HCA: Hashchain-Based Consensus Acceleration Via Re-Voting

Zijian Zhang, *Member, IEEE*, Xuyang Liu, *Student Member, IEEE*, Meng Li, *Senior Member, IEEE*, Hao Yin, Liehuang Zhu, *Senior Member, IEEE*, Bakh Khoussainov, *Member, IEEE*, and Keke Gai, *Senior Member, IEEE*

***Abstract*—In the context of consortium blockchain, consensus protocols set permission mechanisms to maintain a relatively fixed group of participants. They can easily use distributed consistent algorithms for achieving deterministic and efficient consensus and generate incessant blocks as the ledger. However, most of the existing consensus protocols do not sufficiently leverage the chain structure of blocks, and therefore leaving room for performance improvement. In this paper, we first propose a Hashchain-based Consensus Acceleration (HCA) protocol. The HCA protocol enables a leader to generate blocks that contain a quorum of votes on the previous block, and allow voters to re-vote for accelerating the block generation to Byzantine Fault Tolerance (BFT) consensus protocols. Then, we present a rolling-based leader selection (RLS) scheme to further optimize the HCA protocol. In the RLS scheme, the leader is changed in a round-robin fashion. Finally, theoretical analysis proves the safety, liveness and responsiveness of the optimized HCA protocol, while experimental evaluation shows that the optimized HCA protocol outperforms the existing BFT consensus protocols, from the viewpoint of efficiency.**

***Index Terms*—Acceleration, blockchain, byzantine fault tolerance, consensus, re-voting.**

I. Introduction

CONSENSUS [1], [2], [3] is a crucial component of blockchain, a unique public ledger held by participants in the network. Each participant executes the consensus protocol to pack transaction events into blocks. These blocks are linked one by one using a hashing function to form chain structure, i.e. blockchain, which shows the same view by all participants. Bitcoin [4], the first application of blockchain, employs the chain structure to propose a consensus algorithm named Proof-of-Work (PoW). In this consensus protocol, a participant, also called miner, is required to find a valid block [5] that satisfies its hash value less than a target number (i.e. mining difficulty). Although there is no restriction for mining participation, too much computing power is wasted in generating the valid block. For reliable consistency, the system has to adaptively adjust the mining difficulty to limit the speed of block generation.

To address the mentioned drawbacks of resource waste and poor performance, consortium blockchain presents a more practical network model than public blockchain. This model [6] designs a permission mechanism among miners. Only authorized miners can participate in the network to maintain the blockchain, and thus building a relatively fixed group of miners to execute the consensus protocol. As a result, traditional distributed consistent algorithms like Practical Byzantine Fault Tolerance (PBFT) [7] can be applied to the consensus group. One member of this group is specified to propose a valid block and executes the consensus protocol with other members to uniquely decide the block. Thus, consortium blockchain gets higher performance to decide blocks without waiting for more block confirmations.

However, traditional distributed consistent protocols do not have satisfying in efficiency, especially when encountering massive transactions. The reason is that these protocols rely on message exchange for the finalized blocks, which has a large communication complexity. Though some consensus protocols like Tendermint [8] use gossip protocol to spread consensus messages or apply efficient cryptographic primitives to reduce communication bandwidth, multi-phases communication still generates massive data and thus causing unacceptable latency on the network. As a result, the performance bottleneck still exists.

Different from the aforementioned protocols, the chain structure as the vital innovation of blockchain provides an alternative way to reach consensus, e.g. the longest chain rule in Bitcoin, and the heaviest chain rule in Ethereum [9]. Unfortunately, to the best of our knowledge, most of the existing consensus protocols especially for Byzantine Fault Tolerance (BFT) consensus protocols have not sufficiently leveraged the chain structure to optimize their performance. Although HotStuff attempts to adopt the

Manuscript received 29 March 2021; revised 16 January 2023; accepted 21 March 2023. Date of publication 28 March 2023; date of current version 14 March 2024. This work was supported in part by the National Natural Science Foundation of China (NSFC) under Grants 62172040, 62002094, and U1836212, and in part by the National Key Research and Development Program of China under Grants 2021YFB2701200 and 2022YFB2702402, and in part by the Anhui Provincial Natural Science Foundation under Grant 2008085MF196. *(Corresponding authors: Meng Li; Liehuang Zhu.)*

Zijian Zhang is with the School of Cyberspace Science and Technology and the School of Southeast Institute of Information Technology, Beijing Institute of Technology, Beijing 100811, China (e-mail: zhangzijian@bit.edu.cn).

