The University of Melbourne
School of Computing and Information Systems
COMP90020 Distributed Algorithms

Assignment 1, 2020

Permissioned and Permissionless Consensus: A Survey and Discussion

Marco Marasco 834482 Austen McClernon 834063

Abstract

The consensus algorithms used in blockchain technologies are the basis of the quality and integrity of the platform. This paper outlines the foundation of the consensus problem within distributed systems, providing a taxonomy of protocols for various types blockchains that address this problem. The review of related works provides an overview of the inherent lack of practicality of Paxos, due to complexity; Raft and subsequent papers were explored to address this issue within permissioned consensus protocols in need of simplicity in practice. Further analysis critiques the dichotomy of underlying contemporary block chain protocols, noting the context and risks present. Key to this analysis is the study undertaken on the requirements for consensus across permissioned and permissionless protocols with respect to each implementation; Integrity, Safety and Termination. Moreover, a case study is performed on Ethereum's novel PoS consensus model Casper. Casper's design details outline an in depth solution to contemporary issues of limited TPS, resource waste and long range attacks that are pervasive in permissionless P2P blockchain consensus protocols. An application was developed to simulate, and test the robustness and durability of the Casper protocol inside a rudimentary blockchain environment.

Introduction

Consensus

Consensus is a crucial and pervasive topic in the realm of distributed systems. Essentially, consensus algorithms are defined protocols for nodes in a distributed network to arrive at an agreement on some value. Consensus algorithms are essential for distributed systems to ensure reliability of the system, particularly when nodes in the system are faulty or behave maliciously. As the consensus protocol a system implements directly impacts the quality of a distributed system, significant research has been undertaken to solve the problem.

Consensus Review

Consider a system model that consists of a collection of processes $p_i, (1 \leq i \leq N)$ that can communicate with each other, via reliable communication. The consensus problem can be defined as follows. For the system to achieve consensus, each process has an initial **undecided** state. Each process proposes to to all other processes a value x_i $(1 \leq i \leq N)$, which is chosen from a set A. Eventually, the processes each set a value of its decision d_i $(1 \leq i \leq N)$, and set their state to **decided**. Thus, consensus has been reached. By this definition of the problem, three requirements are necessary for any consensus algorithm:

- **Integrity:** If all correct processes proposed the same value, then any correct process in the **decided** state has chosen that value.

- **Safety:** The decision value of all correct processes is the same, that is, $\forall i, j \ (correct(p_i) \wedge correct(p_j) \wedge decided(p_i) \wedge decided(p_j)) \Rightarrow (d_i \Leftrightarrow d_j)$

- **Termination:** Eventually each correct process sets its decision value.

In ideal circumstances, such as where system processes cannot fail or a synchronous system (i.e. all messages arrive within some time limit), reaching consensus is no difficult task. However, these idyllic environments exist only in theory, and reaching consensus therefore is not a trivial task. Real life systems are exposed to faults such as:

- **Network partitions:** Connection between processes is interrupted, resulting in a ”partition” of networks.

- **Process failure:** Processes are susceptible to unexpectedly crashing, whether this be a hardware or software fault.

- **Byzantine faults:** Intentionally malicious actions by processes.

Blockchain

A blockchain is an implementation of a distributed ledger, in essence a trusted database of records of digital events that have been executed and shared among participating parties that do not necessarily trust each other. Conceptually, it is a chain of ”blocks”, where each block contains a certain number of transaction records. Blocks are added to the chain via a distributed process carried out by the nodes, with each block containing a cryptographic hash of its previous block, thus ensuring a block has a fixed placement in the chain. Blockchain inherently is based on a trust system between the nodes in the network, nodes coordinate with each other to construct and protect the chain. This in turn requires the system to be dependable to ensure successful service delivery. As outlined in Avizienis et al [Avi+04], the concept of dependability is based on (but not limited to) ensuring the system maintains its reliability, security, integrity and resilience.
Consensus protocols in blockchain implementations are used to decide whether to add a block, and to ensure all nodes create the same blockchain. Like all distributed systems, a key risk for blockchains is dealing with malfunctioning nodes (caused by crashes, network partitions and Byzantine failures). Accordingly, it is crucial that the consensus protocol be fault tolerant to ensure reliable and continual service of the blockchain. The choice of consensus protocol for a blockchain further has significant implications for its applicability. Throughout this report, two types of blockchains are referenced throughout:

Permissionless blockchains: Permissionless blockchains most suited to blockchains operating on public networks. There are no restrictions in place as to who can join as a node in the network, allowing any node to participate in the construction of the chain. Consensus protocols in these public blockchains do not require nodes to be known, so each participating node is anonymous. A well known example of a permissionless blockchain system is Bitcoin [Nak19].

Permissioned blockchains: Permissioned blockchains restrict the participating nodes to a known set, who by the network environment, have permission to be participate in blockchain activities. They generally can possess two types of ledgers. Private ledgers, where permission to join the blockchain network is held by a single entity, these are most suited to privately hosted blockchain networks. The nodes in these systems are now known, but the blockchain data remains encrypted and private. The second type of ledger, the consortium ledger, possesses both private and public ledger qualities. Consortium ledgers are designed that the ”leader” of the network is not a single entity, but a group or ”consortium”. In turn, the risks of how centralised private ledgers are mitigated, resulting in a ledger that is both private, but less centralised.

Background Survey

Permissionless Blockchain Consensus Algorithms

Proof of Work

First published in 1993 by Cynthia Dwok and Moni Naor [DN93], Proof of Work (PoW) was popularised in the 2008 Bitcoin paper by Satoshi Nakamoto [Nak19]. The principle, related to a solution that is difficult to find but easy to verify. Contextually, PoW assumes n nodes in a zero-trust environment. Within PoW implementation, miners perform computation work in solving complex numerical problem to add a block to the network, where the block added next is the one with the longest block height. Note that the computation complexity involved is not related to the transactions to be added into the block, instead it lies linking the new block to the last block in the valid blockchain. The consensus involved revolves around verification of a node broadcasting a completed block to the network, all connected nodes then verify the block and the broadcasting node is rewarded. Hence proof of work. More specifically, relating to Bitcoin’s implementation of the hard numerical problem: Given the previous blocks header hashed H_p, a cryptographic hash of the transactions to be included H_t and some 256 bit number H_{target}, find a number $Nonce$ such that:

$$SHA256(H_t + H_p + Nonce) < H_{target}$$

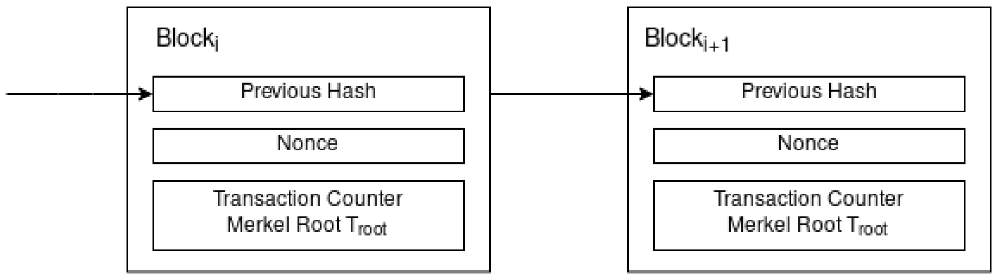

Figure 1: Proof of Work

Figure 1 illustrates a PoW block composition. Note that the Merkle Tree is a hash tree, where each leaf node represents some data input, each internal node is the computed hash of it’s left and right child node. Specific to PoW, each leaf node is the transaction data, while the root node, built upon by each internal node - represents the complete set of hashed transactions. The arbitrary value H_{target} is picked via consensus on the network, block creation time is monitored and if it falls below the targeted 10 minute value, target difficulty is increased and vice versa.

The primary issues of PoW related to the 51% risk, where an entity controlling a majority of the nodes can corrupt the block-chain (approve blocks without proper verification). Resource consumption, where electricity, hardware and the space to perform the complex numerical problem solving is immense [MJ14].

