# MWPoW+: a strong consensus protocol for intra-shard consensus in blockchain sharding

YIBIN XU, University of Copenhagen, Denmark
JIANHUA SHAO, Cardiff University, UK
TIJS SLAATS and BORIS DÜDDER, University of Copenhagen, Denmark

Blockchain sharding splits a blockchain into several shards where consensus is reached at the shard level rather than over the entire blockchain. It improves transaction throughput and reduces the computational resources required of individual nodes. But a derivation of trustworthy consensus within a shard becomes an issue as the longest-chain based mechanisms used in conventional blockchains can no longer be used. Instead, a vote-based consensus mechanism must be employed. However, existing vote-based Byzantine false tolerance consensus protocols do not offer sufficient security guarantees for sharded blockchains. First, when used to support consensus where only one block is allowed at a time (binary consensus), these protocols are susceptible to progress-hindering attacks, i.e., unable to reach a consensus. Second, when used to support a stronger type of consensus where multiple concurrent blocks are allowed (strong consensus), their tolerance of adversary nodes is low. This paper proposes a new consensus protocol to address all these issues. We call the new protocol MWPoW+ as its basic framework is based on the existing Multiple Winner Proof of Work (MWPoW) protocol but includes new mechanisms to address the issues mentioned above. MWPoW+ is a vote-based protocol for strong consensus, asynchronous in consensus derivation but synchronous in communication. We prove that it can tolerate up to $f < n/2$ adversary nodes in a $n$-node system as if using a binary consensus protocol, and does not suffer from progress-hindering attacks.

Additional Key Words and Phrases: Strong consensus, Blockchain Sharding, Byzantine Fault Protocol, BFT, MWPoW, PBFT, Asynchronous consensus protocol, Blockchain, Distributed Ledger

**ACM Reference Format:**
Yibin Xu, Jianhua Shao, Tijs Slaats, and Boris Düdder. 2023. MWPoW+: a strong consensus protocol for intra-shard consensus in blockchain sharding. *ACM Trans. Internet Technol.* 37, 4, Article 1 (August 2023), 28 pages. https://doi.org/0

## 1 INTRODUCTION

Blockchain sharding, which improves transaction throughput of a blockchain system by hosting parallel shards and reduces workload for individual nodes by requiring them only to work for a single shard, has attracted much recent attention from the blockchain research community. Over the years, blockchain sharding solutions have progressed from early prototypes [6, 17] with weak security features to models of different security levels [12, 28–31].

One of the key issues for blockchain sharding is consensus derivation inside a shard. Unlike a conventional blockchain, sharded blockchains cannot use the longest-chain based mechanisms

Authors' addresses: Yibin Xu, work@xuyibin.top,yx@di.ku.dk, University of Copenhagen, Copenhagen, Denmark, København N. 2200; Jianhua Shao, Cardiff University, Cardiff, UK; Tijs Slaats; Boris Düdder, University of Copenhagen, Copenhagen, Denmark, København N. 2200.


1533-5399/2023/8-ART1 $15.00
https://doi.org/0

that use Proof of Work (PoW) [3, 7] or Proof of Stake (PoS) [23] to propose blocks. This is because when nodes are divided into shards, their calculation power is also divided. Thus, an adversary may only have a relatively small amount of calculation power in the overall system, but this power may be enough for it to become a dominating power in a single shard to manipulate the longest chain of blocks. To the best of our knowledge, there is no method that can upper bound the amount of calculation power one can put into PoW or PoS.

Therefore, most blockchain sharding approaches use vote-based mechanisms for consensus derivation, where a leader node is elected periodically to propose a block and other nodes verify and vote on the block. These approaches consider every node having equal weight in voting and many employ the well-known asynchronous Byzantine consensus protocol PBFT [4] or its edited versions as their intra-shard consensus protocol. However, PBFT can only tolerate a third of the nodes in a shard and so allowing a total of $n/4$ nodes in an $n$-node sharded system to be adversaries. Rapidchain [31] and the approaches proposed by Xu et al. [28, 29] improve these approaches by using a synchronous intra-shard consensus protocol which allows half of the nodes in a shard acting adversarial.

However, we observe that the vote-based protocols, when used in a blockchain, may be hindered from progressing when an adversarial leader generates a faulty block or remain in silence. These represent a new type of anti-liveliness attack on consensus protocols that has not been considered before. We refer to this type of attack as **progress hindering attack** in this paper. This occurs because consensus is *binary* on a single input value (block): if at some point in time nodes cannot reach consensus on the current block, yet there is no alternative to vote on, then the nodes must wait for the leader node to be replaced and a new block to be proposed. In sharded blockchain, the problem does not only exist at the system level, but also exists at the shard level. This means that the probability for the attack to occur in a system as a whole is $s$ times more likely than in a non-sharded vote-based system where $s$ is the number of shards. It may destabilize operations like cross-shard communications and node membership adjustments.

Alternatively, we can employ a stronger type of consensus protocol, or simply *strong* consensus, where multiple blocks are allowed to enter into consensus resolution concurrently. It has been proven, however, that asynchronous and synchronous protocols for this type of consensus have security lower bounds of $n > (iv+1)f$ and $n > \max(3, iv)f$, respectively, where $f$ is the number of adversary nodes and $iv$ is the number of different initial values in the system, $iv > 2$ [8, 19]. This is lower than binary consensus, because when multiple initial values are allowed, the most supported value does not necessarily imply the majority support anymore. For example, if a synchronous communication system uses a number of sensors to determine the temperature around a machine, any temperature reported by the sensors is acceptable, and the most supported one among the sensors will be accepted. As such, in an extreme case, an adversary only needs to take control of $\frac{n}{\max(3,iv)} + 1$ sensors in an $n > \max(3, iv)f$ system to make it most supported. In blockchains, however, whether a block is acceptable (e,g. whether containing invalid transactions) is checked before a vote is cast, and any honest node would not support or accept a faulty block even if it has gained $\frac{n}{\max(3,iv)} + 1$ support in a synchronous communication system. We propose the validated strong consensus for the blockchain using this feature. We show in this paper that combing this feature with pre-defined rules for voting (the honest nodes always support the most supported block to their knowledge), it allows the honest nodes to gradually shifting to one block, a majority consensus reached and the block accepted. In this way, the validated strong consensus protocol can achieve the same security bound as a binary consensus protocol in blockchains.

Furthermore, vote-based blockchains are still less secure than conventional blockchains. In a longest-chain blockchain, nodes can have different weights (calculation power or stake) to create a consensus, and as such, an adversary needs to control half of the overall weights in order to manipulate the system ($p/2$ security level where $p$ is the overall weight). In vote-based blockchin, each vote requires only a threshold weight. Thus, it is only under the assumption that honest participants create as many nodes as they should then a $n/2$ security level system reaches the $p/2$ security level.

In this paper, we address the three issues identified above: (1) progress-hindering attack for vote-based blockchain. We solve it by employing the consensus protocol that allows multiple initial values. (2) inability to reach an $n/2$ security level in an $n$-node system (or $m/2$ for a sharded environment where $m$ is the number of nodes inside a shard) for vote-based consensus protocols when allowing multiple initial values. We solve it by designing a protocol that uses the validated strong consensus. (3) inability to reach an $p/2$ security level in a $p$ calculation power system for vote-based consensus protocol. We solve it by weighting the vote of the nodes using their declared calculation power. We call the new protocol MWPoW+ as its basic framework is based on the existing Multiple Winner Proof of Work (MWPoW) protocol [27], but includes new mechanisms to address the issues identified above. MWPoW+ is a vote-based protocol for strong consensus. It is asynchronous in consensus derivation but retains all the merits of a binary synchronous consensus protocol by maintaining synchronous communication. More specifically, we show that:

- MWPoW+ allows participants to vote on one block out of multiple concurrent ones (validated strong consensus), while other vote-based consensus protocols used for blockchains only allow one block at a time (binary consensus).
- MWPoW+ can work in an open-membership setting (permission-less). To achieve this, a membership list is maintained for nodes, and the nodes must declare (register) an amount of calculation power when they join the system, i.e. when they are added to the list.
- MWPoW+ is asynchronous in consensus derivation, but synchronous in communication. That is, the block interval is controlled by adjusting PoW difficulty for blocks, but a specific amount of calculation power registered must be used to vote for a block in order for it to be accepted eventually. Each node should vote in every block height, and every vote carries at least the amount of calculation power it has registered and the vote is weighted by that amount.
- MWPoW+ can achieve an $n/2$ or $p/2$ security level in a non-sharded environment and can provide an $m/2$ security level in a sharded blockchain where the size of a shard is $m$.
- By employing MWPoW+ as an intra-shard consensus protocol, blockchain sharding approaches can enjoy the benefit of the longest-chain based mechanisms in consensus derivation, thereby addressing the progress-hindering problem, yet not jeopardising the security of sharded blockchains.

It is worth noting that while MWPoW and MWPoW+ share a basic framework as a protocol, there are significant differences between them. MWPoW was initially designed to strengthen blockchain decentralization by increasing reward probability for resource-constrained nodes in strength competition games (mining). MWPoW+, on the other hand, is designed primarily to be used for intra-shard consensus in blockchain sharding. More specifically, we have incorporated two key elements into MWPoW+. We have introduced a vote chain into the system, which links a node's vote in one round to its vote in the preceding round. This allows the exact vote history of a node to be determined and synchronized by other nodes in the system. We have also introduced a vote array into the system, which helps nodes to derive other nodes' vote chains. These new mechanisms enable nodes to reason if a node suggesting a different view has sufficient ground to support that view, thereby filtering out adversary nodes supporting a faulty view. As we will discuss in detail in Section 4 and prove in Section 5, it is these new mechanisms that have allowed our new protocol to take advantage of the best features from both synchronous and strong consensus protocols.

The rest of the paper is organised as follows. In Section 2, we describe how some classical synchronous binary and strong consensus protocols work and compare their differences. In Section 3, we introduce the progress-hindering attack and formulate the likelihood of its happening. In Section 4, we first give a short description of MWPoW+ and how nodes reach a consensus, and then describe its working procedure in detail. In Section 5, we explain and prove why MWPoW+, as an asynchronous consensus protocol, can achieve an $n/2$ security level in an $n$-node system, and we formally analyse its security. In Section 6, we calculate time complexity for MWPoW+ and in Section 7, we discuss how MWPoW+ can be used to vastly improve the stability and security of blockchain sharding. In Section 8, we report experimental results for MWPoW+ used in a sharded system and compare it with other sharding systems. We conclude the paper in Section 9.

## 2 BACKGROUND AND RELATED WORK

In this section, we first give some background necessary for our discussion in this paper and then analyse two existing binary consensus protocols and one strong consensus protocol that are closely related to our work.

### 2.1 Synchronous vs Asynchronous Protocols

The difference between asynchronous [1, 4, 5, 13, 22] and synchronous consensus protocols [11, 21] has been discussed in [16], and the key observations are:

(1) Synchronous protocols require synchronization with a global clock shared among the nodes. They are round-based and rely critically on a network that guarantees the delivery of messages within a pre-set time-bound. This type of protocol can tolerate up to $f < n/2$ adversary nodes in an $n$-node system, but is less reliable when maintaining the time-bound.
(2) Asynchronous protocols are more reliable, and they do not assume the same as a synchronous one does. In particular, this type of protocol can achieve Byzantine agreement despite arbitrary (but finite) message delays. But asynchronous protocols can tolerate at most $f < n/3$ adversary nodes in an $n$-node system.