Meng Li is with the Key Laboratory of Knowledge Engineering with Big Data, Ministry of Education, School of Computer Science and Information Engineering, Anhui Province Key Laboratory of Industry Safety and Emergency Technology and Intelligent Interconnected Systems Laboratory of Anhui Province, Hefei University of Technology, Hefei, Anhui 230009, China (e-mail: mengli@hfut.edu.cn).

Xuyang Liu, Hao Yin, Liehuang Zhu, and Keke Gai are with the School of Cyberspace Science and Technology, Beijing Institute of Technology, Beijing 100811, China (e-mail: 3120225643@bit.edu.cn; yinhao@bit.edu.cn; liehuangz@bit.edu.cn; gaikeke@bit.edu.cn).

Bakh Khoussainov is with the School of Computer Science, University of Electronic Science and Technology of China, Sichuan 610056, China (e-mail: bmk@uestc.edu.cn).

Digital Object Identifier 10.1109/TDSC.2023.3262283

streamline method to speed up the process of consensus, there is still room to improve the performance.

Roughly, assume there are three malicious voters, and the total amount of voters are 10, ($N = 3f + 1 = 3 * 3 + 1 = 10$). If a leader in HotStuff receives 6 votes to support a message and 2 votes to object this message. In other word, two voters support a conflict message. If so, the leader has to wait for a new vote which support the first message, such that the number of the support votes is no less than the specific threshold ($N - f = 10 - 3 = 7$). However, suppose we allow voters to change their idea and re-vote, the leader can request two objectors to update their votes. If one of them change the vote, the leader can stop waiting and continue. This is usually faster than simply waiting a new support vote from a new voter, because of the communication latency. But it brings a new challenge. That is, the safety is difficult to be guaranteed if votes are allowed to be updated, because a malicious leader can request a honest voter to change her vote from one message to another. If so, this malicious leader can obtain two valid votes for two arbitrary messages, even they are conflict with each other. It is difficult to simply combine several existing consensus protocols to solve this problem. Therefore, we create a hash chain and explore a two-round consensus protocol to guarantee that all of the re-votes will be correctly recognized and consistently handled for all of the honest voters.

From scratch, we first propose a new hashchain-based Consensus Acceleration (HCA) protocol to handle the aforementioned challenge. Due to the chain structure supported by the hashing function, the blockchain can show the self-evidence that each block obtains multiple rounds of valid votes. What we need to do is to use the deterministic PBFT principle to design a consensus protocol, ensuring only one main chain in the blockchain network. By integrating the chain structure into the traditional distributed consistent algorithm, we can improve blockchain performance in transaction throughput and consensus latency. The specific contributions of this paper are summarized as follows.

- We first propose a Hashchain-based Consensus Acceleration (HCA) protocol. The HCA protocol enables each proposal to act as a block that contains a quorum of votes on the previous block, and allow re-voting to accelerate. So far as we know, the re-voting mechanism is first to be explored to improve the performance and keep the safety of Byzantine Fault Tolerance (BFT) consensus protocols.
- We then present a rolling-based leader selection (RLS) mechanism to further optimize the process of the HCA protocol. In the RLS mechanism, the leader is changed in a round-robin fashion. This can further improve the performance and keep the liveness of BFT consensus protocols.
- We theoretically prove the safety, liveness and responsiveness of the original HCA protocol and the optimized HCA protocol. We also implement our consensus protocol in Java, and the experimental results show that the HCA protocols meet the requirement of performance enhancement.

The rest of the paper is organized as follows. The related works presented in Section II. Section III recalls the preliminaries. Then we define the necessary models, goals and messages structures in Section IV. Sections V and VI propose the HCA protocol and the Optimized HCA protocol. Meanwhile, the safety, liveness and responsiveness of both protocols are also formally proven. Sections VII and VIII run the theoretical analysis and experimental analysis for the performance. Finally, we draw the conclusion in Section IX.

II. Related Works

Since blockchain emerges, many studies introduce various techniques to solve the block confirmation delay problem in blockchain consensus, trying to reach strong consistency. The traditional BFT protocol is considered once again, which aims to change the way of deciding a block. For this topic, too many related works are involved. Due to the limited space, we just present some representative schemes to show their rough idea.