Proof of Stake

Crowd sourced and formalised in Sunny King's and Scott Nadal's 2012 paper Peer-to-Peer Crypto-Currency with Proof-of-Stake [KN12], PoS addressed many of the issues outlined above in PoW and the ideation of PoS forms directly from the aforementioned concerns surrounding security and energy consumption. Within this context, the core principle of PoS is simple: A proof of stake is a cryptographic proof of ownership. As related to PoW, the miner of a new block, known as a forger in PoS is chosen in a semi-random two phase process. Each forger must have a stake - implemented by way of tokens. The selection process weighs forgers with greater stakes as having higher probability - given the relative stake made. Note that unlike in the previous PoW, once the block is validate - the payment is by way of transaction fees or "gas" as it is colloquially known [Poe15]. Importantly, to avoid the largest stake from winning every block, an element of randomness is introduced whereby one of two methods are commonly used. First, Randomised Block Selection: forgers are selected via lowest hash value and highest stakes combined. The second method follows Coin Age Selection - where the deposit age of staked tokens is also taken into account. The relative strengths of PoS lie in its energy efficient implementation, whereby no complex numerical calculations are required to validate a block. Moreover, attackers are required to effectively gamble massive amounts of money to attempt a 51% attack, with no guarantees of success [Bab+12]. The weaknesses of PoS have been highlighted in the popular crytocurrency Ethereum's switch from PoW to PoS, requiring additional security layering. Namely, the *nothing at stake* problem. The Nothing at Stake problem evolves from the minimal downside of staking on two different sides of a validation. Whereby, halting the system as it is unable to reach a consensus. However, Casper - Ethereum's implementation avoids such issues at the cost of possible duplication of stake [BG17].

Directed Acylic Graph

Directed Acyclic Graphs (DAG's) have a deep history in Graph Theory, with the applications to distributed ledger technology being realised and implemented in the novel cryptocurrency Nano (formerly known as railbocks). DAG models have fundamentally different consensus mechanisms. Unlike both PoW and PoS, where the consensus mechanism acts upon deciding the next block to be appended to the blockchain. Participants are able add nodes to the DAG, where each edge represents a transaction. Interestingly, the consensus model as implemented by Nano relies on no leader election, where leader election in PoW and PoS relates to election of the agent enabled to write the next block. Nano [Lem18] instead allows for every node to create it's own transactions. Noting that transactions only require approval or external validation when there is a conflict. In the case of such conflict, a system of representatives is introduced. When an account (root node) is created, it must select a delegate to act as a representative. In the case of conflict, such conflict is resolved via representative voting. Where the representative with the highest weighted vote (sum of all balances of accounts who delegated to this node) is largest [BŽ18].

Introducing a concrete example, we examine IOTA's tangle [Pop18]. The tangle is a DAG, where in order for a participant to propose a transaction, it must first tip two previous transactions. Tipping acts as a way of verifying a previous transaction, whereby the more tips a transaction has indicates the higher confidence in the network of it's validity. Tipping a node forms an edge between the tipped node A and the tipping node B. An indirect tip is possible if there is a directed path between B and A such that the path length is at least two [Pop18]. A formal explanation follows:

1. The participant (node) chooses two other transactions to approve according to a tip selection algorithm, favouring transactions have higher stochastic confidence via cumulative weight tipped.

2. The participant (node) checks if the two transactions conflict, if so it starts again.
3. The participant (node) finally must solve a cryptographic puzzle, much like PoW where it must find a $Nonce$ value that when concatenated to data from the tipped transaction and crypographically hashed has N leading zeroes, in a 256 bit resulting number.

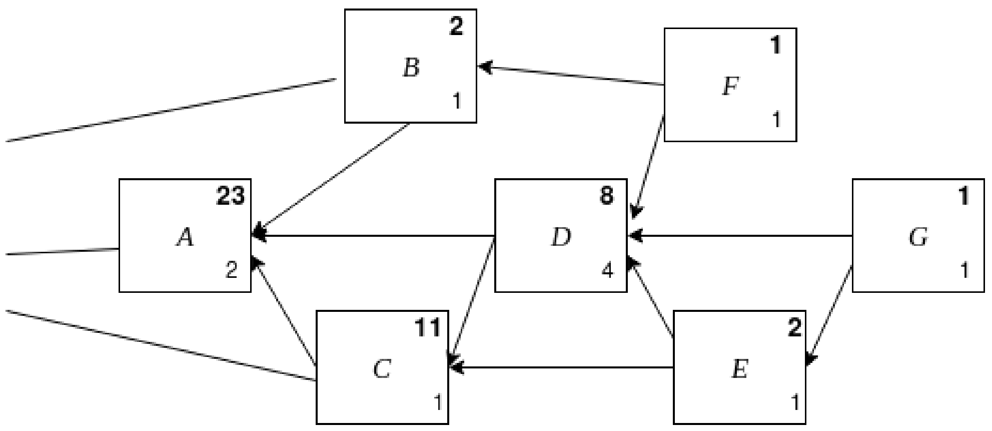

Figure 2: Tangle: Directed Acyclic Graph

Figure 2 illustrates the introduction of node weights, noting that the out-degree of every node is 2. Node's are labelled in lexicographic order of transaction. Note that the weight shown in the bottom left corner, represents the transaction weight W_t. The cumulative weight of a node A can be shown as

$$W_{A_c} = \sum W_{X_t} + W_{A_t}, \exists P_{X \to A}, P = X \to ... \to A$$

. The sum of transaction weights for all nodes for which a path to node A exists.

The DAG model presents a fundamentally different approach, addressing the issues of transaction throughput. Bitcoin enables 3-7 transactions a second, Ethereum 7-15 a second while Nano allows for no inherent cap on TPS - however network latency and inherent limitations has bench marked Nano at 306 peak TPS [Pug18].

Casper

The Casper consensus protocol [BG17] is the latest consensus protocol designed for the Ethereum blockchain platform. The original implementation of Ethereum utilised the Proof of Work protocol as its consensus mechanism, however, Ethereum's developers have decided to transition the platform's consensus protocol to a Proof of Stake based protocol to decrease the computational cost of maintaining the blockchain [Nak19]. In Buterin and Griffith's paper, Casper the Friendly Finality Gadget [BG17], they proposed the Casper protocol, a hybrid Proof of Stake/Proof of Work protocol for the Ethereum platform. This hybrid approach was chosen to reduce the impact risk on the Etherum platform. Casper consists of a Proof of Work mechanism, and a fixed set of nodes called validators. The Proof of Work mechanism adds blocks to the blockchain, however, it does not enforce finality of the block. The blockchain is still exposed to the risk of a malicious node adding multiple children to a given block (as opposed to only adding new blocks to the end of the chain). Casper seeks to address this issue by selecting the correct child for a given block in the chain. Casper enforces finality in the chain by validators in the network casting votes on a block to finalise. To maximise efficiency, this process is completed at defined block intervals. By requiring validators to provide hash's of their public keys when voting, Casper is able to further extend Byzantine fault tolerance by being able to identify, and punish malicious validators in the network.

Permissioned Blockchain Consensus Algorithms

Paxos

In the groundbreaking 1998 paper The Part-Time Parliament [Lam98], Lamport defined the first consensus algorithm Paxos. Paxos is fundamentally a family of crash-tolerant algorithms that are able to achieve consensus on a single decision via message passing in asynchronous systems where processes may fail. The original paper of Paxos portrays the algorithm using a loud narrative of a parliament on the mythical greek island of Paxos, to which does not aid in the difficulty of understanding the algorithm. Several papers [VA15] [KA09] have been written to assist in simplifying the algorithm, a testament to its conceptual difficulty. The algorithm can handle up to f failed processes in an asynchronous system of $2f + 1$ processes (this does not include Byzantine failures). The breadth of the Paxos family algorithms include a plethora of implemented tradeoffs between the number of processes, failure types, message delays, number of sent messages and even activity levels of individual participants. Paxos' robustness makes it particularly useful in ensuring consistency in database replication, with a notable implementation in Google's distributed database application Chubby [CGR07]. Further research has been made into variations of Lamport's original Paxos algorithm to better suit practical implementations. In real life systems, where not one but a series (or stream) of decision requests are received by a distributed system, performing the entire Paxos algorithm for each decision introduces a significant amount of network overhead. A variation of the original Paxos, Multi-Paxos [MPP12], has been proposed to reduce network overheads via adjusting the specific actions each process role performs, and relaxing some of the requirements for system consensus. Other notable variations include Fast Paxos [Lam06], which reduces the number of message delays between the client and the result from 3 to 2, but reduces reduces its fault tolerance, handling at most f faults in a system of $3f + 1$ processes, and Byzantine Paxos [Lam11], which was designed to handle include the handling of Byzantine failures, via introducing an extra verification message within the algorithm.