While asynchronous Byzantine consensus protocols, such as the well-known PBFT protocol, are often used for blockchain sharding, we argue that this is not necessary and a synchronous protocol should be considered instead for the following reasons:

(1) Blockchain, by nature, is a synchronous arrangement. In a traditional mining mechanism, the longest chain (mainchain) of valid blocks is considered to be the consensus, and the length difference between the mainchain and the longest fork chain is required to be greater than a predefined number (time-bound). This is to ensure that the mainchain is unlikely to change after time-bound, and the consensus reached at the time-bound is final. Since this effectively requires nodes to synchronise with the mainchain within time-bound, a synchronous protocol should be considered as a natural candidate.
(2) Asynchronous protocols are implemented in a vote-based blockchain system where nodes can vote for (support) or against (reject) a block at every block height. The block height moves forward if the support votes reach a threshold number. Alternatively, the block is replaced with a new block when the reject votes reach the threshold. There is no time-bound for either consensus to be reached. However, lacking a time-bound for consensus agreement does no good to blockchain sharding, because blockchain sharding requires every shard to progress within a similar time period in order for the system to deal with operations like cross-shard transactions [2, 25, 26]. Moreover, it is unrealistic to assume that no time-bound

would be needed in the progress of block height. So, even if it is asynchronous at the shard level, it is still synchronous at the system level and the non-deterministic problem can still exist at the system level with an asynchronous protocol.

(3) Despite the fact that synchronous consensus protocols can suffer the non-deterministic and fail-stop problem when an update happens at the time-bound, a synchronous consensus protocol can achieve a $f < \lceil n/2 \rceil$ security level in comparison to $f < \lceil n/3 \rceil$ for an asynchronous one. A sharded blockchain system would benefit from the $f < \lceil n/2 \rceil$ security level as it would allow the system to split more shards, thereby allowing better performance [28].

## 2.2 Binary vs Strong Consensus Protocol

In consensus derivation, $n$ participants attempt to reach an agreement on one value from a set $D$ of candidate values. Note that $f$ of these participants can be adversaries. Each participant $p_i$ starts with an initial value $v_i \in D$, and a binary consensus protocol will ensure

- *Termination*: All honest participants decide on one value from $D$.
- *Agreement*: If two honest participants decide on two values $v$ and $w$, then $v = w$.
- *Binary Validity:* If all honest participants have the same initial value $v$, then they all decide on $v$.

When using a strong consensus protocol, the Validity requirement needs to be strengthened to

- *Strong Validity:* If an honest participant decides on $v$, then $v$ is the initial value of some honest participants.

For blockchains, we further adjust the Validity requirement to

- *Validated Strong Validity:* If an honest participant decides on a value $v$, then $v$ is a *valid* one and is an initial value of some participants.

The security lower bound for asynchronous and synchronous strong consensus protocols is $n > (iv + 1)f$ and $n > \max(3, iv)f$, respectively, where $iv > 2$ is the maximum number of initial values allowed [8, 19]. This is lower than a binary consensus protocol. However, we will show in this paper that this security can be increased to the same level as a binary consensus protocol by enforcing the Validated Strong Validity to filter out faulty blocks. This is in contrast to existing strong consensus protocols that cannot filter out adversary inputs in consensus derivation.

Strong consensus protocols are preferable to binary ones for blockchains because a correct block for each epoch is the collection of updates observed by the node that proposed it. We can have different correct blocks and they are all acceptable.

## 2.3 Archetypal consensus protocols

We now briefly describe two classical binary consensus protocols, one synchronous [4] and one asynchronous [21], to illustrate how each type works and its main properties. We also briefly describe a strong consensus protocol [8]. Their security proofs are reproduced here from [4], [21] and [8] for ease of reference.

*2.3.1 Practical Byzantine Fault Tolerance (PBFT) Protocol.* PBFT [4] is designed to provide a Byzantine state machine replication [14, 24] based on an asynchronous or partially-synchronous assumption that is practical in real-life. Nodes in a PBFT system are ordered sequentially with one node acting as the leader node (primary), and other nodes are referred to as the backup nodes (secondary) that synchronize with the leader node. Nodes take turns to be the primary. All nodes participating in the system synchronize with the primary before a request from a client is processed. They then process the request and a consensus is reached or deemed to be reached when $\lfloor 2n/3 + 1 \rfloor$

nodes generate the same verdict, where $n$ is the number of nodes in the system. A PBFT consensus derivation round is split into five phases:

(1) **Request phase**: The client sends a request for consensus to the leader node.
(2) **Pre-prepare stage**: The leader node broadcasts a notice to all nodes that there is a request to process and the leader node's current state.
(3) **Prepare stage**: If a node agrees with the state sent by the leader node, it will notify all other nodes that it will process the request based on that state.
(4) **Commit stage**: If a node receives at least $\lfloor 2n/3+1 \rfloor$ notification from other nodes agreeing to start working on the request, it will process the request and broadcast its verdict to other nodes.
(5) **Reply stage**: When a node receives at least $\lfloor 2n/3+1 \rfloor$ results from other nodes that give the same verdict, it will send that verdict to the client. The client accepts the result if it is confirmed by at least $\lfloor 2n/3+1 \rfloor$ nodes.

Note that the sender of every piece of information must use a digital signature.

Lemma 2.1. *Let $Rp$ and $Rq$ be two consensuses for the same request from a client in a PBFT system and $f$ be the number of adversary nodes in the system. When $f < n/3$, we have $Rp = Rq$ (the content of $Rp$ and $Rq$ is the same).*

Proof. In the worst case, we have a maximum of $f$ adversary nodes and remaining $f+f+1$ honest nodes in an $n$-node PBFT system. Suppose that we have two honest nodes $h_1$ and $h_2$, both having accepted $Rp$ and $Rq$ in the commit stage of PBFT procedure. Since at least $\lfloor 2n/3+1 \rfloor$ votes are required for a consensus to be accepted and $n = f+f+f+1$, $Rp$ and $Rq$ must each have at least $2f+1$ votes. So there must exist $f+1$ nodes who have voted for both $Rp$ and $Rq$. This is not possible as the adversary only has $f$ nodes and honest nodes will not send different votes to different nodes. Therefore, we must have $Rp = Rq$. That is, even in the worse case, there can only be one consensus when $f < n/3$. □

*2.3.2 Practical Synchronous Byzantine (PSB) Consensus.* PSB [21] is designed to provide a Byzantine state machine replication based on a synchronous assumption that is practical in real-life. Nodes in a PSB system are organised as primary and secondary nodes like in a PBFT system, but the voting process is different. Before processing a request, all nodes vote and reach a consensus on their current state. The primary updates its state to the consensus state, processes the request based on the consensus state and broadcasts the result to all nodes. All nodes then vote on the result generated by the primary, and the result is accepted when $\lfloor n/2+1 \rfloor$ nodes have voted for it. A PSB consensus derivation round is split into six phases:

(1) **Request phase**: The client sends a request for consensus to the leader node.
(2) **Status stage**: The leader node broadcasts a notice to all nodes that there is a request to process, and all nodes send their current states to the leader node.
(3) **Prepare stage**: The leader node generates a result for the request based on the consensus state, i.e. the state that has at least $\lfloor n/2+1 \rfloor$ nodes in it. The leader node then broadcasts this result to all nodes.
(4) **Commit stage**: If a node agrees with the result sent by the leader node, it will broadcast a commit request for the result to the system. Note that if the leader node had sent different results to different nodes, it would be detected at this stage. Note also that the nodes are able to determine if a commit request is based on the result sent by the leader. If an adversary generates an invalid commit request which is not based on the result sent by the leader, the leader node will not be considered as an adversary.

(5) **Notification stage**: A node notifies others (including the client) when it has received at least $\lfloor n/2+1 \rfloor$ commit requests for the result, if it has received no other results from the leader node.
(6) **Reply stage**: If the client receives at least $\lfloor n/2+1 \rfloor$ notifications within the time-bound, it knows that the consensus has been reached.

Note that the sender of every piece of information is required to use a digital signature.

Lemma 2.2. *Let $Rp$ and $Rq$ be two consensuses for the same request from a client during a single time-bound in a PSB system and $f$ be the number of adversary nodes in the system. When $f < n/2$, we have $Rp = Rq$.*

Proof. In the worst case, we have a maximum of $f$ adversary nodes and remaining $f+1$ honest nodes in an $n$-node PSB system. Suppose that we have two honest nodes $h_1$ and $h_2$, both having accepted $Rp$ and $Rq$, respectively, after the notification stage of a single time-bound. Since at least $\lfloor n/2+1 \rfloor$ votes are required for a consensus to be accepted and $n = f + f + 1$, $Rp$ and $Rq$ must each have at least $f+1$ notifications after the notification stage. However, since an honest node will only send the notification if it knows only one result that has at least $f+1$ commit requests, there must be one honest node that has sent the notifications for both $Rp$ and $Rq$. This is not possible as if an honest node has two results at the commit stage, it will not send any notification at all. On the other hand, if the honest node does not know one of the two results, it cannot send the notification for a result that it does not know. Therefore, we must have $Rp = Rq$, even in the worse case, as long as $f < n/2$. □

*2.3.3 Strong Consensus (SC) protocol for synchronous networks.* SC runs $n$ Byzantine broadcast protocols [9] concurrently, one instance for each node acting as the creator and sender of its input value to the protocol. When the execution of the protocol is completed, the most frequent input value from all the received input values will be decided. If there is a tie, then the lowest of all values will be decided.

Lemma 2.3. *In a synchronous network, if $n > \max(3, iv)f$, where $f$ is the number of adversary nodes, then a Strong Consensus protocol achieves unconditional strong consensus in $f+1$ rounds.*

Proof. First, observe that since $n > 3f$ all the broadcast protocols invoked will work correctly, and all the honest participants will decide on the same value for each invocation. Therefore, the protocol satisfies the Agreement condition. Furthermore, since $n > iv \times f$, there must be a value $v \in D$ that has been received more than $f$ times by each honest participant. Hence, the decision value must be tallied more than $f$ times and is an input value from an honest participant. This gives Strong Validity. Termination in $f+1$ rounds follows directly from the round optimality of the protocol given in [9]. □

In this article, we explore how to increase the security lower bound for a strong consensus protocol using the validated strong validity. We consider the following: (1) in the blockchain consensus, blocks are interconnected and are of inheritance relationship. Therefore, to vote on a block implies recognition of all blocks beforehand of it. A consensus resolution in a new epoch can be considered as a re-vote on all the previous epochs. (2) a blockchain runs non-stop, therefore we can separate the termination of a consensus derivation process from the termination of the execution of a consensus protocol. That is, a block does not need to be accepted before new blocks can build on top of it. A block is accepted when the majority of nodes have shifted to the branch of blocks steamed from it, provided there are mechanisms ensuring that no other branches can be the majority supported in the future.

