using their byte code helps avoid the flaws existing in current smart contract programming languages and the lack of practical programmer experience with contract-oriented programming languages. On the other hand, there is another cohort who wants to use model-driven engineering to generate smart contracts from abstract models cite (Towards Model-Driven Engineering of Smart Contracts for Cyber-Physical Systems) to simplify the contract modeling, and automate the source code generation.

4.2 Execution Environments

The Ethereum Virtual Machine (EVM) is Turing complete 256 bit Virtual Machine that allows anyone to execute EVM Byte Code. EVM/Solidity are the most popular development combination for Ethereum but it is possible to compile other domain specific languages into the EVM. Probably the main competitor platform to Ethereum for developing Smart Contracts is NEO[2] which offers an alternative platform to Ethereum. NEO allows developers to use C#, Java, Kotlin, Python or GO programming languages to write smart contracts using a customized version of the Docker VM called NeoVM[3]. Stratis[4] implements the idea of blockchain as a Service (BaaS), supporting smart contracts written in C#.

[1]https://github.com/ethereum/wiki/wiki/Ethereum-Virtual-Machine-(EVM)-Awesome-List

[2]https://en.wikipedia.org/wiki/NEO_(cryptocurrency)

[3]https://docs.neo.org/docs/en-us/basic/technology/neovm.html

[4]https://stratisplatform.com

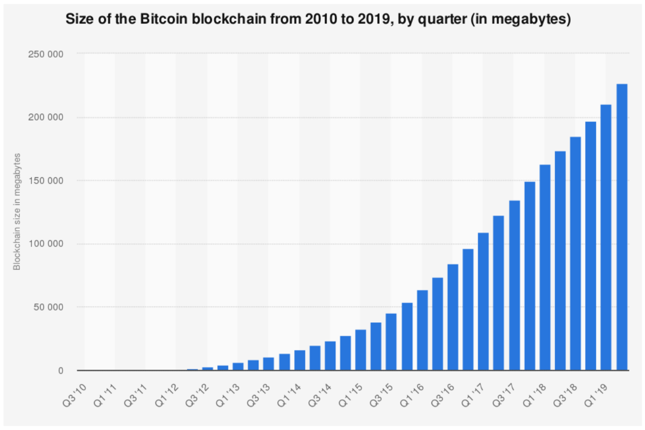

Figure 6: This graph shows the growth of the Bitcoin blockchain from 2010 to Q1 2019, data and graph from https://www.statista.com. As at August 2019 the Bitcoin blockchain is 276GB.

Other systems are LISK (LSK)[5], allowing smart contracts to be written in Javascript and EOS [6] using C++, but could accommodate any domain specific language that can compile into the Web Assembly Language (WASM)[7] which has the obvious utility of being able to run in popular web browsers.

5. ABBREVIATING THE BLOCKCHAIN

Not every node on the Bitcoin blockchain is a full node – full node meaning that the node is required to hold the entire Bitcoin blockchain recording every transaction ever made since 2009. This is fortunate because the size of the Bitcoin blockchain as at August 2019 is 276.67GB[8]. If everyone that used the Bitcoin network had to store 276.67GB of blockchain on their laptop (or phone or wallet) to interact with it, this would exclude many. However, a full node with a blockchain of 276.67GB is most likely stored on a miner-node server-side. The always growing size of all blockchains is sometimes called 'blockchain bloat'. Since the blockchain is immutable, serial and increasingly more popular, blockchain sizes must eventually present a limiting factor on their scalability, particularly in the IoT world. Figure 6 shows the growth of the Bitcoin blockchain.

The complete Ethereum blockchain is 426.98GB as at August 29, 2019[9]. Ethereum is that large because it includes not only the complete history of the blocks, the Ethereum blockchain, but also the state-history, a history of the account balances of all 20 million or more Ethereum clients. This state-history can be pruned to contain simply the current state of participating clients, only miners need the full image. When the state-history is included in the image of a full node the Ethereum blockchain is twice as large as the Bitcoin blockchain, however when the state-history is pruned from the image the Ethereum blockchain is only about 40GB, which is, in practice, all a participating client needs to know and hold in order to transact.