Raft

The Raft algorithm [OO19] is a consensus algorithm born out of a desire for a conceptually simpler solution to the consensus problem than Lamport's Paxos. Created by Diego Ongaro and John Ousterhout at Stanford University, Raft has been designed to be as efficient as Paxos, and to provide a stronger foundation for implementing it in practical systems. It can manage a replicated log across a distributed system, producing a result equivalent to a Multi-Paxos instance, and has been proved to be as efficient as Paxos [OO19].
Like Paxos, Raft can handle up to half of the processes in a given system failing. Raft's graciously simple structure is comes from a key difference to Paxos, in that is separates some of the key components of consensus, including leader election and log replication, and further commands a stronger level of coordination and coherency between nodes. This in turn, reduces the number of states in the algorithm's execution. There are three types of roles processes take on in Raft, leaders, candidates or followers. The role of the (elected) leader is to replicate the logs to each of its followers. In the context of blockchain, entries to the blockchains ledger in a Raft based system flow in one direction from the leader to the followers. Raft attempts to reduce the frequency of leader elections by implementing the leader to regularly sending "heartbeat" messages to its followers to maintain its authority. If no heartbeat is received by a follower within a certain interval, it becomes a candidate and begins a leader election. Raft's simple design without compromising safety has enabled it achieve enormous success in real-world implementation with notable implementations in distributed code generation from formal models [EL17] and further formal verification [Woo+16].

Practical Byzantine Fault Tolerance

The Byzantine General Problem [LSP82] is a famous analogy in computer science where different parties of a system must all come to an agreement on a particular matter in order to avoid failure, whilst dealing with the possibility of some parties behaving maliciously and are unreliable or disseminating false information. In the context of a distributed system, each party is a different node, and the overall goal is achieving consensus for some value. A system's capability to perform correctly and reach consensus, whilst handling these malicious nodes is classified as its Byzantine Fault Tolerance. Practical Byzantine Fault Tolerance (PBFT) [Cas01] was a method created to handle Byzantine faults within and asynchronous systems with at most f faulty nodes, and least $3f + 1$ nodes in the system.
Prior to PBFT, existing algorithms that handled Byzantine faults assumed synchronous systems and possessed low throughput, which was becoming less and less viable in a rapidly digitising decade [Cas01]. PBFT provides high message throughput as well as low latency, and a relatively inexpensive verification process. Additionally, it ensures privacy within the distributed system by the network asserting messages have not been tampered by other nodes.
PBFT has seen success in being the one of the core consensus algorithms used by the Hyperledger Project. The project, founded by the Linux foundation, with enormous industry support, is an umbrella project for the development and research of private blockchain technology. A notable Hyperledger framework is the Iroha, a private blockchain used by the National Bank of Cambodia to revolutionise their payment infrastructure [HYP18].

Comparative Analysis

Permissionless Blockchain Consensus Algorithms

In order for distributed systems to grow to internet-scale, in a zero-trust environment key assumptions must be made regarding network synchronization, adversarial nodes and required transaction throughput. Importantly, permission-less block chains operate on principle assumptions that all and any nodes can participate within the consensus protocol by choice [Ter20]. The number of participants within permission-less blockchains is also in constant flux, with full anonymity assumed by each node in the network. Moreover the number of participants within the network is unbounded and unknown to any node. Network delay is also asynchronous by default in all permission-less schemes, with no upper limit on network delay. However, as discussed further - the impact of delayed transaction propogation leads to many popular implementations parameterizing consensus protocols in order to create stability. These assumptions that are fundamental to the environment within which many popular permission-less blockchain consensus algorithms operate - such as Bitcoin, Ethereum and Litecoin ultimately mean that classical consensus algorithms that require quorum or some number of cooperative nodes (Byzantine Fault Tolerance) are unable to applied in an absolute. Permission-less consensus algorithms despite these challenges are prolific and countless variations implemented following the Bitcoin Whitepaper in 2008 [Nak19]. The comparative analysis to follow will build upon the previous overview of Proof of Work, Proof of Stake Directed Acyclic Graph consensus algorithms. Particular focus will be made to contrast each algorithms attack vectors, transaction throughput and computational requirement.

Attack Vectors

Sybil Attacks are the most commonly referenced attack vector present within permission-less systems. Peer to Peer systems are notoriously difficult to defend against Sybil Attacks. As permission-less blockchains assume anonymity and dynamic participation within the network, it is possible for an unbounded number of nodes to join and attempt to alter the blockchain. Historically, Sybil attacks have been used to try to break the routing, block access to information or otherwise partition the network [TK14]. Within the context of contemporary blockchain consensus implementations, all aforementioned permission-less algorithms attempt to mitigate attacks via employing variations of a fundamental concept of Proof. Proof of Work requires participant's to solve complex computations in order to participate in the block creation processes and hence be rewarded. Importantly, PoW requires immense computation resources to solve the challenge presented. A successful Sybil attack would therefore require greater than 50% of the resources within the network in order to succeed, which in the case of Bitcoin - the most prolific implementation of PoW would cost $17m$ USD to exert control over 51% of the computation resources in the network for a day [Dic17]. Noting that within a PoW system, the longest chain is considered to have consensus, where forks from this chain are abandoned by the network at no promise of recompense. Although PoW mitigates potential Sybil attacks via the computation expense required to perform, it comes at the cost of efficiency and resource waste. With excessive electricity use, the annualized carbon footprint of Bitcoin alone is comparable to the country of New Zealand [SKG19].

Proof of Stake attempts to mitigate the risks of a Sybil attack at significantly reduced computation cost. Primarily the different lies in an attacker requiring 50+% of the tokens issued within the network - as opposed to 50+% of the computation resources. Notably, an attack on Bitcoin if it were implementing PoS would cost half of the market capitalization of the token, close to 90 billion USD [Dic17]. Note that several other issues arise out of this defense such as Monopolization, Nothing at Stake and initial stake, which will be discussed further at length. Peercoin goes further, to reduce efficacy of Sybil attacks - introducing the novel concept of Coin Age in 2013 [KN12], wherein a token has an attached age indicating either (a) time since creation or (b) time since last staked. This allows for rules to be enforced, limiting rapid re-staking by setting a threshold under which a coin can not be used to stake [TK14].

DAG provides extremely high throughput in the network, with the fundamental concept allowing for minimal verification of transactions unless contradiction occurs within the network. However, due to the dynamic node participation - it provides the greatest possible surface for Sybil attacks, with many implementations layering PoW or other mechanisms on top to prevent such attacks. NANO a popular DAG implementation, requires both the sender and receiver to verify their own transactions using PoW, if transactions do not meet the required standards then NANO further employs a PoS voting system to be used by nodes in the network to reach a consensus regarding the in question transaction [Lem18].

Transaction Throughput

Consensus throughput, which in this paper will be measured as transactions per second, is fundamental to the scalability and success of all three permission-less blockchain consensus protocols examined. As previously stated, PoW mitigates Sybil attacks via crypographic puzzles which are inherently computational expensive and slow. Although the leading number of zeros, for which each block has to have a hash below can be shortened. Increases in performance come at the cost of increased exposure to nefarious actors.

Formally, within the framework described by Cao et al [Cao+20],we define block size L, time to generate a new block T_b and transaction arrival rate λ. It can be shown that TPS of PoW is limited by block size L, as no matter the magnitude of λ - TPS cannot exceed $\frac{L}{T_b}$. This is formalized below.

$$TPS^{pow} = \begin{cases} \sum_{i=1}^{n} \lambda_i, & \sum_{i=1}^{n} \lambda_i < \frac{L}{T_b^{pow}} \\ \frac{L}{T_b^{pow}}, & \sum_{i=1}^{n} \lambda_i > \frac{L}{T_b^{pow}} \end{cases}$$

Likewise, in PoS assuming a coin age implementation where all valid blocks must satisfy the condition $U \leq \frac{stake_i \cdot t_i}{D} \leq 1$ where U represents the target difficulty. $stake_i$ represents the forgers stake. t_i indicates $stake_i$ time since last stake or minting. D represents target difficulty. Subsequently, mining time T_b^{pos} for n stakers is $\frac{D}{\sum_{i=1}^{n} stake_i \cdot t_i}$ [Cao+20]

Following:

$$TPS^{pos} = \begin{cases} \sum_{i=1}^{n} \lambda_i, & \sum_{i=1}^{n} \lambda_i < \frac{L}{T_b^{pos}} \\ \frac{L}{T_b^{pos}}, & \sum_{i=1}^{n} \lambda_i > \frac{L}{T_b^{pos}} \end{cases}$$

Lastly, DAG implementations treat each block (node) as a single transaction, subsequently T_b^{dag} the mining time of a new block is determined by only new transaction arrival time. TPS can be shown as follows:

$$TPS^{dag} = \sum_{i=1}^{n} \lambda_i$$

Comparatively, it is clear that block size L severely limits transaction throughput in both PoS and PoW systems. Moreover time taken to generate a new block T_b which is parameterized in both Ethereum and Bitcoin via cryptographic hash difficulty and to allow for transaction propogation imposes basic limitations on theoretical TPS.