We use a vote chain and a vote array (to be discussed in Sections 4 and 5) to force an adversary to either stay silent or vote correctly (follow the most supported branch to their knowledge), and we establish acceptance criteria for blocks that ensures once a block is accepted, it is final and unchangeable. We show that while strong consensus protocols need to use rounds of votings for the nodes to reach a consensus on the input value in general, a blockchain is able to improve this process by combining the consensus derivation of multiple epochs at the same time by exploiting the above mentioned properties. Note that it is distinctively different from repeating a binary consensus protocol, where the process of blockchain is halted until an honest leader proposed a correct block. In our approach, the new round of voting is also the voting for the preceding block heights. Thus, the process of the blockchain is not interrupted when a faulty block is proposed.

Our design of MWPoW+ follows from the above observations. The new protocol optimizes the time complexity of a blockchain system implementing a strong consensus protocol overall, while achieving the same security level as a synchronous binary consensus protocol.

## 3 THE PROGRESS-HINDERING ATTACK

In this section, we discuss a new type of attack on consensus protocols that has not been considered before. We call this type of attack *progress hindering attack*. We will first briefly outline a consensus protocol that we proposed specifically for blockchain sharding [29], and then analyse how progress hindering attack will happen when such a protocol is used.

### 3.1 Consensus using a committee shard

In our previous work [29], we proposed a blockchain sharding solution that, by using a synchronous protocol for intra-shard consensus, allows up to $\lfloor (m-1)/2 \rfloor$ adversary nodes in a shard of size $m$ and $\lfloor (n-1)/2 \rfloor$ adversary nodes in an $n$-node system overall. In the proposed solution, there is a committee shard that records information such as the public identity key of every node in the system, the shard ID of the shard in which they reside, and the hash of the block header of the latest block that has been confirmed through the consensus protocol in every shard. Note that the number of shards is dynamically adjusted in this design to overcome a halting situation that can be caused by the distribution of adversary population across the shards, hence the information recorded in the committee shard is updated dynamically too.

The consensus procedure inside a shard in that solution is divided into the following steps:

(1) **Leader node election**: Let $Blockhash[i][BH]$ be the block header hash of shard $i$ in block height $BH$. $L_i$ is the index number of the leader node of shard $i$ in $BH+1$

$$L_i = hash(Blockhash[i-5][BH], \ldots, Blockhash[i+5][BH]) \mod m$$

Specially,

$$Blockhash[j][BH] = Blockhash[abs(s-j)][BH], \quad j \in [-4, 0] \cup [s+1, s+4]$$

If shard $i$ did not reach a consensus on block height $BH$ then

$$Blockhash[i][BH] = Blockhash[i][BH-1]$$

(2) **Intra-shard consensus**: The leader proposes a block and other nodes do two rounds of voting following the same process given in Section 2.3.2. The nodes need to generate an amount of PoW above a threshold when voting in order to prevent a Sybil Attack.

(3) **Global synchronous:** A *detect*-then-*verify* mechanism is used for global synchronisation. The nodes of a shard will inform the committee shard when more than a certain number of nodes have approved a block of this shard. Then the nodes of the committee shard will download the block header of this block. If a node in this shard is opposing this block, it will

inform the committee shard. When a conflict is *detected*, the honest nodes in this shard will send the data associated with the consensus resolution process to other shards for *verification*. Other shards will accept the correct block after verification. If there is no conflict reported within a given time frame, then the committee shard will accept this block as the new accepted block for the corresponding shard.

Note that in so doing, a node is required to synchronise with the block generated by the committee shard and the blocks of the shard in which it resides. It is also worth noting that because the sharding protocol and the intra-shard consensus protocol are run synchronously in this setting, honest nodes will always make contact with each other within a time-bound, and thus there will be no consistency issues in terms of which block is finally accepted after the communication time-bound. As a result, forking attacks [15], in which an untrusted node can provide various views to multiple non-communicating nodes, will be dealt with in a timely manner in this case and will not compromise security.

### 3.2 Adversary and threat model

We now consider one specific vulnerability associated with the consensus protocol outlined in Section 3.1.

As can be seen, in this type of consensus protocol, a shard must select a leader periodically to propose a block for other members of the shard to verify. Since we may have adversaries in a shard, we must consider what happens when an adversary is elected as the leader. By definition, an adversarial will not propose a correct block for other nodes to verify. Instead, it can generate a faulty block, provide different blocks to different nodes or stay silent. Clearly, when this happens, no useful work could be done during the current round, and the nodes in the shard will simply abandon it and elect a new leader for the next round. We call this progress-hindering attack, as the system is prevented from making progress when this happens. Note that in a longest-chain blockchain, every node can propose a block for an epoch. Thus, there is no need to wait for the next round for a new leader node to propose a new block.

How serious is this a threat to the operation of a sharded blockchain? In the following, we calculate the chance for an adversary node to be elected as a leader, hence the chance of a progress-hindering attack happening. Let there be $f$ $(0 \leq f < \lceil m/2 \rceil)$ adversary nodes in a shard of $m$ nodes. The probability of having an honest leader for $t$ rounds and an adversarial leader for the remaining rounds in $h$ continuous number of epochs is

$$P = \left(\frac{f}{m}\right)^{h-t} \times \left(\frac{m-f}{m}\right)^{t} \tag{1}$$

Figure 1 shows an example of this probability with different numbers of adversary nodes.

As can be seen, when there are close to $50\%$ adversary nodes in a shard, a progress hindering attack would almost certainly occur. Even with only $10\%$ of the nodes being adversaries, there is still close to $50\%$ chance that there will be an attack every 6 epochs. This could seriously affect the smooth or even normal operation of a sharded blockchain. Therefore, in order to make a vote-base blockchain practical, especially in open-membership settings, there needs to be a way to deal with the progress hindering attack.

## 4 THE MWPOW+ PROTOCOL

Based on the arguments so far, we propose a new asynchronous consensus protocol MWPoW+. The basic framework of this new protocol is similar to that of the existing Multiple Winner Proof of Work (MWPoW) protocol [27] (hence the name MWPoW+), but the two differ significantly in terms of functionality. The new protocol can offer the same level of security as PSB and can avoid

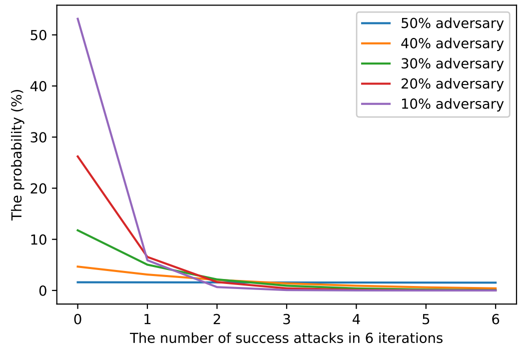


Fig. 1. The probability for the success attacks

progress-hindering attacks. In this section, we first give an overview of MWPoW+ in Section 4.2 and then focus on the operational details in Section 4.3.

### 4.1 Calculation Power

The calculation power is defined as, in a given interval, the amount of PoW difficulty that a machine can achieve. PoW difficulty is a measure of how difficult to generate a PoW:

$$\text{Difficulty} = \frac{\text{difficulty\_target}}{\text{current\_target}} \tag{2}$$

where $\text{difficulty\_target}$ is a pre-defined 256-bit constant and $\text{current\_target}$ is any 256-bit number. The PoW process runs by the machine generating a random *Nonce* to a given *String* to change its hash value. The PoW difficulty of this *string* is calculated by using its hash value as *current_target*. A PoW is generated when a *String* is generated with a PoW difficulty fulfilling a given value.

### 4.2 Summary description

The proposed MWPoW+ protocol allows up to $f \leq \lfloor (n-1)/2 \rfloor$ adversary nodes in a $n$ node blockchain system and the block height moves within a time-bound. The key features of the proposed protocol are:

- MWPoW+ is a vote-based consensus protocol but no leader node is elected. Nodes use a Proof-of-Work mechanism to propose a block. The property that blocks are inherited and interconnected in a blockchain is used to prevent progress-hindering attacks.
- Just like in the Nakamoto blockchain, it assumes that the majority of participating calculation power is honest.
- It assumes honest nodes would only vote for one block candidate at a block height while there can be multiple block candidates.
- The block height can move on regardless if a consensus has been reached or not. Therefore, when MWPoW+ is implemented in a sharded system, the progress of block height can be synchronous for every shard, despite the progress for consensus is asynchronous among the shards.

The working procedure of MWPoW+ is divided into the following steps:

(1) **Power registration:** When joining the system, nodes are required to declare an amount of calculation power (in PoW difficulty form) that they will put into competition in every block interval, with an attached PoW to prove that.

(2) **Progress of block height:** There are two PoW difficulties for a block of every block height: Entrance Difficulty and Acceptance Difficulty. The Entrance Difficulty is for proposing a block and the Acceptance difficulty is for the system to move on to the next block height. The two PoW difficulties are adjusted after every block interval. Every node is in charge of a different interval in $[0, 2^{256}]$ for the Nonce of the block. By adjusting the Nonce, the hash of the block can be adjusted, then the PoW difficulty of the block is adjusted.
At the beginning of every block interval, nodes create their own blocks. When a node finds a Nonce in its Nonce interval for its block that makes the hash of the block fulfil the Entrance difficulty, it broadcasts the block and the Nonce. Every node then tries to find a Nonce of the Acceptance difficulty in their Nonce interval for the block that reaches the Entrance difficulty first. When such a Nonce is found and broadcast, the system moves to the next block height.

(3) **Voting:** A node should vote four times at each block height by broadcasting its Nonces for a block. Each Nonce for the block should make the PoW difficulty of this block fulfill $25\%$ of the node's registered power. The Nonce is embedded in a data structure called *Share*. Note that a Nonce is only valid for one *Share*. Table 1 shows the structure of a *Share*. Because every *Share* embeds the hash of the preceding *Share*, every node is linked to the preceding *Share* and therefore there is a **vote chain** for every voter. There are four vote chain heights within every block height.

Table 1. The structure of a *Share* at the vote chain height $i$

| Name | Description |
|---|---|
| Block candidate hash | The hash of the block candidate which a voter votes for. |
| Last vote hash | The hash of the last vote that the voter sent. |
| Vote Array | An array of the *Share* hashes at the vote chain height $i-1$ from the nodes which together registered at least $\lfloor n/2 \rfloor + 1$ of the calculation power. |
| Not support | A Boolean flag that is always FALSE if not changed. See section 4.3.4 for details. |
| Nonce | A 256-bit integer for adjusting the PoW difficulty of the block. |

(4) **Vote counting:** Instead of counting the number of *Shares* for a block in $h$ block height, we calculate the accumulated PoW difficulty for this block as well as the whole branch of blocks stemming from it (from $h$ block height to the latest block height $H$). Note that if the PoW difficulty is larger than $25\%$ of the voter's registered power, only $25\%$ of the voter's registered power will be counted. Let $Rp$ be a block candidate of the block height $h$. Let $W_i(Rp)$ be the number of PoW difficulties at the block height $i \geq h$ which voted for the descendent block candidates of $Rp$ at the block height $i$ (if $i = h$ then for the $Rp$ itself). The accumulation of PoW difficulties for $Rp$ is

$$P(Rq) = \sum_{i=h}^{H} W_i(Rq) \tag{3}$$

where $H$ is the latest block height.