The biggest difference from blockchain is that the BFT protocol assumes a relatively static and stable network. Ripple [10] leverages collectively-trusted subnetworks within a larger network to execute a consensus algorithm. The subnetwork simply requires minimal trust but maintains agreement throughout the whole network. As a result, the consensus algorithm executed in this network reaches low-latency and stands Byzantine failures. Tendermint [11] implements a BFT algorithm in a relaxed network model that relies on a peer-to-peer gossip protocol among nodes. It selects at most 64 validators to evaluate consensus results, avoiding too large latency in message communication.

A more direct way is to build an environment on top of the blockchain network to execute the BFT protocol. Byzcoin [12] dynamically forms consensus groups from miners in Bitcoin according to the proportion of hash power. Under the open membership, the scheme selects recently successful block miners and designates a leader to drive the BFT protocol. Solidus [13] uses PoW for leader election, which adapts the PBFT protocol to a permissionless setting. Before issuing a proposal, members submit proofs and get a high rank to join the consensus committee. Algorand [14] presents a novel mechanism based on Verifiable Random Functions (VRF) to check whether participants are selected to decide the next block in the designed Byzantine Agreement (BA) protocol.

Another way of building the environment is to divide the large network into several shards, where each shard runs the BFT protocol independently. Elastico [15] uses PoW to securely partition the network into smaller committees. After confirming the identity and forming the committees, each of them uses PBFT or other Byzantine algorithms to reach consensus. A final committee is specified to aggregate the results and outputs the final block. Chainspace [16] adopts the sharding technology to design a smart contract platform, which executes a distributed commit protocol over the shards to guarantee consistency.

Recently, an innovative idea is to integrate the chain structure into the BFT protocol. Based on the context of blockchain, the consensus protocol can realize pipeline operation. HotStuff [17] proposes a novel framework that forms a bridge between classical BFT foundations and blockchains. The solution changes the original three phases to simple votes collection from quorum replicas and then chains these phases to do more useful work.

Sync hotstuff [18] follows the same framework but ensures safety in a weaker synchronous model. It designs a simple and practical BFT protocol, which relies on latency time to discover the leader equivocation and solve Byzantine fault. Streamlet [19] simplifies the structure design of chain BFT protocol. It shows better performance when the network environment is in a good condition because of its simplicity in engineering implementation and the small delay brought by its advantages in synchronization mode. How to achieve better performance using less number of phases is also an open direction to improve HotStuff. Wendy [20] is allowed to commit operations in two-phases only using a novel aggregate signature scheme that allows leaders to prove. Marlin [21] tried to use two rounds to achieve finality while keeping linearity in view-change in the “happy” path. Another critical issue in HotStuff and its varient is that their throughput are linearly negatively correlated to latency, that is, in a network environment with high latency, the throughput will be much lower than expected. Bullshark [22] Narwhal [23] achieve zero communication overhead and high throughput in an asynchronous network condition while keep scalability and low latency.

III. Preliminaries

A. Proof of Work

In 2008, an anonymous named Nakamoto Satoshi publishes the design of Bitcoin [4], which is a peer-to-peer electronic cash system. The system allows participants (i.e. miners) to record transactions in an append-only public ledger without building trust among them. Technically, the ledger is organized in the form of a blockchain, maintained by all miners. Except for the first block (i.e. genesis block), each block contains a hash value linked to the last block. Every miner holds completely the same blockchain data on the local disk. Building such a blockchain in a public network is equivalent to reaching consensus over those miners.

The consensus of Bitcoin is to solve a difficult puzzle by providing a proof-of-work [24]. Miners create a valid block and broadcast it to the whole network, where the block is valid if its hash value less than a target value. Here Bitcoin introduces randomness and incentive to the blockchain. In terms of randomness, the puzzle is different for each miner because the locally generated block may include different transactions. While the computing time to find a solution is uncertain because the collision of the hash function is uncertain. In terms of incentive, the miner who generates a valid block on the blockchain can gain block reward and transaction fees. Thus, many miners have the motivation to mine new blocks, which reaches a consensus on the blockchain.

Nakamoto consensus adopts the longest chain rule to decide which fork is valid. Rational miners will follow and extend the longest branch they see in the network. Due to the single-chain structure, this rule is to select only a leader from all possible miners in each round to decide the next block. Inspired by Bitcoin, alternative schemes make improvements to the chain structure for high scalability. For instance, ghost protocol [25] defines the heaviest chain rule to organize the blockchain instead of a single chain, and some schemes [26], [27], [28] that use directed acyclic graph (DAG) to maximum leverage those valid blocks. From the perspective of blockchain, the chain structure is the main innovation and contribution to addressing the consensus problem.