In practice Bitcoin can process up to 7 TPS, notably the a reduction in hash difficulty resulting in a block creation time below 10 minutes weakens the ability to ensure every new block is properly propagated before a new block is mined [Xia+20]. PoS attempts to improve upon PoW TPS shortcomings, primarily through reduced computation requirements. Ethereum's updated PoS model is anticipated to be 150 TPS [BG17].DAG blockchain consensus protocols address the prevalent issue of throughput in a novel minimal validation method, allowing for unbounded throughput upon transaction arrival. More specifically, as transactions themselves provide partial pair-wise ordering in the network [SSG18] with node creation on both send and receive - It is possible to avoid external validation except in the case of network dispute, where layered methods such as Pow or delegate voting is implemented, orphaning transactions until consensus is achieved. As previously noted Nano a popular DAG implementation as a peak throughput of 306 TPS, where limitations again are inherent in a P2P permission-less network as transactions must propagate throughout the network; relying on network speed.

Idiosyncratic attributes

Lastly, the idiosyncratic weaknesses which are otherwise unrelated to the aforementioned comparisons will be analyzed. PoS has further key vulnerabilities which require ad-hoc remediation via layering additional protocols on-top (Casper). Most prominent of these is Nothing at Stake. It is also known as multi-betting or rational forking. As stated previously, PoS requires little extra computation to validate blocks on multiple competing chains. Consequently, if a much lower proportion of the network attempts to multi-bet on all in-play branches they can successfully launch a double spend attack without the required 51% of the economic resources [Xia+20]. Forking from the honest branch is a serious issue however the Nothing at Stake problem can be addressed by the network penalizing multi-bets and implementing automatic forfeiting protocol of stakes. Alternatively, in Snow White and Casper - check-pointing can be implemented to prevent posterior corruption. [DPS19] Centralization risk or monopolisation remains a prevalent risk in all permission-less blockchains consensus protocols discussed. Despite the obvious economic cost either within (PoS) or external to the network (PoW), without large decentralized adoption in networks such as Bitcoin and Ethereum - the risk is very real. Most notably PoS is susceptible to Long-Range attacks where an alternate chain is formed that does not include the genesis block, growing the malicious chain quickly and claiming historical block rewards. As previously mentioned, check-pointing can alleviate these issues by setting an immutable "main" chain from which there is a path to the genesis block.

Conclusively, we have seen that all three permissionless blockchain protocols achieve consensus whilst operating in a dynamic participation, asynchronous and anonymous environment. The degree to which each protocol is tolerant to faults (or attacks) varies by differing metrics with PoW being the most secure but also most resource intensive. PoS attempts to shift the resources required to inside the network via a staking system however this gives rise to potential security issues such as Nothing at Stake, Monopolisation and Long Range Attacks. Moreover DAG blockchain consensus protocols present a novel approach to improve upon limited p2p transaction latency demonstrated in contemporary PoS and PoW implementations, however is still limited by network latency and tolerance to nefarious nodes, which require tertiary layering to resolve disputes and hence inflate transaction confirmation timing.

Permissioned Blockchain Consensus Algorithms

If any algorithm is to have widespread success in adoption into practice, it is paramount that the algorithm is conceptually palatable, or put simply, understandable. An overly complicated algorithm is less likely to generate attention, despite benefits it may hold for a problem. As previously discussed, Lamport's white paper for Paxos was a notoriously difficult piece of academic literature, eventually requiring Lamport to write Paxos Made Simple [Lam01] to remedy the issue. Despite this attempt to straighten out the conceptual difficulties, practical implementations of Paxos still hold little resemblance to the protocol defined in its paper. Whilst Paxos itself is an exceptional consensus algorithm, cited as "the only consensus algorithm" [CGR07], attempts to implement Paxos start off pure, but due to the inherent subtleties and practical difficulties of the algorithm, changes are made to the end product. A prominent example is Google's Paxos based Chubby system, which was described as having "significant gaps between the description of Paxos and the needs of a real-word system" [CGR07]. For practical settings where Multi-Paxos is generally required, implementations have fallen short due to there being no defined practical algorithm for Multi-Paxos. Research has been made into addressing the Paxos' shortcoming in practicality ([Maz07] [KA09]) however these all diverge from Lamport's original design to meet the requirements of practical systems. Paxos' mathematically rigorous proofs demonstrated its correctness, but practical implementations risk losing its correctness, as slight adjustments to the algorithm can have significant impact on the correctness of the adapted protocol. Conversely, both Raft and PBFT were designed to be understandable and practical, a key component to their respective success in practical adoptions.
Ongaro and Ousterhout developed Raft with understandability as the main goal of their paper [OO19]. Having seen the struggles of reliably implementing Paxos in practical settings, it was deemed pivotal to Raft's success to ensure it could be easily understood. Accordingly, during each step of the design phase, Ongaro and Ousterhout ensured wherever possible to break each problem in Raft into smaller, exclusive sub-problems, resulting in clear and easily digested separations for various aspects of the protocol such as leader election and safety guarantees. Ongaro and Ousterhout's made efforts to improve upon Paxos' practicality, as Raft was designed to be non-trivial (from a developer's perspective) in its practical implementation, and not require extensive changes to the protocol to meet system architecture limitations.
Similar to Raft's crusade of defining a practical protocol for crash-tolerant consensus, PBFT, as by its name, moves on a tangent to previous work in the field by providing a practical solution to the Byzantine Fault Tolerance problem. Much of the earlier work into designing Byzantine Fault tolerance [CR93] [GM93] [MR97] was flawed from a practicality perspective. Many of these pre-

vious studies derived solutions from unrealistic synchronous assumptions to guarantee correctness, or produced a theoretically correct, but practically inefficient process to address Byzantine faults. PBFT was explicitly designed to work in asynchronous systems, and provide a level of efficiency that was a viable solution in practical settings. Across all three of the concerned algorithms, from a practical perspective, it is clear Paxos is significantly disadvantaged in its practical applicability. It's difficulty and lack of foundation in consideration for practice have stunted its use and translation to real world implementations. Coupling both the higher degree of understand-ability of practicality of both Raft and PBFT, these two algorithms provide strong practical foundations for developments, and have accordingly seen success in real world implementations [And+18].
As derived by FLP in [FLP85], no deterministic fault-tolerant consensus algorithm cannot guarantee correctness in asynchronous systems. FLP does not conclude consensus is impossible, but under a given algorithm's system model assumption, it is impossible to guarantee the termination of the algorithm. This however is highly unlikely in practice, and as such these algorithms can safely be considered for asynchronous implementations.
Paxos is able to guarantee its safety as claimed in Lamport's Paxos Made Simple [Lam01], however, Paxos' limitation in guaranteeing termination stems from a scenario where processes are continually competing to propose a number to be accepted by other processes. An infinite loop can occur where the proposal numbers are ever increasing and never accepted. Lamport addresses this issue by claiming if enough of the system is operating correctly, termination of the algorithm can be achieved by utilising randomness or timeouts during the proposal phase. However, this termination is not guaranteed, but becomes more and more probable the longer the system behaves synchronously.
Raft is also able to guarantee safety, and provides a more rigorous explanation of how it does so. Defined in Raft's white paper, it's safety requirements were defined using the state machine replication problem [OO19]:

- Election safety: At most one leader can be elected in a given execution round (term).
- Leader append-only: A leader can only append new entries to its logs.
- Log matching: If two logs contain an entry with the same index and term number, then the logs are identical for p all entries up through the given index.
- Leader completeness: if a log entry is committed in a given term then it will be present in the logs of the leaders since this term
- State machine safety: if a server has applied a particular log entry to its state machine, then no other server may apply a different command for the same log.