(5) **Vote discounting:** According to the vote chain of a node, if this node voted for multiple block candidates at $i$ block height, the *Shares* in this vote chain for the block candidates from $i$ to $H$ block heights are discounted in Equation 3. If a node has multiple vote chains (due to sending multiple *Shares* with the same *Last vote hash*), we only count the PoW difficulties of the longest vote chain. If the vote chains are of equal length, we only count the PoW difficulties before the forking. According to a node's vote chain, if this node has not voted four times in the block height $i$, its *Shares* for the block height $i+1$ will not be counted. Figure 2 shows an example of the vote chain and the vote counting and discounting.

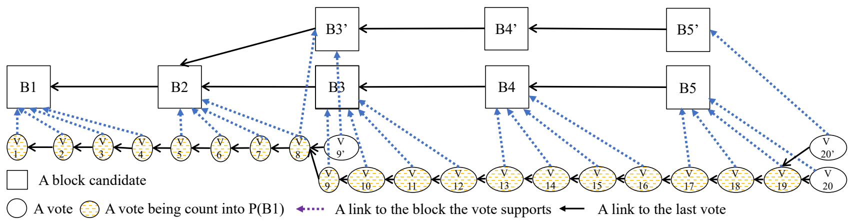


Fig. 2. An example of a vote chain for a node $V$. As can be seen from the figure, there are two links for a vote: a link to the block candidate that it supports and a link to the preceding vote. $V9$ in white is discounted because it is not in the mainchain of $V$'s vote chain. $V20$ and $V20'$ are discounted for $P(B1)$ because of the rule that if there are two vote chains of the same length, we only count the votes before forking. $V20'$ is discounted too because it supports a different block candidate from $V19$, which contradicts the rule that a node can only vote for one block candidate per consensus chain height.

(6) **Consensus reaching:** A block *Alice* at the block height $h$ is accepted when the following two conditions are met simultaneously.

(a)

$$P(Alice) > P(Gary) + \sum_{i=h}^{H} P(U_i) \tag{4}$$

where $P(Alice)$ is the accumulation of the PoW difficulties for *Alice* and its descendant block candidates. $H$ is the latest block height. *Gary* is the second most supported block candidate at the block height $h$. $P(U_i) = CP[i] - VP_i$, where $VP_i$ is the accumulation of the PoW difficulties of the votes in the block height $i$ and $CP[i]$ is the overall registered power at the block height $i$. $P(U_i)$ stands for the PoW difficulties of the nodes which have not yet voted in the block height $i$.

(b)

$$P(Alice) - P(Gary) - \sum_{i=h}^{H} P(U_i) > \frac{1}{4} \times \sum_{i=h}^{H} CP[i] \tag{5}$$

(7) **Adversary eliminating:** A graph can be built over the Vote Array in a *share* sent from any node $q$. The vote chains of all the nodes can be derived from this graph, and these vote chains are the vote chains from the perspective of the sender node $q$ of this *share*. Suppose that we have two block candidates *Alice* and *Gary* at the $h$ block height, and they are both correct (acceptable), and $P(Alice)$ and $P(Gary)$ from the perspective of node $q$ have been determined, and we refer to them as $P_q(Alice)$ and $P_q(Gary)$ respectively. Then node $q$ in the branch of *Gary* is an adversary if

$$P_q(Alice) - P_q(Gary) - \sum_{i=h}^{H} P_q(U_i) > \frac{CP[H]}{2} \tag{6}$$

The future votes from the node $q$ should not be counted in Equation 3 after it has been considered as an adversary.

(8) **Membership adjustment:** Every block records the *Shares* of the blocks at the last block height. The membership of a node is cancelled from the branch of this block onwards provided that the block does not embed at least two of the node's *Shares*.

Note that the sender of every information must use a digital signature. We now give an example of consensus derivation using this protocol. Figure 3 shows how a block is accepted, where $D$ stands for the PoW difficulty and $SR$ stands for the Support Rate (the percentage of PoWs for a block and its descendant blocks). In $(a)$, when blocks $A$, $B$ and $C$ are announced, none of them get a support rate over $50\%$, so we cannot determine which block to accept. In $(b)$, there are successive blocks of $A$, $B$ and $C$, and the support rates of blocks $A$, $B$, $C$ have changed. Blocks $C$ and $D$ are eventually accepted because they have a support rate over $50\%$. Meanwhile, this support rate is greater than that of either block of the same block height plus $25\%$ of the registered power.

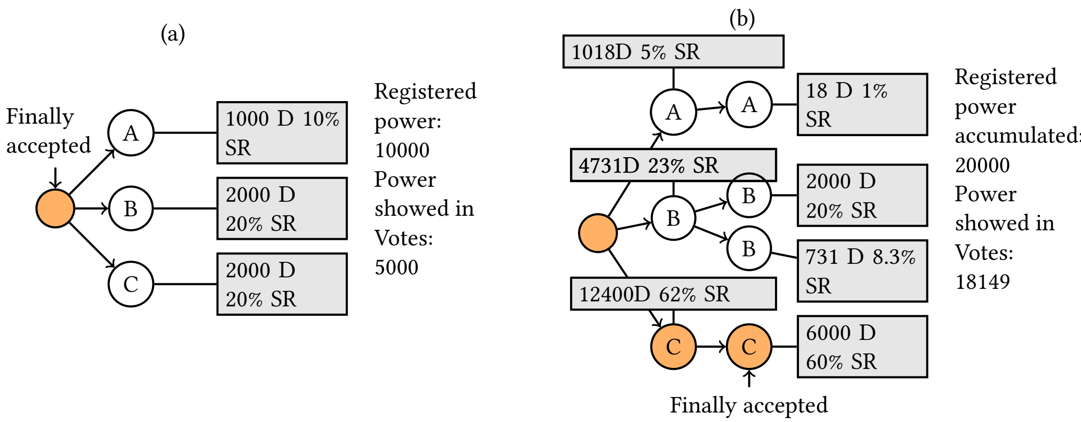


Fig. 3. Accepting a block

Observe, that condition 6b given above seems to suggest that it is difficult to achieve sometimes in the process of deriving consensus, if we only count the PoWs in one block height. However, as we count the PoWs for the whole branch stemming from the block (the block and all its descendant blocks), it is actually quite easy to do. In other words, if a consensus is not reached in a block height, it may be reached in the next block height, in the third, the fourth and so on. This is unlike a synchronous protocol: the state will not stop from progressing if a consensus is not reached within the current block interval.

### 4.3 Details of the MWPoW+ protocol

We now describe in detail how the proposed MWPoW+ protocol works, including nodes joining process, competition rounds, rewarding and adjustment mechanisms and consensus resolution.

*4.3.1 Nodes joining.* Suppose that a new node, *Bob*, is to join the system. The steps that *Bob* takes to join are given below:

(1) *Bob* creates and sends an entrance ticket which contains its public identity key, its calculation power $CP_{Bob}$ to be used in competition, the hash of the latest block at the time when the entrance ticket is created, and a Nonce that makes the hash of this entrance ticket fulfil $CP_{Bob}$.

(2) A block at the block height $X + 2$ will collect entrance tickets that contain the hash of a block at the block height $X$. When a block is published with *Bob*'s entrance ticket in it, a $TR$ (Try Range) which is a number interval within $[0, 2^{256}]$ is assigned to *Bob*, and *Bob* is then considered having joined the system after this block. Note that $TR$s for nodes never overlap and $TR_i$ is calculated by

$$TR_i = \left[\sum_{k=0}^{i-1} Tt_k, \sum_{k=0}^{i} Tt_k\right), Tt_{i \in N} = \frac{CP_i}{CP} \times 2^{256} \tag{7}$$

where $CP = CP_0 + CP_1 + \cdots + CP_{N-1}$, the overall calculation power claimed by all the registered nodes in the system, $N$ the number of registered nodes in the network, and $CP_i, 0 \leq i \leq N-1$ is the calculation power claimed by registered participant $i$.

Figure 4 shows an example of a node joining the system.

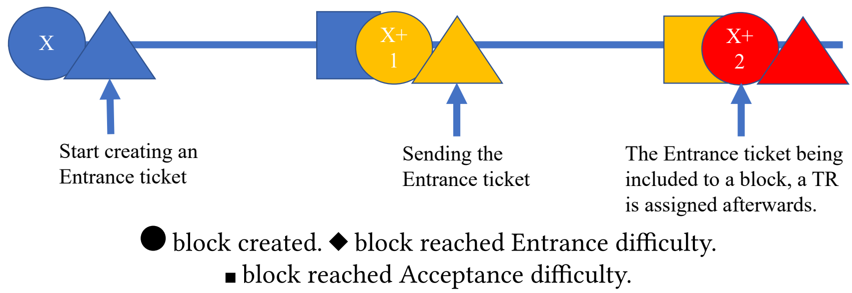


Fig. 4. An example of joining the system

#### 4.3.2 *Competition.*

There are two difficulties: *ED* (Entrance difficulty) and *AD* (Acceptance difficulty) for every round of competition. The following explains how *Bob* participates in one round of competition.

(1) *Bob* first constructs a block and tries to find a Nonce that makes the block reach *ED*.
(2) A block is broadcast to all the nodes when it reaches *ED*. *ED* of a new epoch is adjusted based on how many valid blocks reach *ED* in the previous round of competition.

$$ED_x = \min\left(\frac{NE_{x-1}}{DN} \times ED_{x-1}, \frac{AD_x}{2}\right) \tag{8}$$

where $ED_x$ and $AD_x$ are the entrance difficulty and the acceptance difficulty respectively at block height $X$, $NE_{x-1}$ the number of blocks reaching entrance difficulty at block height $X-1$, and $DN$ the ideal number of $NE$ which is set to 1.
(3) *Share* is a container of Nonce for a block. The Nonce inside a *Share*, sent by a node, must make the hash of the block fulfil at least 25% of this node's calculation power. Alternatively, if the node is proposing a block then the Nonce should fulfil *ED*. When *Bob* receives a block, and a *Share* which fulfils *ED*, it verifies the block. Then, it should stop creating its own block if it is not already working on the blocks of other nodes. *Bob* tries to create *Shares* for the first block, to its knowledge, reaching *ED*.
(4) When a valid *share* is created, *Bob* should broadcast it immediately. During the attempt to create *Share*, if a Nonce of *AD* is found, *Bob* then broadcasts the *Share* containing the Nonce. This block is then considered *announced*, and all nodes who have sent four *Shares* for it should move to its descending block height. Nodes that have not sent four *Shares* for it should only move to its descending block height when the remaining *Shares* are sent. The first block which reaches *AD* in an epoch should be placed in the mainchain. *AD* is adjusted based on how much time has been consumed for the preceding block to achieve the *AD*.

$$AD_x = \frac{BI \times AD_{x-1}}{Timestamp_{x-1} - Timestamp_{x-2}} \tag{9}$$

where $AD_x$ is the *AD* at block height $X$; *BI* is a pre-defined block interval, and $Timestamp_x$ is the time at which block $X$ is created.

(5) An *announced* block will only be accepted according to the acceptance criteria stated in Section 4.2.
(6) After *Bob* has sent four *shares*, and there is at least one *announced* block at the current block height, *Bob* moves to work on the next block height of the *announced* block that has the most support.