B. Byzantine Fault Tolerance

The consensus demand is originated from the distributed system. For high availability, technology companies adopt a distributed architecture [29] to let a cluster of machines work as a large server. It can provide reliable service because the distributed cluster has strong fault tolerance. Generally, the fault refers to a crash on machines, which cannot be connected or power shutdown. While the byzantine fault makes a stronger requirement for the distributed system. It assumes the machine may be controlled by adversaries, occurring any malicious behavior including the crash. The byzantine consensus means a more reliable guarantee.

In 1982, Lamport first described the Byzantine Generals Problem [30] in a vivid way that all loyal generals must have the same decision although some traitors tell fake messages. He designs an oral message algorithm to address this problem. Generals recursively broadcast the received message and finally execute the majority function to decide the result. The oral algorithm needs exponential time complexity but can allow the byzantine fault nodes up to one-third of all nodes. Under the same ability of fault tolerance, Practical Byzantine Fault Tolerance [7] relaxes the consistency condition, requiring simply more than half of the honest nodes make the same decision. The algorithm reduces time complexity to $O(n^2)$ with two rounds of voting, making it practical to the distributed system.

Different from blockchain, byzantine consensus generally specifies a leader from the cluster of nodes. The leader drives the protocol by creating a proposal and finally decides a deterministic value among the most nodes. All nodes follow a designed procedure to communicate with others for exchanging intermediate messages. After several rounds of the vote collections, each node can confirm the majority of others obtain the same value, and thus locally committing it. For byzantine consensus, the message exchange is the core method to solve the consensus problem. More variant protocols still use the voting fashion to spread the agreed view on a proposed value.

C. Mixed Consensus

The first two kinds of consensus protocols have their own strengths and shortcomings. The first kind of consensus protocols rely on the chain structure to decide new blocks in a probabilistic way. While the second kind of consensus protocols decide a block via message exchange and thus being slow in outputting new blocks. Some improved ideas combine the two techniques for better performance. For quick confirmation, some schemes [12], [31] use Nakamoto consensus to select groups and then make BFT consensus on them. To avoid forks, some schemes [32] use BFT consensus to decide one branch after Nakamoto consensus just like a checkpoint. However, those schemes simply combine two technologies in a direct way.

They fail to leverage the chain structure of blockchain under the context of Byzantine consensus.

To the best of our knowledge, Hotstuff [17] first applies the chain structure of blockchain into the execution in Byzantine consensus. The scheme flattens the three phases of the classic PBFT protocol, letting each phase only takes a propose-vote operation. As a result, it can pipeline the block generation and block confirmation. When the current block is proposed to enter the first phase of Byzantine consensus, the previously proposed block is pushed to enter the next phase of Byzantine consensus. To ensure liveness, this scheme also takes the same operation in the view-change protocol like PBFT. Finally, a block is decided only if having three successive blocks extend to the block.

Our scheme also adopts the pipeline skill proposed in the Hotstuff but simplifies the protocol to get an optimal effect. By carefully studying the classic PBFT, we find that two-round voting is the key to locally commit blocks. The existing schemes generally lock the proposal once sending a vote to avoid divergence. We attempt to relax the requirement, which allows all replicas to change their vote but rely on the chain structure to converge consensus. Thus, we can reduce one phase of message exchange in Hotstuff without locking any proposal. Next, we present our idea from the basic state protocol to a rolling-view fashion.

IV. Overview

In this section, we present the system model, threat model, and system goals. To better illustrate how it works, we also show the message structure used in the protocol.

A. System Model

We assume that the system consists of n replicas that process transactions from users to ensure the global consistency of the operation logs. Once waiting a period of time or reaching the max batch size, the transactions are packed in the form of a block and being handled by replicas. Each replica r has a public/private key pair (pk_r, sk_r) and uses the public key as identity. All authorized replicas connect to each other to build a consensus group. The system executes a multi-phase consensus protocol over the group. In a period of time, generally called *view*, there is a replica as the leader to drive the protocol one round or more. We use an integer number v to designate the replica (v mod n) in a round-robin order.

Akin to PBFT, we make a partial synchronous network [33] assumption in the system. The messages between two honest replicas can be transmitted within Δ time after some unknown global stabilization time (GST). The network connection is established from point to point among replicas, which is reliable and authenticated by the public identity. When broadcasting a message, it means sending the same message to all replicas in a point-to-point way. We also assume the replicas run the clocks at the same rate.