Some blockchain implementations store only the hash-key to a NoSQL database stored off the system, off-chain. This has the advantage of allowing a purpose built DBMS to manage the content off-chain and store only a minute quantity of the actual data. However, it does not ensure immutability of anything other than the hash-key because it is the only part of the data stored on the blockchain. The DMBS then becomes a point of vulnerability for attack, if it is compromised or even removed, the hash-key carried on the blockchain is as a good as useless. The hashes saved in the blockchain can however be used as a checksum to confirm the validity of the data stored in the database. For each individual document retrieved from the database, the blockchain is queried with the appropriate hashes, thus confirming the validity the data retrieved from the database. The blockchain validity of data is always sent to the user along with the data retrieved from the database. This approach is well-understood from traditional databases [Pavlou and Snodgrass 2006].

6. CONCLUSION

Requirements influence to a large extend the computational complexity and scalability of the system. As we have seen, if permissionless public blockchain systems can be avoided, consensus protocols can be simplified and this has a positive impact on network latency and transaction throughput. Efforts by major blockchain initiatives to achieve scalability, and improve transaction throughput and convergence, focus mostly on simpler and more efficient consensus protocols.

Likewise, analysis of the anticipated network traffic patterns based on the topology of the distributed network that underlines the blockchain system, have become a recognized analysis tool for the forensic study of blockchain system performance. If there is a great deal of centralization of the network toplogy, a hub-spoke topography, then performance varies from a less centralized, scale-free peer-to-peer network. Where the nature of the communication occurs via hubs in the scale-free network traffic, the provisioning of validator nodes need to made strategically close to those hubs. If transactions are more randomly point-to-point then validator nodes can be more evenly distributed throughout the network. The provisioning of the supporting hardware and cloud infrastructure also depends on the topography.

The complexity and size of the blockchain structure itself is also a pinch-point for system scalability. The blockchain bloat of Ethereum is avoided by dropping the requirement to distribute the block-states to every validating node with considerable resource saving. While the nature of existing blockchain systems prohibit further reductions, by any other means than unpopular forks, we can anticipate further more compact blockchain variants in future implementations, particular those aimed at IoT implementations.

[5]https://en.bitcoinwiki.org/wiki/Lisk
[6]https://en.bitcoinwiki.org/wiki/EOS
[7]https://en.wikipedia.org/wiki/WebAssembly
[8]https://bitinfocharts.com
[9]https://bitinfocharts.com

7. REFERENCES

[Abadi et al. 2005] Martin Abadi, Mike Burrows, Mark Manasse, and Ted Wobber. 2005. Moderately Hard, Memory-bound Functions. *ACM Transactions on Internet Technology* 5, 2 (May 2005), 299–327. https://doi.org/10.1145/1064340.1064341

[Angelis et al. 2017] Stefano De Angelis, Leonardo Aniello, Roberto Baldoni, Federico Lombardi, Andrea Margheri, and Vladimiro Sassone. 2017. PBFT vs Proof-of-Authority: Applying the CAP Theorem to Permissioned Blockchain. In *Italian Conference on Cybersecurity*.

[Barabási et al. 1999] A.L. Barabási, R. Albert, and H. Jeong. 1999. Mean-field theory for scale-free random networks. *Physica A: Statistical Mechanics and its Applications* 272, 1 (1999), 173–187.

[Bhargavan et al. 2016] K. Bhargavan, A. Delignat-Lavaud, C. Fournet, A. Gollamudi, G. Gonthier, N. Kobeissi, N. Kulatova, A. Rastogi, T. Sibut-Pinote, N. Swamy, and Zanella-B. 2016. Formal Verification of Smart Contracts. In *Proc. of the 2016 ACM Workshop on Programming Languages and Analysis for Security (PLAS16)*. ACM, 91–96. https://doi.org/10.1145/2993600.2993611

[Buchman 2016] E. Buchman. 2016. *Tendermint: Byzantine Fault Tolerance in the Age of Blockchains.* Master's thesis.

[Buterin 2014] V. Buterin. 2014. Ethereum: A next-generation smart contract and decentralized application platform. https://github.com/ethereum/wiki/wiki/White-Paper. (2014). Accessed: 2018-08-13.