Figure 5 shows the flowchart of the competition procedure.

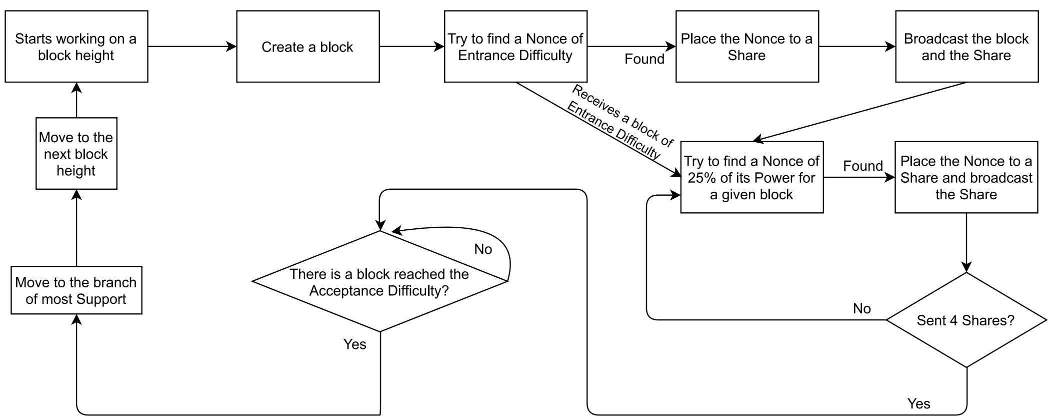


Fig. 5. The procedure followed by a node in a competition

*4.3.3 Block structure, Rewarding, and Membership adjustment.* A block is divided into four sections: The block header, the entrance tickets section, the *shares* section and the transaction section. The entrance tickets section records the latest valid entrance tickets and the entrance tickets embedded in the preceding blocks are not inherited. The *shares* section records, to the block creator's knowledge, all the *shares* sent for the blocks at the previous block heights that have not yet been written into a block of the preceding block height. The transaction section records the latest transactions.

Because nodes are overseeing different *TRs* and *shares* are signed by their identity keys, MWPoW+ can split the mining reward among the registered nodes.

$$R_{i \in N^R} = \frac{SD_i}{SD_{\{X\}}} \times R_{\{X\}} \tag{10}$$

$N^R$ represents the registered nodes whose *share* has been embedded in the block. $R_{\{X\}}$ is the overall reward assigned from the system to the block in block height $X$; *Shares* of block $X$ are embedded in block $X+1$. $SD_{\{X\}}$ is the total difficulty of the *shares* embedded in block $X+1$; $SD_i$ is the difficulty of the *shares* which the node $i$ sent, and $R_{i \in N^R}$ is the amount of remuneration given to the node $i$ as a Coinbase transaction in block $X + 1$.

If a node has not contributed at least two *Shares* for any block at a block height according to the record shown in a descending block, say *Alice*, of that block height, its membership is cancelled from *Alice* and the branch stems from *Alice*. Otherwise, a node does not need to send the entrance ticket again to rejoin the system.

It is important to simplify MWPoW+ blocks so that an increase in the number of participants will not significantly affect block size. We use the block simplification algorithm Graphene [20] to simplify a block when its size increases due to embedding entrance tickets and *Shares*. Graphene combines Bloom filter [18] and IBLT [10] and has detailed mechanisms to deal with the failure of decoding. Note that while a block is simplified, nodes still need to synchronise all entrance tickets and *Shares* in the system to decode simplified blocks. Figure 6 shows the structure of the block.

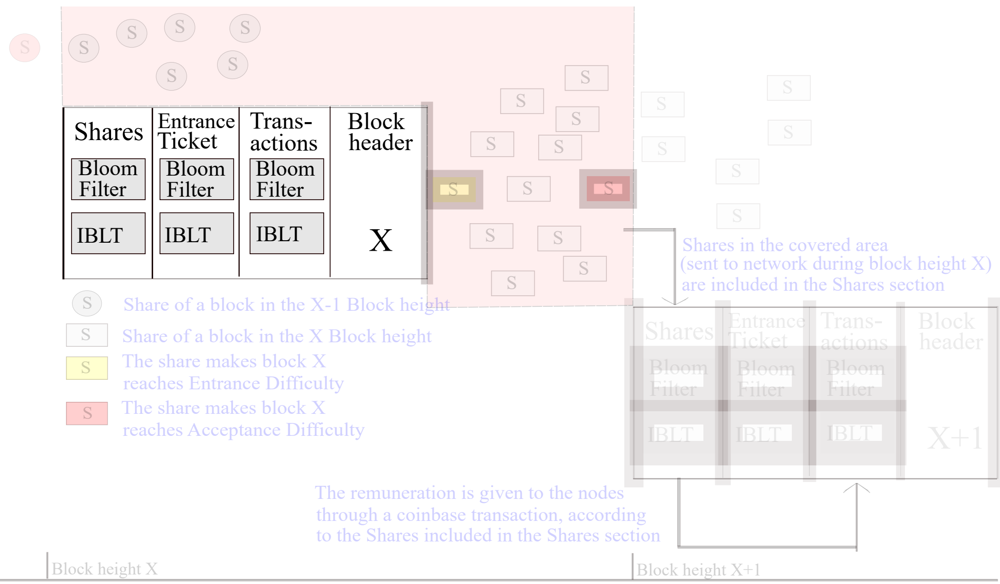


Fig. 6. Block Structure

*4.3.4 Shares.* When sending *Shares*, a node should ask for a signed acknowledgment from several nodes which together have at least 50% of the overall calculation power. A node should not start creating *Share* for a block in a block height $H$ if the following two conditions are met at the same time: (1) This node is not a new participant beginning at block height $H$ and (2) It hasn't finished sending its *shares* at block height $H-1$. When these two conditions are met at the same time for a node, say *Bob*, an honest node should not reply to a request for acknowledgment of *Bob*'s *share*, if the *share* is for the block height $H$.

If an honest node receives an unknown *share*, it should re-transmit it to all the nodes in the network. Therefore, under the security assumption that more than half of the participated power is honest, when a node sends its *shares* to several nodes which represent more than half of the overall calculation power, its *shares* will reach all the nodes.

We allow weak synchronisation for the content of the blocks, which means that provided a block does not contain faulty information, regardless of whether the information has been received previously, the block will be accepted. However, when the following three conditions are met at the same time, a node should broadcast an opposing request to reject the block: (1) the node has sent at least two *shares* for a block at the preceding block height; (2) the node's *shares* are not all included in the block and (3) the result in its membership has been cancelled. The opposing request should contain all the signed acknowledgment the node has received for the *shares* it sent. To fulfill the security, we add a flag into the *Share* structure which states if the node currently supports the block or not. Having received a valid opposing request and the block has yet to reach the acceptance criteria for it to be finally accepted, all honest nodes who have sent *shares* for this opposed block continue to generate *shares* for this block, but they turn the flag in their *shares* to "Not support". The PoW in a "Not support" *share* should not be counted when doing vote counting. The honest nodes can switch to a new block in the next block height.

There can be two possible scenarios for a block at the time when it has been opposed.

- **Reached the criteria for final acceptance**: In this case, a consensus has been reached and this block is accepted. The membership is cancelled for the opposing node from this block onwards (from this block and the branch it stems).
- **Not yet reach the criteria for final acceptance**: In this case, all honest nodes turn the flag in their *shares* to "Not support", the opposed block will not receive further support from the honest nodes.

The security assumption assumes that the honest node will not vote for more than one block at the same block height. The opposing request would not create ambiguity because the security assumption remains.

## 5 SECURITY ANALYSIS AND CORRECTNESS ENFORCEMENT

### 5.1 Power constrain

Because nodes cannot begin voting for block height $i+1$ if they have not finished their vote for block height $i$, adversary nodes cannot avoid consuming calculation power by skipping voting. By Lemma 5.1, in order to catch up with the majority, adversary nodes must bind the given amount of calculation power to every round of consensus resolution. The chain of consensus is progressed periodically. In order to catch up with the majority and the progress of the block height, the nodes must work on the PoW constantly. This design bounds the calculation power of all the nodes to every block height.

Lemma 5.1. *A node $p$ posts four votes of $w$ difficulty per vote per block height until the $i$-th block height requires the same calculation power as a node $q$ who posts $4 \times i$ votes of $w$ difficulties per vote at the $i$-th block height for all block candidates in the chain from the first block height to the $i$-th block height.*

Proof. The calculation power is defined by the PoW difficulty that one can achieve within the given time interval. This time interval is the consensus interval, which is maintained by adjusting the acceptance difficulty. Assume that node $p$ has

$$CP_p = \frac{4 \times W}{BI} \tag{11}$$

and node $q$ has

$$CP_q = \frac{4 \times W \times i}{BI \times i} \tag{12}$$

where $BI$ is a predefined block interval. Therefore,

$$CP_p = CP_q \tag{13}$$

□

Lemma 5.2. *If two honest node $p$ and $q$ accept $Rp$ and $Rq$ then, at all times $Rp = Rq$.*

Proof. For this section only, let $n$ be the overall registered power, $f$ be the adversary power, $f < n/2$. Let there be two different block candidates $Rp$ and $Rq$. Assuming that there is no descendent block candidate, $P$ is the calculation power derived from valid votes at a given time. $P(U)$ is the amount of calculation power which has not yet voted. Assume that every node has voted, $P(U) = 0$. The support rate for $Rp$ is

$$SR(Rp) = \frac{P(Rp)}{P(Rp) + P(Rq) + P(U)} \tag{14}$$

and the support rate for $Rq$ is

$$SR(Rq) = \frac{P(Rq)}{P(Rp) + P(Rq) + P(U)} \tag{15}$$

For both $Rp$ and $Rq$ to be accepted, we require $SR(Rp) > 50\%$, $SR(Rq) > 50\%$. So $Rp$ and $Rq$ cannot be accepted at the same time.

For $Rp$ to be accepted at time 1, then the consensus is withdrawn because $SR(Rq) > 50\%$ at time 2: At time 1, $SR(Rp) > 50\%$ and

$$P(Rp) - P(Rq) >= \frac{n}{4} \tag{16}$$

To make $SR(Rq) \geq 50\%$ at time 2, more than

$$f' > P(Rp) - P(Rq) \tag{17}$$

amount of calculation power from the adversary must have voted for $Rp$ before time 1, they then create a fork vote chain that votes for $Rq$ after time 1, again consumes $f'$ amount of calculation power (as the Nonce in every vote must fulfil $50\%$ of the registered power of the voter). By this operation, they temporarily **invalidate (cancel)** their votes in $Rp$ (as discussed in the working procedure, when there are two vote chains of equal length, only the votes before forking are counted).