B. Threat Model

We consider the system containing at most f replicas that are controlled by an adversary, of which the number does not exceed 1/3 of all replicas. For simplicity, we let $n = 3f + 1$ be the total number of replicas. The compromised replicas are Byzantine faulty with arbitrary behavior, e.g. violating the protocol or refusing a response. They eavesdrop on network communication but cannot impede and alter messages from the point-to-point transmission. Besides, the remaining non-faulty replicas honestly communicate with others and correctly execute the protocol.

C. System Goals

The system involves a protocol that consistently decides on transactions from users in chronological order. The protocol needs to satisfy the following properties:

- *Safety.* All non-faulty replicas see the same ordered operation log of transactions.
- *Liveness.* Each transaction from users is eventually committed by all non-faulty replicas.
- *Responsiveness.* Any correct leader just waits for the first $n - f$ responses to make the protocol progress.

where safety is guaranteed even when the network is asynchronous but liveness needs to rely on the synchronous network assumption. We implement such a protocol that follows the FLP Impossibility. Responsiveness means the leader does not need to wait a long time for potential conflicts to take steps.

D. Message Structure

We build a system under the weak asynchronous model for consensus. Likewise, Blockchain shows another way to reach consistency with a weaker goal. Some schemes (e.g. Hotstuff) proposed ingenious methods by using the chain structure to optimize PBFT, but we want to combine both technologies more elegantly.

In our system, we allow replicas to change minds re-voting on different proposals while designing block generation rules to eventually converge the blockchain to one single main chain. We use public-key digital signatures to authenticate the source identity of messages. Let $\langle m\rangle_r$ denote a message m signed by replica r with the hash digest of the message. To generate such a BFT-supported blockchain, we define the following types of messages.

1) Proposal message. A leader packs transactions into this message and broadcasts it as a proposal. In our protocol, a proposal is a *block* that is bound to a unique height. We denote B_k be the block at height k in view v by the following format

$$B_k = \langle \mathsf{propose}, d, v, h_{k-1}, s_{k-1}^{v}\rangle_L$$

where d denotes the packed transaction data. h_{k-1} denotes the hash value of the referred block B_{k-1}, and the genius block sets this field by $\perp$. In addition, s_{k-1}^{v} denotes the set of votes on this referred block (generally signing for its hash value). Finally, the leader of the corresponding view signs the block to make it valid.

2) Vote message. Replicas receive the proposal message (i.e. block) from the leader of the current view. After checking the block is valid, replicas sign the block hash value to vote for it. A vote for a block at height k in view v has the following format

$$\langle \mathsf{vote}, v, h_k \rangle_r$$

where h_k denotes the hash value of the valid block B_k. A block is valid if (i) its signature from the leader is correct, (ii) its predecessor block refers to valid one B_{k-1} or $\perp$, (iii) its packed transactions meet application-level requirements, and (iv) its number of valid votes for the referred block is enough.

3) Recall message. When a leader cannot collect enough votes in the current view v, this message can be used to persuade replicas in this view to vote for an alternative block. A recall message has the following format

$$\langle \mathsf{recall}, v, m \rangle_L$$

where m denotes a valid proposal message the leader wants to refer to. To create a new proposal, the leader sends the message to those replicas for getting more votes.

4) View-change message. If replicas are unable to commit any blocks in view v before the timeout, they can send this message to the next view to change the leader. A view-change message has the following format

$$\langle \mathsf{view-change}, v+1, h_k \rangle_r$$

where h_k denotes the last block replica r vote for. In fact, the view-change message has the same functionality as the vote message except for specifying a different view.

5) New-view message. Once the majority of replicas in view v send view-change messages to the leader of the next view, the leader broadcasts this message to change view. A new-view message has the following format

$$\langle \mathsf{new-view}, v+1, \pi, m \rangle_L$$

where π denotes the proof of enough view-change message in the previous view. m denotes the last proposed block the current leader wants to extend from.

V. HCA: The Hashchain-Based Consensus Acceleration

The basic idea of HCA is to build a blockchain under the BFT principle, where each block is a proposal decided by replicas. Learned from PBFT, a committed proposal needs quorum votes (i.e. correct signatures) two rounds from all replicas. We let each block refer to a predecessor block and contain enough votes for that referred block, which corresponds to a normal phase of the PBFT protocol. Benefit from the chain structure, replicas can easily verify the source of the current proposal and using the new block to commit the previous blocks.

For HCA, we do not need to lock any block because the chain structure can help to confirm the proposed block. Thus, we allow replicas to change their vote with the guide of the leader, producing a temporary fork. The system aims to increase the block height by following the longest chain rule. Replicas eventually commit the blocks of a single main chain, in which each block contains BFT validation to ensure consistency.