Raft's safety requirements are easily adapted the context of blockchain, where logs can be considered as the blockchain. Termination issues arise within Raft when competing candidates for its leader election process have split votes. Raft addresses this issue in a similar manner to Paxos, by introducing random timeouts during the election process, and ideally, resulting in a situation where a candidate is able to win the election and progress through the algorithm.
Before analysing PBFT's safety and liveness guarantees, it should be noted that the guarantees of safety and liveness for Paxos and Raft have similar, assumption that heavily differs from PBFT – processes do not behave maliciously. As such, they are only crash tolerant. They can maintain their safety and termination in any system provided a majority of the process in the system have not experienced crash failures. More formally, for a given system of $n = 2f + 1$ processes, at most f processes can be faulty. This fault tolerant bound is optimal for asynchronous systems without Byzantine faults [Lam03].

Accommodating Byzantine faults in consensus has negative impacts on this optimal bound of faulty processes. In ("Byzantine General's Problem" by Leslie Lamport, Robert Shostak, and Marshall) [LSP82], the first formal solution proof for the Byzantine General's Problem concluded that the this bound was increased to $n = 3f + 1$. PBFT however, is able to handle both crash and Byzantine faults. It was proven to guarantee safety meeting these optimal bounds, hence in the perspective of Byzantine Fault safety, it is optimal. Whilst the algorithm was explicitly designed for asynchronous systems, it still is able to guarantee termination so long as the number of faulty nodes is within the previously mentioned bounds, and a "weak synchrony" assumption holds for the network – that for a given moment t, when a given message is sent, the time between t and when it reached its destination does not indefinitely grow faster than t.

When analysing the network assumptions of each of the three algorithms, they each require some level of synchronization in the network to still guarantee termination, assumptions which are more than reasonable with modern networking infrastructure. Purely considering their fault tolerance bounds, Paxos and Raft are superior to PBFT, being able to tolerate up to $\frac{n-1}{2}$ faulty nodes, compared to $\frac{n-1}{3}$. However as mentioned, they do not consider Byzantine faults, so in that respect, PBFT commands superiority.

Accordingly, there appears to be trade-off between being able to tolerate crash faults, and Byzantine faults. In permissioned blockchain networks, blockchain participants are previously known. For any practical setting, the risk of malicious nodes in the network is significantly low, particularly for private blockchains where participants are controlled by a central body. Relating back to the trade-off of fault tolerance, this low risk of Byzantine faults causes a shift towards favoring Paxos and Raft, the significant reduction in overall fault tolerance from PBFT is an unnecessary precaution for permissioned blockchain systems.

Within permissioned blockchains, as the risk of Byzantine faults is not as important, a greater degree of trust can be granted to each participating node. As such, repeatedly performing a voting process to form consensus on generating the chain can also be safely reasoned to be an excessive use of resources, with the main aim of the algorithm being to just ensure safet when replicating ledgers across nodes. Compared with PBFT and Paxos, Raft algorithm is more appropriately designed to capitalise on the relaxed trust issues between nodes. Raft uses a strong form of leadership, so additions to the blockchain's ledger are delegated from the current leader to all other follow nodes. This reduces the network costs of constantly requesting voting to edit the ledger. Although leader election is typically slower in Raft [HM20], it employs stronger requirements for leader election. Raft only allows candidates with the most up to date ledger to become a leader, and the added feature of the 'heartbeat' messages to maintain leadership reduces the frequency of leader elections. Hence from an efficiency perspective, Raft's strong leadership allows for superior addition and replication of the ledger across the blockchain network. This advantage does come at a cost of significant availability interruptions if the leader fails. Paxos and PBFT do not possess the same strictness of only allowing the most up to date node in the network to be the leader, or to propose additions to the ledger. They allow for multiple leaders at a time. These relaxed conditions remove the network bottleneck Raft possesses by restricting flow through a single node and increased cost of leader failure.

For permissioned blockchains, where the network environment and participants are closed and known, careful consideration must be made into deciding what consensus algorithm to use for maintenance of the blockchain. From a development perspective, implementing a difficult algorithm such as Paxos has demonstrated to be a non-trivial task, with end results lacking key safety requirements. Choosing a more understandable, practically based algorithm like Raft and PBFT allows for greater intuition and control in implementation. Depending on the system environment, determining preference between crash tolerance and Byzantine fault tolerance results in different overall node fault tolerance levels, but generally speaking, Byzantine faults are less likely to occur within permissioned blockchains. Lastly, it has been discussed Raft provides greater efficiency in terms of how it replicates the ledger across the blockchain compared to Paxos and PBFT, but at a cost of increasing the impact of leader failure.

Casper

The main role of Casper is to act as a "guardian" over a blockchain network. Consider a blockchain as a tree of n nodes (blocks), in an ideal scenario, a function defining the longest path from the root (referred to as the genesis) to a leaf in the tree, $longestPath(n)$, would behave as a monotonic function. That is, the protocol that proposes blocks only ever proposes to add a block to a leaf block, resulting in a stick tree. Real-world applications rarely follow this ideal structure. If there is significant latency within the blockchain's network, or if a malicious attack is being performed on the blockchain, it is possible that the blocks can be proposed to non-leaf block in the tree, resulting in a block having multiple children, and new leaves in the tree. As a result, multiple possible "chains" exist in the tree, rather than the ideal single chain. Casper's main role is to ensure all nodes choose the "correct" child for each parent block in the tree, thereby establishing a singular path from the genesis to a leaf block. To maintain its ability to scale to larger service larger blockchain networks, Casper only performs these checks on a set of nodes in the tree known as checkpoints. Checkpoints in a blockchain s are defined as $\{c \mid c \in s \wedge height(x) \,\%\, d = 0\}$, where d is the height difference between checkpoints in the blockchain tree. The value d has direct impacts on the efficiency of Casper, as it effectively proportional to the workload of Casper in a given environment. Whilst decreasing d does reduce the overhead of Casper on the blockchain, it comes at a trade-off of increasing the time interval of achieving consensus for checkpoints in the tree. In the original paper for Casper [BG17], setting d to 100 was proposed as an suitable value for this trade-off. Each checkpoint c in a tree also has a value for its "checkpoint height", which is the number of proper ancestors of c that are also checkpoints. Conceptually, Casper only considers blocks in what is defined as the checkpoint tree, a tree of only checkpoints in the blockchain tree.

When a validator joins the blockchain, it submits a number of coins (any medium that can be considered to have value) to the blockchain, this is known as its deposit. In exchange for a deposit, a validator is given the ability to vote on blocks, and the weight of their votes is proportional to the size of their deposit. In traditional Byzantine fault tolerant algorithms, reaching consensus requires agreement between at least $\frac{2}{3}$ of the participants in the network (if they each have equal voting power), whereas Casper requires agreement between a set of participants, whose sum of all their deposits is at least $\frac{2}{3}$ of the total deposited to the blockchain network. Casper creates an incentive for correct behaviour from validators by rewarding and penalising their deposit size, based on if they have been participating correctly in the network or maliciously. For the remainder of the report, when referencing $\frac{2}{3}$ of validators, this will actually be referring to the $\frac{2}{3}$ of the total deposit sum. Furthermore, following the practice established in the white paper for Casper, when discussing the height of nodes in the blockchain tree, this is actually referring to its depth (consider the tree to be upside-down to traditional trees, such that the tree grows upwards).

Casper is able to support a dynamic set of actively participating validators [But+19]. To discuss how it is achieved, the concept of a block dynasty is proposed. A dynasty d of a given block x is the number of finalised checkpoints in the path from the genesis to the parent of x in the blockchain tree. When a validator v joins the blockchain network and submits its initial deposit, its initial join message is added to some block with dynasty d. v is added to the validator set when a block with dynasty $d + 2$ is created, that is, it cannot start voting straight away. The value $d + 2$ is considered the start dynasty $DS(v)$ of v. Each validator also has a value known as its end dynasty $DE(v)$, that is initially set to ∞. When v requests to leave the network, it broadcasts a leave message which is included in a block with some dynasty d. v is removed when a block with dynasty $d+2$ is created, and the value of $DE(v)$ is set to $d+2$. Casper does not allow for validators to re-join the network, once a validator leaves it its identification value (e.g. public key) is recorded and banned.