Therefore, for $SR(Rp) > 50\%$ first, then $SR(Rq) \geq 50\%$, more than

$$f > 2 \times f' = \frac{n}{2} \tag{18}$$

of calculation power is required. Note that the nodes can vote for the descendent block candidates of either $Rp$ or $Rq$, their votes will not be invalidated in that case because it does not violate the rule that a node should only vote for one block candidate at one block height. However, given the honest node will not vote for more than one block candidate per block height, and the honest node always follows the most supported block candidate, and an honest node has more than $\frac{n}{2}$ power, $Rp = Rq$ at all times. □

Lemma 5.3. *If an adversary attempts to invalidate/change its votes in the block height $k$, it must first invalidate its vote in the block height $k + 1$.*

Proof. As discussed in the working procedure, a node should only vote for one block candidate per block height, and every node should vote four times at each block height. The votes of a node at the vote chain height $i$, $\{i | i = 4k, k \in \mathbb{N}\}$, $i + 1$, $i + 2$ and $i + 3$ are the votes for a block candidate at the block height $k$. According to the vote chain, if a node vote for multiple block candidates at the same block height $k$, its votes in its vote chain after the vote chain height $i$ will not be counted.

To invalidate/change a vote for a block candidate of the $k$ block height, which has been counted, the adversary must generate a fork vote chain at the $k$ block height, and this chain must be of the same length/longer than the current vote chain. Therefore, to invalidate/change a vote $i$, the adversary must invalidate/change the votes in the vote chain height $(i + 1) \ldots (4 \times H)$ by creating a fork vote chain, where $H$ is the latest block height. □

### 5.2 Unable to reach a consensus?

Let honest nodes altogether have $x$ calculation power and adversary nodes have $x-1$ calculation power, $x>2, x\in\mathbb{N}$, then Equation 5 is equal to

$$\begin{aligned} i\times(x-(x-1)) &> \frac{1}{4}\left(i\times(x+x-1)\right) \\ &\longrightarrow 1>\frac{1}{2}x-\frac{1}{4} \end{aligned} \tag{19}$$

So Equation 19 does not hold for $x>2, x\in\mathbb{N}$. Note that in Lemma 5.2, we only show that an adversary with $\lfloor(n-1)/2\rfloor$ calculation power will not be able to reverse a consensus. However, if an adversary does not attempt to reverse a consensus, but instead supports consensus $Rq$ while honest nodes support consensus $Rp$, then $Rp$ cannot reach the acceptance criteria given in Equation 5 as suggested by Equation 19.

In this case, if a node believes $Rq$ is faulty, it can simply ignore $Rq$ and commit to $Rp$ because the $x$ nodes voted for $Rp$ are honest nodes, as the honest nodes will not vote for $Rq$. However, if $Rq$ and $Rp$ are both correct (acceptable), we cannot assume that all nodes voted for $Rp$ are honest, therefore the relationship between $P(Rp)$ and $P(Rq)$ can be reversed and we cannot commit to $Rp$. To solve this problem, we must be able to determine if a node which votes for a minority branch is an adversary or just an honest node which has not yet synchronized with previous votes. To enable this, we change the structure of vote by adding a **Vote Array** which records the previous votes at the $i-1$ vote chain height from the nodes which have together registered at least $\lfloor n/2\rfloor+1$ of the total calculation power, supported the current vote height is the $i$ vote chain height.

When receiving a vote, we can then generate a graph based on the vote array of all the linked votes. We can derive the vote chains for every node from the voter's perspective. Then, we can know if the node has downloaded all the necessary votes to make the correct judgment, and if it has a reasonable ground to stay in the minority branch. Figure 7 shows an example of the connected graph. Assume that an honest node is in a different branch from node $N3$. Through this graph, an honest node knows that $N3$ has not yet received the vote $N1'$ at the vote chain height 4, therefore it counts the votes of $N1$ at vote chain heights 3 and 4. If the voter receives the vote $N1'$ at the vote chain height 4, it would not count the votes of $N1$ at the vote chain heights 3 and 4, because if there are two vote chains of equal length, only the votes before forking are counted. The honest node can then determine if $N3$ is out of synchronisation or is an adversary.

Lemma 5.4. *Let there be two block candidates $Rp$ and $Rq$ at the same block height and $P(Rp)>P(Rq)$. Node $q$ in the branch of $Rq$ is an adversary if $P_q(Rp)-P_q(Rq)-P_q(U)>\frac{CP[H]}{2}$*

Proof. As discussed in Section 5.2, a vote at vote chain height $i$ has links to votes at vote chain height $i-1$ which has been voted by the nodes who have together registered at least $\lfloor n/2\rfloor+1$ of the total calculation power. The vote chains of every node can be derived from the connected graph of the votes.

Because an adversary only has $\lfloor n/2\rfloor$ calculation power, there must be one honest vote in the vote array of every vote. Assuming a vote $V$ who votes for $Rp$ at vote chain height $i$ broadcast the vote, the honest nodes in $Rq$ are able to use this vote to synchronise the view from the nodes in $Rp$ at vote chain height $i-2$. Because there must be one link to the vote for $Rp$ in vote chain height $i-1$ which indicates in the Vote Array of $V$ (due to $P(Rp)>P(Rq)$), and that link provides the view from the nodes in $Rp$ for the vote chain height $i-2$.

Assuming that a vote $V_1$ at vote chain height $i$ supports $Rq$ and the voter of $V_1$ has different views for the most votes in vote chain heights $i$ and $i-1$ with the nodes which support $Rp$: the calculation power for $Rp$ in vote chain heights $i$ and $i+1$ are therefore counted in either $P(U)$ of $P(Rq)$ from

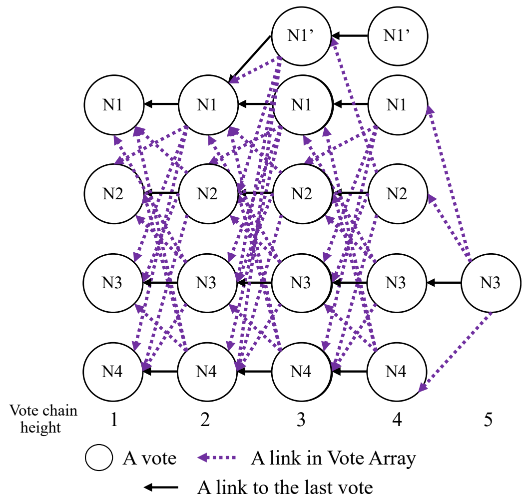


Fig. 7. The graph created using the *Share* of a node $N3$ at the vote chain height 5.

the voter's perspective. Each vote chain height globally corresponds to $\frac{CP[H]}{4}$ of calculation power, therefore the maximum amount of calculation power that the voter fails to synchronise with the nodes which support $Rp$ is lower than $\frac{CP[H]}{2}$. $CP[H]$ is the overall registered power at the block height $H$. In this case, if $P_q(Rp) - P_q(Rq) - P_q(U) > \frac{CP[H]}{2}$, this voter is considered an adversary. $P_q(U)$ is the accumulation of unused calculation power. Because even assuming all the calculation power has voted for $Rq$ at vote chain heights $i$ and $i-1$, we still have $P(Rp) > P(Rq)$ at vote chain height $i$. Therefore, the honest nodes have no reason to still support $Rq$, then this voter is an adversary. □

When a node is deemed to be an adversary, its votes are discounted in the blockchain from that point onwards. Therefore, eventually Equation 5 can be achieved if an adversary chooses to support $Rq$ instead of the most supported $Rp$.

### 5.3 Correctness

Theorem 5.5. *The proposed MWPoW+ protocol is an $\lfloor (n-1)/2 \rfloor$-resilient strong consensus protocol.*

Proof. As Lemma 5.1 shows, the calculation power is constrained in every block height. Therefore, all nodes (including adversaries) must actively participate in voting in order to catch up with the block height, and they cannot skip any votes. As shown in Lemma 5.3, to cancel/change a vote, an adversary must cancel/change all the votes after this vote, so that to cancel/change a consensus, the adversary must cancel/change all the consensus after this consensus. As shown in Lemma 5.2, there cannot be a different consensus at the same time, so it is strong consensus protocol, and for a consensus to be cancelled/changed, an adversary must have more than half of the calculation power. As shown in Lemma 5.4, the system will ultimately commit on the correct consensus, when honest nodes have at least $\lfloor (n-1)/2 \rfloor + 1$ calculation power, and an adversary has up to $\lfloor (n-1)/2 \rfloor$ calculation power in a $n$ calculation power system. When assuming every node has the same calculation power, our protocol achieves $\lfloor (n-1)/2 \rfloor$-resilience in a $n$-node system. □

The design of MWPoW+ is similar to MWPoW in that (1) a determined node membership and each node's PoW difficulty is recorded; (2) a consensus is reached by voting instead of using the longest-chain mechanism. (3) fair compensation for nodes that contribute to consensus derivation. However, MWPoW is insecure and lacks formal security proofs and verification. MWPoW+, on the other hand, offers the necessary and desirable security features for a strong consensus protocol, as our proofs in this paper show. Table 2 shows the key vulnerabilities of MWPoW and improvement offered by MWPoW+.

Table 2. Key differences between MWPoW and MWPoW+

| Limitation of MWPoW design | Support in MWPoW+ | Technical enabler |
|---|---|---|
| *shares* are not signed. | The creator of *share* needs to sign them using the identity key. | In MWPoW, an adversary may work on other nodes' $TR$ and change the consensus by cancelling their votes. In MWPoW+, *shares* are signed and an adversary cannot work on other nodes' $TR$. |
| Nodes move into creating the descending block immediately after a block reaches $AD$ regardless of whether the nodes have finished their *shares* or not. | We ask the nodes to continue working on a block if it has not finished sending the shares for the block (see Section 4.3.4). | As a node only needs to send two *shares* per iteration to keep itself in the system, an adversary may double its power by only using half of the declared power per node per round. In MWPoW+, we ask the nodes to continue working on a block if it has not finished sending the shares for the block. A block may contain the *shares* for its preceding block as well as ancestor blocks. However, if the block has not caught up the mainchain on time, it is expelled (see Section 4.3.4). |
| A block only records the *shares* for its preceding block. | We ask a block to record the *shares* for all the blocks at its preceding block height. | This allows nodes to change branches at the new block height but does not jeopardise security, because the new block height is, in the end, another round of voting, and the proof in Section 5.2 still holds. This improvement avoids nodes being expelled from one branch when it has voted for another. However, such a case is rare, as $ED$ is adjusted to control the number of block candidates. |
| There is no "Last vote hash" or "Vote Array" in the *Share* structure. Therefore there is no vote chain. | We added "Last vote hash" and "Vote Array" in the *Share* structure to support vote chain and vote array. | Without a vote chain and a vote array, the system cannot determine if a node is a delayed node or an adversary when reaching the consensus. The system may encounter the problem discussed in Section 5.2 that a consensus may never be reached. |
| No chance to oppose a block. | Nodes require acknowledgements for their *shares*, and the "Not support" option is added to the *share* structure. | These prevent an adversary from removing an honest node by publishing a block which does not include the honest node's *shares*, and also avoid the ambiguity as stated in Section 4.3.4. |

We end this section by noting the significance of the new mechanisms that have been designed into MWPoW+. Although as a protocol, they are simply some modified steps and additional record keeping structures for consensus derivation, their importance can be seen from the discussion and proofs we gave in the previous sections. These mechanisms collectively have made MWPoW+ both more secure and more robust than its predecessors. For example, a vote-based blockchain suffers from progress hindering attack due to its design that counts only the votes for a block instead for the branch of blocks stemmed from it. With the new mechanisms, MWPoW+ is now able to support validated strong consensus that allows nodes to unite on the same branch of the blockchain without slowing down its progress. Equally, lacking designs like "vote array" to form a "vote graph" in MWPoW means that one cannot ensure that support from one node for a block branch is finalized and that a consensus reached cannot be reversed later on. These new mechanisms, therefore, serve as the key components in a more secure and efficient consensus protocol that we propose in this paper. Without these, our protocol cannot achieve $\lfloor (n-1)/2 \rfloor$-resilience in an $n$-node system.