A. Normal State Protocol

Figure 1 shows the process of the Normal State Protocol, which runs in iterations. The normal state protocol runs in iterations. Each round is driven by a single leader of the current view.

Propose. The leader L proposes a proposal (i.e. block B_k) by broadcasting $\langle \mathsf{propose}, d, v, h_{k-1}, s^v_{k-1} \rangle_L$. The block refers to a predecessor block by its hash value $h_{k-1} = H(B_{k-1})$. Before packing transactions to this block, L needs to collect at least $2f+1$ valid votes on the referred block to make this proposal valid.

Vote. Each replica r starts a timer after sending a vote on block B_{k-1} and waits for a proposed block B_k from the leader, where block B_k extends from block B_{k-1}. If the proposed block B_k is valid, replicas send a vote $\langle \mathsf{vote}, v, h_k \rangle_r$ to the leader, where $h_k = H(B_k)$. They also reset the timer and go into the next iteration.

Revote. If the leader L cannot collect enough votes on a certain referred block because of conflict blocks, L can select one predecessor proposal m^* and persuade some replicas to change their vote by sending the recall message $\langle \mathsf{recall}, v, m^* \rangle_L$. Only if the proposal m^* is at a higher height or has a larger hash value than the last voted proposal, the replica agrees to revote on m^*.

Commit. Once replica r receives a valid proposal at a higher height than the current blockchain, for example, block B_k. It means that the block at height $k-2$ obtains confirmations from the majority of correct replicas twice. Thus, r can safely commit block B_{k-2}.

Note that the timer is reset only if the replica votes on a proposal at a new height. In the Revote phase, condition 1) requires the replica to catch up for the current state, which is the same situation as voting on a new proposal. While condition 2) simply changes vote on another proposal at the same height and does not trigger to reset the local timer. We use the timer to ensure liveness, making the protocol constantly generate and commit new blocks.

In the traditional BFT algorithms, replicas lock votes on the proposal they first receive. Replicas will conduct view change if they are aware of a failure of reaching consensus. In our protocol, we allow replicas to change votes on a conflict block in the non-blocking Revote phase. Due to the chain structure of blocks, such a conflict is equivalent to forks in the blockchain. Our idea is to design a rule to lead replicas to fast make a convergence to one of the forks. Only a valid block with the rank of a higher height to a larger hash value can trigger the replicas to change votes. Eventually, there is one main chain that records the votes from replicas, and the blocks on the main chain can be safely committed (we show the detailed proof below). In the current view, the leader broadcasts a recall message for collecting enough votes on one fork. If the leader does not propose a new block to extend the blockchain before the timeout, replicas will maintain the current block and move to the next view for expecting to solve the conflict.

B. View-Change Protocol

The view-change protocol ensures liveness while guarantees safety across views. If the leader of the current view cannot drive

Let v be the current view number and replica L be the leader of the current view. After sending a vote on block B_{k-1} to the leader L, a replica r starts a timer and runs the following protocol.

1) **Propose.** If replica r is the leader L, upon receiving at least $2f+1$ signatures $\langle \mathsf{vote}, v, h_{k-1}\rangle_r$, construct the aggregate signature $s_{k-1}^v = \sum \langle \mathsf{vote}, v, h_{k-1}\rangle_r$, batch the transactions with the form of data d, and broadcast $\langle \mathsf{propose}, d, v, h_{k-1}, s_{k-1}^v\rangle_L$ where $h_{k-1} = H(B_{k-1})$.
2) **Vote.** Before the timeout, upon receiving a valid block $B_k = \langle \mathsf{propose}, d, v, h_{k-1}, s_{k-1}^v\rangle_L$ from L where $h_{k-1} = H(B_{k-1})$, send $\langle \mathsf{vote}, v, h_k\rangle_r$ to L where $h_k = H(m_k)$ and reset the timer.
3) **(Non-blocking) Revote.** Let m' be the last block the replica votes for. Upon receiving $\langle \mathsf{recall}, v, m^*\rangle_L$ from L, check if m^* is valid and satisfies i) $height(m^*) > height(m')$, then reset the timer and resend $\langle \mathsf{vote}, v, H(m^*)\rangle_r$ to L , or ii) $H(m^*) > H(m')$ at the same height, then resend $\langle \mathsf{vote}, v, H(m^*)\rangle_r$ to L.
4) **Commit.** When receiving a valid block B_k and this block at the current height emerging for the first time, commit the block at height $k-2$.