[Cachin and Vukolic 2017] Christian Cachin and Marko Vukolic. 2017. Blockchain Consensus Protocols in the Wild. *CoRR* abs/1707.01873 (2017). arXiv:1707.01873 http://arxiv.org/abs/1707.01873

[Chen et al. 2018] Ting Chen, Yuxiao Zhu, Zihao Li, Jiachi Chen, Xiaoqi Li, Xiapu Luo, Xiaodong Lin, and Xiaosong Zhang. 2018. Understanding Ethereum via Graph Analysis. In *Proceedings INFOCOM 2018*.

[Dwork and Naor 1993] C. Dwork and M. Naor. 1993. Pricing via Processing or Combatting Junk Mail. In *Proc. of the 12th Annual Int. Cryptology Conference on Advances in Cryptology (CRYPTO '92)*. Springer, London, UK, 139–147. http://dl.acm.org/citation.cfm?id=646757.705669

[Hanke et al. 2018] Time Hanke, Mahnush Movahedi, and Dominic Williams. 2018. Ivy for Bitcoin. (2018). https://github.com/ivy-lang/ivy-bitcoin Accessed: 2019-09-20.

[Iddo et al. 2014] B. Iddo, C. Lee, A. Mizrahi, and M. Rosenfeld. 2014. Proof of Activity: Extending Bitcoin's Proof of Work via Proof of Stake [Extended Abstract]. *SIGMETRICS Perform. Eval. Rev.* 42, 3 (Dec. 2014), 34–37. https://doi.org/10.1145/2695533.2695545

[King and Nadal 2012] S. King and S. Nadal. 2012. Ppcoin: Peer-to-peer crypto-currency with proof-of-stake. *https://archive.org/details/PPCoinPaper* (2012).

[Lamport et al. 1982] L. Lamport, R. Shostak, and M. Pease. 1982. The Byzantine Generals Problem. *ACM Trans. Program. Lang. Syst.* 4, 3 (July 1982), 382–401. https://doi.org/10.1145/357172.357176

[Liu et al. 2017] Z. Liu, S. Tang, S.M. Chow, Z. Liu, and Yu Long. 2017. Fork-Free Hybrid Consensus with Flexible Proof-of-Activity. Cryptology ePrint Archive, Report 2017/367. (2017). http://eprint.iacr.org/2017/367.pdf Accessed:2017-09-26.

[Nakamoto 2008] S. Nakamoto. 2008. Bitcoin: A Peer-to-Peer Electronic Cash System. https://bitcoin.org/bitcoin.pdf. (Dec 2008). Accessed: 2019-098-20.

[O'Connor 2017] Russell O'Connor. 2017. Simplicity: A New Language for Blockchains. *CoRR* abs/1711.03028 (2017). arXiv:1711.03028 http://arxiv.org/abs/1711.03028

[Pavlou and Snodgrass 2006] KE. Pavlou and R.T. Snodgrass. 2006. Forensic analysis of database tampering. In *SIGMOD Conference*.

[Schrans et al. 2018] F. Schrans, S. Eisenbach, and S. Drossopoulou. 2018. Writing Safe Smart Contracts in Flint. In *Conference Companion of the 2nd International Conference on Art, Science, and Engineering of Programming (Programming Companion)*. ACM, New York, NY, USA, 218–219. https://doi.org/10.1145/3191697.3213790

[Svetinovic 2017] Davor Svetinovic. 2017. Blockchain Engineering for the Internet of Things: Systems Security Perspective. In *Proceedings of the 3rd ACM International Workshop on IoT Privacy, Trust, and Security (IoTPTS '17)*. ACM, New York, NY, USA, 1–1. https://doi.org/10.1145/3055245.3055256

[Unterweger et al. 2018] A. Unterweger, F. Knirsch, Christoph Leixnering, and Dominik Engel. 2018. Lessons Learned from Implementing a Privacy-Preserving Smart Contract in Ethereum. *2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS)* (2018), 1–5.

[Vukolić 2016] M. Vukolić. 2016. The Quest for Scalable Blockchain Fabric: Proof-of-Work vs. BFT Replication. In *Open Problems in Network Security*, J. Camenisch and D. Kesdoğan (Eds.). Springer, 112–125.

[Wüst and Gervais 2017] K. Wüst and A. Gervais. 2017. Do you need a Blockchain? *IACR Cryptology ePrint Archive* 2017 (2017), 375. http://eprint.iacr.org/2017/375