# 6 COMPARISON AND ANALYSIS

## 6.1 Time complexity

In the proposed MWPoW+, as well as blockchains using PBFT and PSB, we ask weak synchronisation for the content in a block. This design means that it is the sender's responsibility to guarantee that information such as transactions or entrance tickets reach as many nodes as possible. Therefore the time difficulty for sending information like transactions and entrance tickets to the network is $O(n)$ in an $n$-node system. In this section, we will consider time complexity of completing one epoch of communication among nodes using different consensus protocols.

*MWPoW+.* We require an all-to-all re-transmit for the *shares*, so a node should re-transmit any unknown *share* it receives. A node needs to generate four *shares* for one epoch. It first broadcasts its *shares* to $n$ nodes, then each receiver re-transmits the *share* to $n$ nodes. Therefore the time needed for a share to be received by all the nodes is $O(n^2)$. Because there are $4n$ shares, the overall time needed for the nodes to synchronise all the *shares* is $O(4n^3)$. There can be many block candidates, but as we adjust *ED* to ensure there will be only one block candidate most of the time. We also require an all-to-all re-transmit of the block candidates because the nodes need to respond to and send the "oppose request" for the faulty blocks rapidly (see Section 4.3.4). Therefore, the overall time complexity is

$$TD_{MWPoW+} = O(n^2 + 4n^3) \tag{20}$$

*PBFT.* The primary sends a new block to all the nodes $O(n)$ (pre-prepare stage). Every secondary then broadcasts if it agrees with the block to the $n$ nodes in pre-prepare stage, and there are $n$ nodes. Therefore the time needed is $O(n^2)$. Every node broadcasts a vote for commit in the commit stage, and the time needed is also $O(n^2)$. Therefore the overall time complexity is

$$TD_{PBFT} = O(n + 2n^2) \tag{21}$$

*PSB.* At the start of an epoch, every node sends a notice informing the primary what the latest state is, and this requires $O(n)$ time (the security assumptions ensure that there can be a consensus there). The primary uses the consensus state to generate a block and sends it to all the nodes, so the time needed is $O(n)$. The nodes decide on this block and broadcast their decision (commit stage), and this takes $O(n^2)$ time. Then, the nodes take $O(n^2)$ time to broadcast to every node based on the decisions they have received in the commit stage. Therefore the overall time difficulty is

$$TD_{PSB} = O(2n + 2n^2) \tag{22}$$

*SC.* As stated in [8], the overall round complexity for the byzantine broadcast protocol run in parallel is

$$TD_{SC} = O(f \times n^2 \times \frac{n}{f} \log iv) = O(n^3 \log iv) \tag{23}$$

## 6.2 Analysis

If viewed purely as a consensus protocol and do not consider the progress hindering attack, MWPoW+ appears to be rather complicated when compared to other approaches. However, when used to support blockchain sharding, it has some distinctive advantages. First, the process of reaching consensus in MWPoW+ does not affect the progress of block height. Nodes can mine on blocks which have not been accepted yet. This is not the case with PBFT or PSB based blockchains, where they must reach a consensus before moving on to the next block height. Second, as the block interval is originally designed for nodes to discover the transactions being sent to the network, both PBFT and PSB based blockchains need to wait for a predefined interval before a new block of the next block height can be proposed. In the traditional strength-competition-based blockchains, as there is no need for voting, they have to wait for the given time-bound to accept the longest chain as the mainchain. However, MWPoW+, as a new type of strength-competition blockchain that mix strength-competition with voting, does not need to wait for this time-bound before the mainchain can be determined. As there is voting process in every epoch in MWPoW+, a consensus is usually achievable within every epoch or a short series of continuous epochs in extreme. Therefore, MWPoW+ can be considered as a fast-confirming solution for strength-competition based blockchains. Third, MWPoW+ also frees a blockchain system from the concern of selfish-mining attack for a strength-competition blockchain: no block can be hidden after one interval as the computation power per round has been pre-registered. The process of reaching consensus for a block height is linked to the process for reaching consensus for other block heights. So the votes are shared. Therefore, to compare with existing strong consensus protocols that separate one epoch from others, the time complexity is improved. Table 3 shows a comparison between different consensus protocols.

| | MWPoW+ | PBFT | PSB | Nakamoto Blockchain | ETH | SC |
|---|---|---|---|---|---|---|
| Time complexity level | $O(n^3)$ | $O(n^2)$ | $O(n^2)$ | $O(n)$ | $O(n)$ | $O(n^3)$ |
| Final confirm | $\sim B$ | $B$ | $B$ | $3B$ | $36B$ | $B$ |
| Byzantine tolerance | $p/2$ | $n/3$ | $n/2$ | $p/2$ | $p/2$ | $\frac{n}{\max(3,iv)}$ |
| Selfish-mining free | Yes | Yes | Yes | No | No | Yes |

Table 3. Comparison between different blockchain consensus protocols, where $B$ stands for one pre-defined block interval, $p$ the overall calculation power, $n$ the total number of nodes in the system. There is no compulsory all-to-all broadcast for a traditional strength competition blockchain, therefore, their time complexity is $O(n)$ where the block creator broadcasts its block to all the nodes. MWPoW+ uses roughly one block interval, it is possible that it took several block intervals.

## 7 MWPOW+ FOR BLOCKCHAIN SHARDING

While MWPoW+ has a greater time complexity when compared to other proposals, it can increase overall security as well as the number of shards allowed when used as an intra-shard communication protocol for blockchain sharding. We have shown in Section 2.1 that synchronous consensus protocols can be used for blockchain sharding and a security proof given in Section 2.2 shows that an $n/2$ security security level can be achieved. We have also shown that MWPoW+ can handle progress hindering attacks which have an even worse impact on a blockchain sharding environment. This is because as blockchain sharding requires a weak global synchronisation for cross-shard transactions, progress hindering attacks inside a shard can destablise the system. In short, MWPoW+ as an intra-shard consensus protocol, has the following merits:

1. MWPoW+ is strong consensus protocol, therefore, no progress hindering attack can happen.
2. MWPoW+ is for strong consensus but achieves the same security level as a synchronous binary consensus protocol.
3. MWPoW+ separates the process of block height progressing from the consensus reaching. As such, it allows synchronous block height among shards although the consensus resolution process is asynchronous.

(4) The reward system of MWPoW+ allows power-constrained nodes to receive compensation for competition, and therefore encourages node participation. This helps blockchain sharding which requires a large number of nodes to participate in order to maintain security and split more shards [28].
(5) With only a small number of nodes to be expected inside a shard, the time complexity for MWPoW+ is practically acceptable.

We have stated in the Introduction that common blockchain sharding protocols require every node to be equal in voting in order to maintain security. When applying MWPoW+ to ordinary blockchain sharding systems like Rapidchain [31] and the two sharding protocols proposed in [28, 29], we only need to ask the nodes to write down the same calculation power in their entrance tickets. By calculation power registration, we can upper bound the calculation power of nodes, so that they are all equal in votes.

Table 4 shows a comparison of MWPoW+ used in both sharded and non-sharded settings, and other consensus protocols. When a $n = 2000$ system runs at a $10^{-6}$ failure probability and it splits $s = 16$ shards with each sized $m = 125$. This suggests a time complexity of $O(125^3)$ for this setting, which is largely smaller than $O(2000^2)$, therefore making it acceptable in practice. On the other hand, when MWPoW+ is used in a sharded setting, it can allow $s$ times more transactions per iteration compared to non-sharded settings.

| | MWPoW+ | PBFT | PSB | Sharded MWPoW+ |
|---|---|---|---|---|
| Time complexity | $O(n^3)$ | $O(n^2)$ | $O(n^2)$ | $O(m^3)$ |
| Progress-hindering-free | Yes | No | No | Yes |
| Transaction per iteration | $V$ | $V$ | $V$ | $V \times s$ |

Table 4. A comparison between MWPoW+ and other consensus protocols when used for blockchains, where $V$ is a pre-defined number of transactions per block, $s$ the number of shards and $m = n/s$.

One may argue that improvement in performance is due to the sharding model itself, rather than from MWPoW+ as an intra-shard consensus protocol, and when PBFT or PSB are applied to sharding, even better performance can be achieved. However,

(1) Rapidchain [31] and the models proposed in [28, 29] use binary synchronous consensus protocols face the progress-hindering problems.
(2) When a progress-hindering attack happens, the performance is degraded, if the time needed for the system to recover from the attack is taken into account.

Also, as we require a large number of nodes for blockchain sharding, the sharded blockchains are often run in open-membership settings. There is no constraint to prevent an attacker from making a progress-hindering attack. The problem does not only exist at the system level, but also exists at the shard level. This means that the probability for the problem to occur in a system as a whole is $s$ times more likely than in non-sharded approaches. We, therefore, consider MWPoW+ to be a more robust and practical protocol for blockchain sharding.

Note that we have not offered an analysis of what the preferred size limit of a shard should be here, because this would depend on network conditions, participants' bandwidth and network quality. The shard size affects the security of the sharding model and the network and the hardware requirement for participants. A preferred shard size could be considered in practice on a case-by-case basis.

## 8 EXPERIMENT

To our best knowledge, there is no work that has so far discussed the progress-hindering attack in a sharded system and there are no blockchain sharding approaches that use a strong consensus protocol. In this experiment, we evaluate (1) how the performance of some recent blockchain sharding approaches [12, 28, 29, 31] is affected by the progress-hindering attack; (2) how does the result compare to an approach that implements MWPoW+ as an intra-shard consensus protocol. Our experiments show that MWPoW+ is resilient when facing such attacks and is useful for sharded blockchain systems.

The experimental setup for each network was a cluster of 8 compute nodes, each offering dual AMD EPYC 32-core CPUs and 256GB of memory. Computer nodes are connected with a 25Gbit network providing RoCE for efficient communication. We implemented four networks, each having 2000 nodes. The communication layer of the experiment is simulated by configuring these computer nodes as a graph, and we map each node of our protocol to a node in the graph. Networks 1 (sharded MWPoW+) implement our previous sharding model[29] but use MWPoW+ as the intra-shard consensus protocol. Network 2 (Flexible) uses the same sharding model but with a simple synchronous binary consensus protocol. Network 3 runs the RapidChain [31] model and Network 4 runs Omniledger [12].

The experiment lasted 2000 block heights, and every block contained 2000 transactions. The experiment began with zero adversary nodes in every network, and we randomly selected one node per block height in every network and turned it into an adversary. The adversary nodes in the previous block height would remain as adversary nodes. This process of adding adversary nodes ended when the overall adversary population in the respective network reached the pre-defined security level. Then the adversary population would remain for the remaining block heights. In every block height, a leader node was randomly selected. If an adversary was selected as the leader node, it would make a progress-hindering attack, and the nodes could not reach a consensus before the leader node was replaced.