Fig. 1. The normal state protocol.

Let L and L' be the leaders of views v and $v+1$, respectively. While waiting for the next block at a higher height, a replica r with a local timer runs the following steps.

i) **Lock and change view.** If the local timer expires in view v, lock the current block by rejecting any message from L, and send $\langle \mathsf{view\text{-}change}, v+1, h_k\rangle_r$ to L' while start a new local timer, where h_k refers to the last valid block the replica r votes on.
ii) **New view.** The new leader L' collects at least $2f+1$ view-change messages to construct a proof π, picks a referred block m^* in all those messages by the rank from height to hash value, and broadcasts $\langle \mathsf{new\text{-}view}, v+1, \pi, m^*\rangle_{L'}$.
iii) **First vote.** Upon receiving a valid new-view message from L', check if the contained block m^* has the rank equal to or higher than the locked block, send $\langle \mathsf{vote}, v+1, H(m^*)\rangle_r$ to the new leader L' and enter view $v+1$ by resetting the timer.

Fig. 2. The view-change protocol.

the consensus protocol to increase the height of blockchain in a period of time, it triggers replicas to change the current view for expecting to reach consensus in the next view. The process of the protocol is shown in Fig. 2. In step i), replicas lock the last block they vote on after the timeout and send a view-change message to the leader of the next view. A new local timer is created to wait for the response. If this timer expires again, replicas maintain the locked block to ask help for the next view of the current view, and so on. After every failure on this step, replicas will increase the waiting time to avoid frequently change the view.

The new leader constructs a valid new-view message to change the view in step ii). The new-view message needs a proof and a referred block. The proof is consists of at least $2f+1$ view-change messages from the previous view. The referred block is selected by the new leader according to the rank strategy, which is the same as the Revote phase in the normal state protocol. Likewise, the new leader can choose a blockchain fork by broadcasting the new-view message to inform replicas of entering a new view.

A new-view message can let a replica move to the next view before the timeout. If the new-view message is valid, which means the contained proof is verified and the rank of the contained block is equal to or higher than the locked block, the replica enters the new view in step iii) by sending the first vote on the contained block. Note that the new leader may also fork the blockchain by using the same proof to issue different new-view messages. Nevertheless, the replica who starts the view-change step still successfully enters the next view as long as seeing a valid new-view message. This operation provides only one chance to change the vote but remains the conflict be solved in the normal state protocol.

C. Safety, Liveness and Responsiveness of the HCA Protocol

A block B_{k-2} is safely committed only if the leader successfully issues a block B_k that extends from block B_{k-2}. Akin to PBFT, this block B_{k-2} obtains at least $2f+1$ valid votes from replicas in two consecutive rounds.

Lemma 1. If the Revote phase does not occur, then for any valid block B_r, B_s in which $h(B_{r-1}) \neq h(B_{s-1})$, we have that B_r, B_s must not be commited at the same height.

Proof. Suppose B_r, B_s both have been commmited in view v at height k. Because a valid block can be commited only with $n-f = 2f+1$ votes for it, there must be a honest replica who voted twice at height $k-2$. This is impossible, because if the Revote phase does not occur, a replica can only vote once for Vote phase in each view. □

Lemma 2. Any valid block B_1, B_2 that have collected at least $2f+1$ valid votes at the same height cannot be both commited.

Proof. To show a contradiction, suppose B_1, B_2 both have been commmited in view v at height k. By the lemma 1, it means that the Revote phase must have occured at height $k-2$

(e.g., when the malicious leader has collected 2f+1 votes for block B_1, she can make a replica revote for the block B_2 that has collected $2f$ votes through the Revote phase, which is feasible in theory). Because $height(B_1) = height(B_2)$, the precondition for completing this operation is $H(B_2) > H(B_1)$ as the replicas only change their vote to the block at a higher height and a larger hash value. So after entering view $v + 1$, the leader cannot collect valid first votes from the replicas that voted for block B_1 in view v, so the block at height $k + 1$ can only be extended from block B_2. Therefore, due to the honest replicas will not change votes for the block at a lower height. Thus, the leader cannot collect more than $2f$ votes on the block at less than height $k + 1$ once proposing a new valid block at height $k + 2$, that is, the length of any fork will be less than 2 blocks. □

Theorem 3 (Safety of the HCA Protocol). If B_1, B_2 are at the same height, then they cannot be both committed, each by a correct replica. There will always be only one main chain.