Voting

A validator participates in voting by broadcasting a vote message to other validators in the network. This vote message is strictly structured to minimise the risk of malicious votes being broadcast. Consider two checkpoints in the blockchain tree, s and t, where a validator v wants to vote on t as a correct checkpoint. In order to consider a vote valid, Casper requires that:

- s is an ancestor of t.
- v is in the set of validators (in practice, identified using its public key).

The vote $\langle v, s, t, h(s), h(t) \rangle$ itself is comprised of various components that are used by Casper to ensure safety:

- v: The validator's public key.
- s (source): The hash of a justified checkpoint.
- t (target): The hash of a descendant checkpoint of s.
- $h(s)$: The height of s in the checkpoint tree.
- $h(t)$: The height of t in the checkpoint tree.

The follow terms are defined to describe Casper specific relations and attributes within a blockchain tree:

- **Supermajority link**: An ordered pair of checkpoints (a, b)in the checkpoint tree, such that a is an ancestor of b, and at least $\frac{2}{3}$ of validators have broadcast votes with a as the source, and b as the target. Note: it is not necessary that a is the parent of b in the checkpoint tree, only that it is an ancestor.
- **Conflict:** Checkpoints a and b are considered conflicting iff a is not an ancestor of b.
- **Justifiy**: A checkpoint b is justified if it is the genesis block, or there exists a supermajority link (a, b), where a is justified.
- **Finalise**: A checkpoint a is finalised if it is the genesis block, or there exists a supermajority link (a, b) where b is a child of a in the checkpoint tree.

Safety and Liveness

A crucial part of Casper's (accountable) safety and (plausible) liveness guarantee is never finalising two conflicting checkpoint, assuming that $\frac{2}{3}$ of validators are behaving correctly. Casper ensures this by defining two rules, known as slashing conditions, for each validator.

Given a validator v, and two separate votes $\langle v, s_1, t_1, h(s_1), h(t_1) \rangle$ and $\langle v, s_2, t_2, h(s_2), h(t_2) \rangle$:

- $h(t_1) = h(t_2)$: No validator can broadcast two votes with the same target height.

- $h(s_1) < h(s_2) < h(t_2) < h(t_1)$: No validator can vote within the checkpoint subtree of another vote.

As mentioned, validators in the network are rewarded or penalised depending on if they submit correct votes. Validators are penalised by violating these conditions, and they have their depost "slashed". From the slashing conditions, it follows that there is **1.** at most one supermajority link (a, b), where $h(b) = n$, and **2.** there is at most one justified checkpoint at height n. These consequences allow for proof of the safety and liveness guarantees. The following discussion assumes that at least $\frac{2}{3}$ of validators behave correctly in the network.

Accountable Safety

Casper's safety guarantee derides from ensuring no two conflicting checkpoints are ever finalised, pending $\frac{2}{3}$ of correctly behaving validators [Cañ19]. As each vote message is stamped with an identifying value for each validator (e.g. public key), Casper's safety is able to be considered accountable safety, due to its ability to identify and punish misbehaving validators from the set of votes. Casper's safety can be proven as follows:

Consider two conflicting nodes, a_m and b_n, who both have a justified direct child (a_{m+1} and b_{n+1} respectively), such that without loss of generality, $h(a_m) < h(b_n)$ as $h(a_m) \neq h(b_n)$ by consequence 1. Consider a chain in the checkpoint tree, that follows the path: genesis $\rightarrow b_1 \rightarrow ... \rightarrow b_i \rightarrow b_{i+1} \rightarrow ... \rightarrow b_n \rightarrow b_{n+1}$, where there exists a supermajority link between all nodes, and b_i is the highest ancestor of b_n such that b_i is not also an ancestor of a_n. By consequence 2 of the slashing conditions, $\forall k \in [i, n+1]$, $\exists! b_k$ $(h(b_k) = h(a_m)) \vee (h(b_k) = h(a_{m+1}))$.

Consider an integer j, such that $h(b_j)$ is the closest greater value to $h(a_{m+1})$ (i.e. $\exists! j'$ $h(a_{m+1}) < h(b'_j) < h(b_j)$). Furthermore, this implies that $h(b_{j-1}) < h(a_{m+1})$, and $h(b_{j-1}) \neq h(a_m)$, by slashing consequence 1. Accordingly, $h(b_{j-1}) < h(a_m)$. The resulting super majority link existing between $h(b_{j-1})$ and $h(b_j)$ is impossible, as it violates the second slashing condition $h(b_{j-1}) < h(a_m) < h(a_{m+1}) < h(b_j)$.

Plausible Liveness

Casper's plausible liveness protocol ensures that no matter the previous events that occurred in Casper's executions in the blockchain environment (slashings, delays, etc), as long as $\frac{2}{3}$ of validators are behaving correctly, it is always possible to finalise a new checkpoint if there are children who extend the finalised chain in the blockchain tree. The proof for this is as follows:

Consider some justified checkpoint a, that has the largest height of all justified checkpoints, and non-justified checkpoint b, that has the largest height for any checkpoint that has been voted for by a validator. A checkpoint c, who is a descendant of a, and $h(c) = h(b) + 1$. It is possible to justify c without violating either of the slashing conditions. c can later be finalised by Casper by adding a supermajority link from c to a direct 'checkpoint tree' child of c.

In Casper, the default behaviour for validators is to add blocks onto the chain in the tree that has the highest justified checkpoint. From the liveness guarantee shown, it follows that this chain can always be added to, as this chain contains the highest justified checkpoint.

Dynamic Validator Sets

For real world settings, participants in public blockchains are (rarely) ever static, they regularly join and leave. As such, it is essential that Casper is able to handle these changes. To ensure that accountable safety and plausible liveness still hold for dynamic sets, each dynasty d is designated two subsets of validators:

- **Forward validator set**: $\{v : DS(v) \leq d < DE(v)\}$
- **Rear validator set**: $\{v : DS(v) < d \leq DE(v)\}$

To maintain accountable safety and plausible liveness, the original definitions supermajority links and finalised blocks are adjusted accordingly.

Supermajority link: An ordered pair of checkpoints (a, b) in the checkpoint tree, such that a is an ancestor of b, b has a dynasty value d, at least $\frac{2}{3}$ of validators in the forward set, and at least $\frac{2}{3}$ of validators in the rear set have broadcast votes with a as the source and b as the target.

Finalise: A checkpoint b is finalised if it is the genesis block, or there exists a supermajority link (b, c) where c is a child of b in the checkpoint tree, and the sets of votes V for the supermajority links (a, b) and (b, c) are contained in b's block, where a is b's parent.

System Failure

Casper's fault tolerance is restricted to the Byzantine fault tolerance bound of no more than $\frac{1}{3}$ faulty validators in the system. If at a given point, more than $\frac{1}{3}$ nodes have failures occurring, whether they be crash, network, or Byzantine, Casper is no longer able to achieve consensus, and the blockchain cannot grow. Casper's designers proposed a method to recover from crash failures known as "inactivity leak". This method progressively drains the deposit size of inactive validators, until the validators who are still active are able to create a supermajority link with their voting power. This inactivity leak however results in the possibility for validators to violate the slashing conditions [BG17], The actual algorithm for the inactivity leak was not defined, and remains an open problem for Casper implementations.

Implementation

To implement the Casper protocol, an application was developed that simulated a blockchain environment, with participants in the blockchain utilising the Casper protocol to justify and finalise blocks. Validators in this simulation propose blocks in a round-robin style, and broadcast their blocks and votes over the simulated synchronous network. Blocks and votes are processed by validators and the Casper protocol is used to ensure that all validators justify and finalise the same blocks. By creating a simulated environment, all aspects of the system were able to be configured to test Casper's scaling and performance across varying factors such as network performance/partitions, Byzantine validators, and how different parameters of the Casper protocol impact the final result.

Advantages

Casper by design only operates on checkpoint blocks within the blockchain. Evidently, this reduces the impact of Casper on the network and creation of the blockchain compared to a consensus algorithm that requires each and every node be "finalised" in some respect. Hence, Casper itself is a lightweight and efficient algorithm to use for Consensus. By utilising the application to simulate varying network latency values, Casper was shown to have very robust tolerance to network latency.