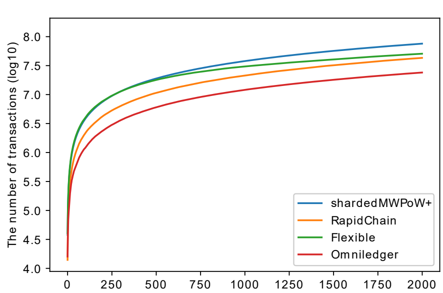


Fig. 8. The overall number of transactions ($log10$) committed into the blockchain with the block height progress

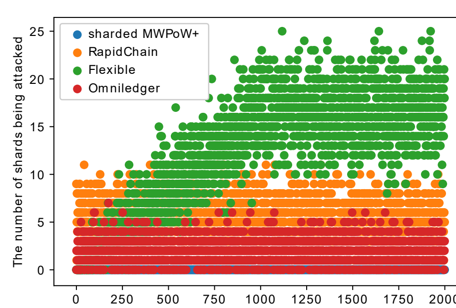


Fig. 9. The number of shards attacked per block height

Figure 8 shows the overall number of transactions being committed into the blockchain with the progress of block height. Because the sharded MWPoW+ is not affected by the progress hindering attack, it outperforms Network 2 which used the same sharding model, and the differences become larger with the increase in block height. Again, MWPoW+ is free from progress hindering attacks, it commits transactions in a stable rate, while others are very unstable due to some shards cannot reach a consensus. Figure 9 shows the number of shards attacked in every block height. Note that

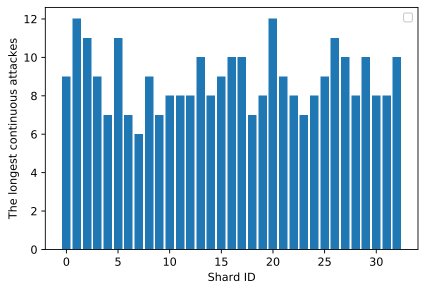


Fig. 10. Flexible, the maximum number of continuous succeeded attacks during the experiment

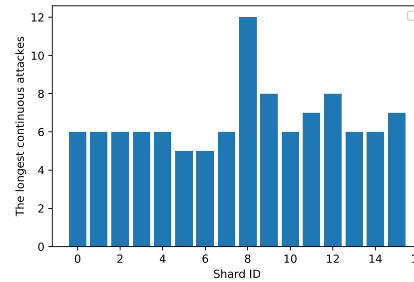


Fig. 11. RapidChain, the maximum number of continuous succeeded attacks during the experiment

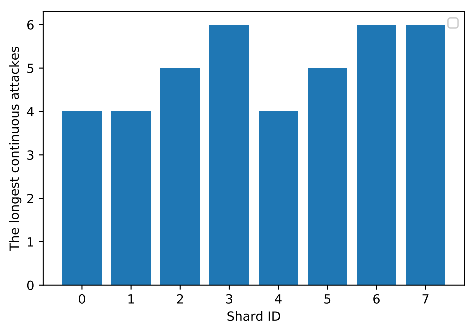


Fig. 12. Omniledger, the maximum number of continuous succeeded attacks during the experiment

the number for MWPoW+ is zero. The cross-shard transactions require the record inside a shard being locked first, then they are sent to another shard and written in a block. This requires some additional procedures to be performed in order to complete the process [2, 25]. In Figure 10, 11 and 12, we show the maximum number of continuous succeeded progress-hindering attacks observed during the experiment: the shards have been hindered for a long period and all the cross-shard consensus is not able to reach a conclusion during periods like these.

## 9 CONCLUSION

In this paper, we proposed the MWPoW+ consensus protocol for blockchain sharding. We have shown and proved that although it is an asynchronous protocol, it can allow $n/2$ adversary nodes in an $n$-node system, just like a synchronous consensus protocol does. It can also allow $p/2$ adversary power in a $p$-calcualtion power system. As a strong consensus protocol for blockchain, it can provide synchronous block height among shards by separating the process of consensus resolution from the progress of block height. It is a solution for fast confirmation compared to the longest-chain based blockchains. While the protocol has a higher time complexity when used for non-sharding blockchains, its time complexity is practically acceptable in a sharding environment. Our experiments have shown that MWPoW+ eliminates the progress-hindering attack in a sharded blockchain and stabilise the system overall. It can make cross-shard operations smoother and allows for better and steadier performance. All these make MWPoW+ a desirable protocol for intra-shard consensus in blockchain sharding.

## REFERENCES

[1] Atul Adya, William J Bolosky, Miguel Castro, Gerald Cermak, Ronnie Chaiken, John R Douceur, Jon Howell, Jacob R Lorch, Marvin Theimer, and Roger P Wattenhofer. 2002. FARSITE: Federated, available, and reliable storage for an incompletely trusted environment. *ACM SIGOPS Operating Systems Review* 36, SI (2002), 1–14.

[2] Mohammad Javad Amiri, Divyakant Agrawal, and Amr El Abbadi. 2019. On sharding permissioned blockchains. In *2019 IEEE International Conference on Blockchain (Blockchain)*. IEEE, 282–285.

[3] Adam Back et al. 2002. Hashcash-a denial of service counter-measure. (2002).

[4] Miguel Castro and Barbara Liskov. 1999. Practical Byzantine Fault Tolerance. In *Proceedings of the Third Symposium on Operating Systems Design and Implementation* (New Orleans, Louisiana, USA) *(OSDI '99)*. USENIX Association, USA, 173–186.

[5] Miguel Castro and Barbara Liskov. 2002. Practical Byzantine fault tolerance and proactive recovery. *ACM Transactions on Computer Systems (TOCS)* 20, 4 (2002), 398–461.

[6] George Danezis and Sarah Meiklejohn. 2015. Centrally banked cryptocurrencies. *arXiv preprint arXiv:1505.06895* (2015).

[7] Cynthia Dwork, Andrew Goldberg, and Moni Naor. 2003. On memory-bound functions for fighting spam. In *Annual International Cryptology Conference*. Springer, 426–444.

[8] Matthias Fitzi and Juan A. Garay. 2003. Efficient Player-Optimal Protocols for Strong and Differential Consensus. In *Proceedings of the Twenty-Second Annual Symposium on Principles of Distributed Computing* (Boston, Massachusetts) *(PODC '03)*. Association for Computing Machinery, New York, NY, USA, 211–220. https://doi.org/10.1145/872035.872066

[9] Juan A Garay and Yoram Moses. 1993. Fully polynomial Byzantine agreement in t+ 1 rounds. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing*. 31–41.

[10] Michael T Goodrich and Michael Mitzenmacher. 2011. Invertible bloom lookup tables. In *2011 49th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*. IEEE, 792–799.

[11] Jonathan Katz and Chiu-Yuen Koo. 2006. On expected constant-round protocols for Byzantine agreement. In *Annual International Cryptology Conference*. Springer, 445–462.

[12] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ewa Syta, and Bryan Ford. 2018. Omniledger: A secure, scale-out, decentralized ledger via sharding. In *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 583–598.

[13] John Kubiatowicz, David Bindel, Yan Chen, Steven Czerwinski, Patrick Eaton, Dennis Geels, Ramakrishan Gummadi, Sean Rhea, Hakim Weatherspoon, Westley Weimer, Chris Wells, and Ben Zhao. 2000. OceanStore: An Architecture for Global-Scale Persistent Storage. *SIGPLAN Not.* 35, 11 (Nov. 2000), 190–201. https://doi.org/10.1145/356989.357007

[14] Leslie Lamport. 1998. The Part-Time Parliament. *ACM Trans. Comput. Syst.* 16, 2 (May 1998), 133–169. https://doi.org/10.1145/279227.279229

[15] Jinyuan Li, Maxwell N Krohn, David Mazieres, and Dennis E Shasha. 2004. Secure Untrusted Data Repository (SUNDR).. In *Osdi*, Vol. 4. 9–9.

[16] Julian Loss and Tal Moran. 2018. Combining Asynchronous and Synchronous Byzantine Agreement: The Best of Both Worlds. (2018).

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

[18] James K Mullin. 1983. A second look at Bloom filters. *Commun. ACM* 26, 8 (1983), 570–571.

[19] Gil Neiger. 1994. Distributed consensus revisited. *Information processing letters* 49, 4 (1994), 195–201.

[20] A Pinar Ozisik, Gavin Andresen, George Bissias, Amir Houmansadr, and Brian Levine. 2017. Graphene: A new protocol for block propagation using set reconciliation. In *Data Privacy Management, Cryptocurrencies and Blockchain Technology*. Springer, 420–428.

[21] Ling Ren, Kartik Nayak, Ittai Abraham, and Srinivas Devadas. 2017. Practical synchronous byzantine consensus. *arXiv preprint arXiv:1704.02397* (2017).

[22] Rodrigo Rodrigues, Miguel Castro, and Barbara Liskov. 2001. BASE: Using abstraction to improve fault tolerance. *ACM SIGOPS Operating Systems Review* 35, 5 (2001), 15–28.

[23] Fahad Saleh. 2020. Blockchain without waste: Proof-of-stake. *Available at SSRN 3183935* (2020).

[24] Fred B Schneider. 1990. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Computing Surveys (CSUR)* 22, 4 (1990), 299–319.

[25] Alberto Sonnino, Shehar Bano, Mustafa Al-Bassam, and George Danezis. 2019. Replay attacks and defenses against cross-shard consensus in sharded distributed ledgers. *arXiv preprint arXiv:1901.11218* (2019).

[26] SJ Wels. 2019. *Guaranteed-TX: The exploration of a guaranteed cross-shard transaction execution protocol for Ethereum 2.0.* Master's thesis. University of Twente.

[27] Yibin Xu and Yangyu Huang. [n.d.]. Mwpow: Multiple winners proof of work protocol, a decentralisation strengthened fast-confirm blockchain protocol. *Security and Communication Networks* 2019 ([n. d.]).

[28] Yibin Xu and Yangyu Huang. 2020. An n/2 Byzantine Node Tolerate Blockchain Sharding Approach. In *Proceedings of the 35th Annual ACM Symposium on Applied Computing* (Brno, Czech Republic) *(SAC '20)*. Association for Computing Machinery, New York, NY, USA, 349–352. https://doi.org/10.1145/3341105.3374069

[29] Yibin Xu, Yangyu Huang, Jianhua Shao, and George Theodorakopoulos. 2020. A Flexible n/2 Adversary Node Resistant and Halting Recoverable Blockchain Sharding Protocol. *arXiv preprint arXiv:2003.06990, Concurrency and Computation: Practice and Experience, DoI:10.1002/CPE.5773* (2020). arXiv:2003.06990 [cs.DC]

[30] Yibin Xu, Tijs Slaats, and Boris Düdder. 2022. Poster: Unanimous-Majority - Pushing Blockchain Sharding Throughput to Its Limit. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security* (Los Angeles, CA, USA) *(CCS '22)*. Association for Computing Machinery, New York, NY, USA, 3495–3497. https://doi.org/10.1145/3548606.3563506

[31] Mahdi Zamani, Mahnush Movahedi, and Mariana Raykova. 2018. Rapidchain: Scaling blockchain via full sharding. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 931–948.