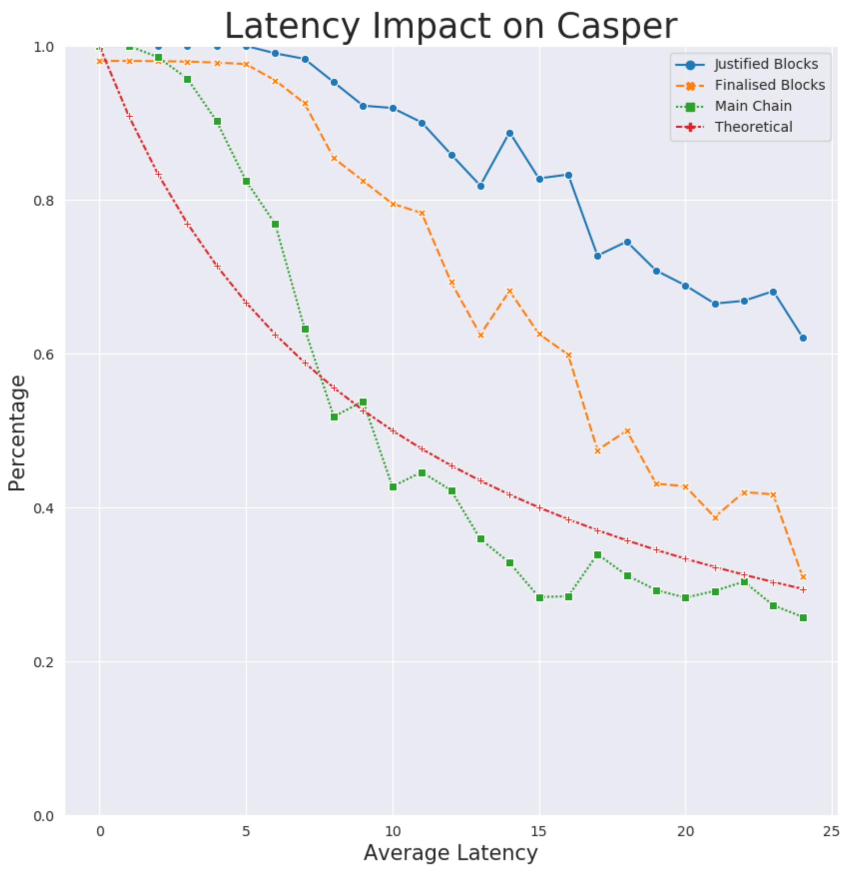

Figure 3: Latency impact on Casper

As plotted in figure 1, it can be seen that latency has a negative impact on the percentage of blocks in the blockchain that are justified/finalised. In the execution of figure 1, which proposes a new block every 10 seconds, even increasing the latency to an average value of 20 (simulated) seconds results in 40% of checkpoints finalised, thus reaching consensus on all ancestors of the finalised checkpoints. This indicates that Casper is still able to achieve consensus on a significant portion of the blockchain even with severe network latency.The theoretical red line represents the percentage of blocks in blockchain that exist within the Casper defined main chain. For a given time between new blocks be proposed t, and the average latency value l, the fraction of blocks in the main chain should be $1 - \frac{t}{t+l}$.
Not only is Casper Byzantine fault tolerant, but its mechanism for slashing Byzantine validators allows for stronger recovery from Byzantine faults. As each validator's voting power is determined by their deposit value, the slashing mechanism provides incentive for validators to behave correctly, else suffer a decrease in their voting power. Coupled with Casper's rewarding of correct validators, these are key advantages in real-world implementations, particularly in Casper's native Ethereum implementation. By requiring validators to deposit a certain amount of the cryptocurrency Ether, Byzantine validators risk losing their cryptocurrency for attempting to be malicious in the network. Further adding to Casper's real-world implemenation advantages are its accountable safety and plausible liveness guarantees. Respectively, these ensure that the same correct blockchain is agreed upon between all validators (with punishment served to malicious validators), and that the blockchain is still able to grow as long as $\frac{2}{3}$ of the validators are still operating correctly.

Disadvantages

Whilst Casper has been shown to be a robust and efficient protocol for achieving consensus, it does contain less desirable features. Whilst only operating on checkpoint blocks was an advantage for efficiency, it comes as a tradeoff for wasting blocks. By not achieving consensus on every single block that is proposed in the network, some blocks added to the chain in validators are essentially useless if they do not end up in the main chain after Casper executes. This occurs in Casper due to another disadvantage of Casper, the fact that it is only a "guardian" over an existing blockchain proposal system. As Casper is only concerned with blocks once they have been added to the blockchain, the risk of blocks ending up not being used is much higher. Thus, whilst Casper is still an effective consensus protocol, the overall quality of the blockchain system is still heavily dependent on the underlying proposal system.

Future Directions

Future directions of consensus in distributed systems focuses on addressing (1) TPS, security and resource conservation within permissionless architecture. Ethereum has detailed a plan to Ethereum 2.0, with Serena security upgrades (ref required), which includes shard chains: scaling, via splitting transactions into a multi-set of randomly organized validator. eWasm: Updating the runtime to use Web Assembly Machine bindings and lastly further plans for super-quadratic or exponential sharding. (ref). With the lightning pace of innovation seen since the initial Nakomoto white paper - spawning thousands of blockchain derivatives and novel protocols and flavours, it it is probable that in 10 years, the prevalent blockchain implementation may be vastly different or otherwise purposed. For permissioned consensus (2) further layering on-top of Raft and Paxos is expected; Notably it's applications within database systems rethinkDB, tikv and etcd providing distributed key-value stores with Raft.

References

[LSP82] Leslie Lamport, Robert Shostak, and Marshall Pease. "The Byzantine Generals Problem". In: *ACM Transactions on Programming Languages and Systems (TOPLAS)* 4.3 (Dec. 1982), pp. 382–401. ISSN: 15584593. DOI: 10.1145/357172.357176.

[FLP85] Michael J Fischer, Nancy A Lynch, and Michael S Paterson. "Impossibility of Distributed Consensus with One Faulty Process". In: *Journal of the ACM (JACM)* 32.2 (1985), pp. 374–382. ISSN: 1557735X. DOI: 10.1145/3149.214121.

[CR93] Ran Canetti and Tal Rabin. "Fast asynchronous byzantine agreement with optimal resilience". In: *Proceedings of the Annual ACM Symposium on Theory of Computing*. 1993. ISBN: 0897915917. DOI: 10.1145/167088.167105.

[DN93] Cynthia Dwork and Moni Naor. "Pricing via processing or combatting junk mail". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 740 LNCS. 1993, pp. 139–147. ISBN: 9783540573401. DOI: 10.1007/3-540-48071-4{_}10.

[GM93] Juan A. Garay and Yoram Moses. "Fully polynomial byzantine agreement in t + 1 rounds". In: *Proceedings of the Annual ACM Symposium on Theory of Computing*. 1993. ISBN: 0897915917. DOI: 10.1145/167088.167101.

[MR97] Dahlia Malkhi and Michael Reiter. "Unreliable intrusion detection in distributed computations". In: *Proceedings - IEEE Computer Security Foundations Symposium*. 1997. ISBN: 0818679905. DOI: 10.1109/CSFW.1997.596799.

[Lam98] Leslie Lamport. *The Part-Time Parliament*. Tech. rep. 2. 1998, pp. 133–169.

[Cas01] Miguel Castro. “Practical Byzantine Fault Tolerance”. In: *Proceedings of the Third Symposium on Operating Systems Design OSDI ’99* February (2001), pp. 1–172. URL: `http://pmg.csail.mit.edu/papers/osdi99.pdf`.

[Lam01] Leslie Lamport. *Paxos Made Simple*. Tech. rep. 2001.

[Lam03] Leslie Lamport. “Lower bounds for asynchronous consensus”. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 2584 (2003), pp. 22–23. ISSN: 03029743. DOI: `10.1007/3-540-37795-6{\_}4`. URL: `http://www.research.microsoft.com`.

[Avi+04] Algirdas Avižienis et al. “Basic concepts and taxonomy of dependable and secure computing”. In: *IEEE Transactions on Dependable and Secure Computing* 1.1 (2004), pp. 11–33. ISSN: 15455971. DOI: `10.1109/TDSC.2004.2`.

[Lam06] Leslie Lamport. “Fast Paxos”. In: *Distributed Computing* (2006). ISSN: 01782770. DOI: `10.1007/s00446-006-0005-x`.

[CGR07] Tushar Chandra, Robert Griesemer, and Joshua Redstone. “Paxos made live: An engineering perspective”. In: *Proceedings of the Annual ACM Symposium on Principles of Distributed Computing*. Proceedings of the Annual ACM Symposium on Principles of Distributed Computing, 2007, pp. 398–407. ISBN: 1595936165. DOI: `10.1145/1281100.1281103`.

[Maz07] David Mazi. “Paxos Made Practical”. In: *Other* (2007). URL: `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.143.6093&amp;rep=rep1&amp;type=pdf`.

[KA09] Jonathan Kirsch and Yair Amir. “Paxos for system builders: An overview”. In: *ACM International Conference Proceeding Series*. Vol. 341. 2009. ISBN: 9781605582962. DOI: `10.1145/1529974.1529979`. URL: `http://www.dsn.jhu.edu..`

[Lam11] Leslie Lamport. “Byzantizing Paxos by refinement”. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. 2011. ISBN: 9783642240997. DOI: `10.1007/978-3-642-24100-0{\_}22`.

[Bab+12] Moshe Babaioff et al. “On bitcoin and red balloons”. In: *Proceedings of the ACM Conference on Electronic Commerce*. Nov. 2012, pp. 56–73. ISBN: 9781450314152. DOI: `10.1145/2229012.2229022`. URL: `http://arxiv.org/abs/1111.2626`.

[KN12] Sunny King and Scott Nadal. “PPCoin: Peer-to-Peer Crypto-Currency with Proof-of-Stake”. In: *PPCoin: Peer-to-Peer Crypto-Currency with Proof-of-Stake*. 2012.

[MPP12] Parisa Jalili Marandi, Marco Primi, and Fernando Pedone. “Multi-ring paxos”. In: *Proceedings of the International Conference on Dependable Systems and Networks*. 2012. ISBN: 9781467316248. DOI: `10.1109/DSN.2012.6263916`.

[MJ14] Andrew Miller and Jj LaViola Jr. “Anonymous Byzantine Consensus from Moderately-Hard Puzzles: A Model for Bitcoin”. In: *Anonymous Byzantine Consensus from Moderately-Hard Puzzles: A Model for Bitcoin*. 2014. URL: `https://socrates1024.s3.amazonaws.com/consensus.pdf`.

[TK14] Zied Trifa and Maher Khemakhem. “Sybil nodes as a mitigation strategy against sybil attack”. In: *Procedia Computer Science*. Vol. 32. Elsevier B.V., Jan. 2014, pp. 1135–1140. DOI: `10.1016/j.procs.2014.05.544`.

[Poe15] Andrew Poelstra. *On Stake and Consensus*. 2015. URL: `https://download.wpsoftware.net/bitcoin/old-pos.pdf..`

[VA15] Robbert Van Renesse and Deniz Altinbuken. “Paxos made moderately complex”. In: *ACM Computing Surveys* 47.3 (2015). ISSN: 15577341. DOI: `10.1145/2673577`.

[Woo+16] Doug Woos et al. “Planning for change in a formal verification of the raft consensus protocol”. In: *CPP 2016 - Proceedings of the 5th ACM SIGPLAN Conference on Certified Programs and Proofs, co-located with POPL 2016*. 2016, pp. 154–165. ISBN: 9781450341271. DOI: `10.1145/2854065.2854081`. URL: `https://raft.github.io/`.

[BG17] Vitalik Buterin and Virgil Griffith. “Casper the Friendly Finality Gadget”. In: *CoRR* abs/1710.0 (2017). URL: `http://arxiv.org/abs/1710.09437`.

[Dic17] Tom Dickman. *Bitcoin (BTC)— Crypto51*. 2017. URL: `https://www.crypto51.app/coins/BTC.html`.

[EL17] Hugues Evrard and Frédéric Lang. "Automatic distributed code generation from formal models of asynchronous processes interacting by multiway rendezvous". In: *Journal of Logical and Algebraic Methods in Programming* 88 (2017), pp. 121–153. ISSN: 23522216. DOI: `10.1016/j.jlamp.2016.09.002`. URL: `https://www.grid5000.fr`.

[And+18] Elli Androulaki et al. "Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains". In: *Proceedings of the 13th EuroSys Conference, EuroSys 2018*. Vol. 2018-Janua. 2018. ISBN: 9781450355841. DOI: `10.1145/3190508.3190538`. URL: `www.hyperledger.org`.

[BŽ18] Federico Matteo Benčić and Ivana Podnar Žarko. "Distributed Ledger Technology: Blockchain Compared to Directed Acyclic Graph". In: *Proceedings - International Conference on Distributed Computing Systems*. Vol. 2018-July. Apr. 2018, pp. 1569–1570. ISBN: 9781538668719. DOI: `10.1109/ICDCS.2018.00171`. URL: `http://arxiv.org/abs/1804.10013`.

[HYP18] HYPERLEDGER. *Hyperledger Iroha – Hyperledger*. 2018. URL: `https://www.hyperledger.org/use/iroha?fbclid=IwAR1iHtbEIObr1-vpuRoQI-pHZtE1NqO8ZVz6IaqkziB7PZdmVQRbSOSqr8s`.

[Lem18] Colin Lemahieu. "Nano: A Feeless Distributed Cryptocurrency Network". In: *White paper* (2018), p. 8.

[Pop18] Serguei Popov. "IOTA Whitepaper v1.4.3". In: *New Yorker* (2018). ISSN: 0028-792X.

[Pug18] Brian Pugh. *Stress Testing The RaiBlocks Network: Part II – Brian Pugh – Medium*. 2018. URL: `https://medium.com/@bnp117/stress-testing-the-raiblocks-network-part-ii-def83653b21f`.

[SSG18] Jagdeep Sidhu, Eliot Scott, and Alexander Gabriel. *Z-DAG: An interactive DAG protocol for real-time crypto payments with Nakamoto consensus security parameters*. Tech. rep. 2018. URL: `https://syscoin.org/zdag_syscoin_whitepaper.pdf`.

[But+19] Vitalik Buterin et al. "Incentives in ethereum's hybrid casper protocol". In: *ICBC 2019 - IEEE International Conference on Blockchain and Cryptocurrency*. 2019. ISBN: 9781728113289. DOI: `10.1109/BLOC.2019.8751241`.

[Cañ19] Alejandro Esquivias Cañadas. *A Comprehensive Survey on Blockchain's Technology*. Tech. rep. 2019.

[DPS19] Phil Daian, Rafael Pass, and Elaine Shi. "Snow White: Robustly Reconfigurable Consensus and Applications to Provably Secure Proof of Stake". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 11598 LNCS. 2019, pp. 23–41. ISBN: 9783030321000. DOI: `10.1007/978-3-030-32101-7{\_}2`.

[Nak19] Satoshi Nakaoto. "Bitcoin: A Peer-to-Peer Electronic Cash System". In: *SSRN Electronic Journal* (2019). DOI: `10.2139/ssrn.3440802`. URL: `www.bitcoin.org`.

[OO19] Diego Ongaro and John Ousterhout. "In search of an understandable consensus algorithm". In: *Proceedings of the 2014 USENIX Annual Technical Conference, USENIX ATC 2014*. 2019, pp. 305–319. ISBN: 9781931971102.

[SKG19] Christian Stoll, Lena Klaaßen, and Ulrich Gallersdörfer. "The Carbon Footprint of Bitcoin". In: *Joule* 3.7 (July 2019), pp. 1647–1661. ISSN: 25424351. DOI: `10.1016/j.joule.2019.05.012`.

[Cao+20] Bin Cao et al. "Performance analysis and comparison of PoW, PoS and DAG based blockchains". In: *Digital Communications and Networks* (2020). ISSN: 23528648. DOI: `10.1016/j.dcan.2019.12.001`.

[HM20] Heidi Howard and Richard Mortier. "Paxos vs Raft: Have we reached consensus on distributed consensus?" In: *arXiv* (2020). DOI: `10.1145/3380787.3393681`. URL: `http://arxiv.org/abs/2004.05074%0Ahttp://dx.doi.org/10.1145/3380787.3393681`.

[Ter20] Benjamin Terner. *Permissionless Consensus in the Resource Model*. 2020. URL: `https://eprint.iacr.org/2020/355`.

[Xia+20] Yang Xiao et al. "A Survey of Distributed Consensus Protocols for Blockchain Networks". In: *IEEE Communications Surveys & Tutorials* (2020), pp. 1–1. DOI: `10.1109/comst.2020.2969706`.