Proof. By lemma 1, once issuing a new block at height k, the leader cannot create a new valid block to substitute the current branch. By lemma 2, it means that the existing forks started at height $k - 2$ are abandoned forever, the length of any fork will be less than 2 blocks, and there is one main chain. As described in the protocol, replicas only commit the block at depth 2, which certainly belongs to the main chain. □

Lemma 4. If an honest replica receives a valid new-view message from new leader L', check if the contained block $H(m^*)$ is equal to h_k in view v at height k, then at least $f + 1$ honest replicas voted for m^*.

Proof. Suppose replica r at height k receives a valid new-view message from new leader L' which contained block $H(m^*)$ is equal to h_k. Then, $(n - f)$ votes were cast for block m^* in which $f + 1$ were from honest replicas. □

Lemma 5. After GST, there exists a time period T that all honest replicas remain in the same view v. An honest leader in such a view can always broadcast a new block at a higher height.

Proof. When starting a new view, the leader collects at least $2f + 1$ view-change messages and calculate a proof π before broadcasting a new-view message. Suppose among all messages from replcas, the block m^* has the highest hash value. By lemma 4, at least $f + 1$ honest replicas voted for m^*, and have already sent them to the leader in their view-change messages. Leaders get m^* from at least one of their view-change messages and broadcast it in a new-view message. Then when replicates enter the First vote phase, the locked block is updated, the timer is reset, and the next view is successfully entered. By the assumption, all correct replicas are synchronized in their view and the leader is nonfaulty. Then, after the leader assembles a valid s^v_{k-1} for this view, all replicas will vote in the Vote phase, leading to form a new block. After GST, the duration T for the phase to complete is of bounded length. □

Theorem 6 (Liveness of the HCA Protocol). All honest replicas keep committing new blocks in the correct view.

Proof. In the normal state protocol, an honest leader can always get enough votes from other honest replicas to issue a new block at a higher height. A malicious leader may impede the progress by proposing other blocks at the same height. As a result, replicas fail to commit the previous blocks to reach termination.

When incurring a malicious leader or terrible network delay, it may fail to issue a new block at a higher height to commit the previous blocks. Each replica sets a local timer to wait for the new proposed blocks, ensuring the protocol proceeding. Once the waiting is timeout, replicas change the view to activate the next leader. By lemma 5, the protocol merely changes the context but remains the snapshot state. The new leader still executes the normal state protocol to drive the consensus. As long as there exists an honest leader in the subsequent views, the leader can always issue a valid block at a higher height to commit new blocks. □

Theorem 7 (Responsiveness of the HCA Protocol). When network becomes synchronous, HCA can generate the honest leader to push the protocol to reach consensus within the actual value of network delay rather than the maximum value.

Proof. HCA has introduced the revoting mechanism to enable honest leaders to change the decisions of some replicas when the current view cannot collect enough votes for any block, so that they can reach an agreement on a proposal before timeout in synchronized networks, thus achieving Optimistically Responsive. □

VI. The Optimized HCA

In the basic scheme, we design two protocols to construct a blockchain for BFT-based consensus, in which the normal state protocol ensures safety while the view-change protocol ensures liveness. Different from PBFT, we only use proposal and vote messages to build the blockchain but introduce recall message for solving potential forks and converging them to one branch. Interestingly, the appended blocks let previous ones safely committed.

The proposed view-change protocol is also different from PBFT. Instead of synchronizing pending requests among replicas, we just transmit the last block replicas vote on from the current view to the next view. From the layer of messages, the view-change message is similar to the vote message while the new-view message acts like the recall message while attaching a proof. This proof has the same function as aggregated votes in the proposal. For simplicity, we can integrate two protocols into one novel consensus protocol in a rolling fashion, which streamlines the block generation.

A. State Protocol

The process of rolling state protocol is shown in Fig. 3, which also runs in iterations. Each round corresponds to a certain view, of which the leader collects votes from the last view to start the protocol.

Propose. The leader L proposes a proposal (i.e. block B_k) by broadcasting $\langle \text{propose}, d, v, h_{k-1}, s^v_{k-1} \rangle_L$. The block refers to a predecessor block created in the previous view via a hash value $h_{k-1} = H(B_{k-1})$. Before packing transactions to this block, L is obliged to collect at least $2f + 1$ votes on the referred block from replicas in the previous view to make this block valid. Specifically, the leader can choose a branch to extend.

Let v be the current view number and replica L be the leader of the current view. When entering view v by sending a vote on block B_{k-1} to the leader L, a replica r starts a timer and runs the following protocol.

1) **Propose.** If replica r is the leader L, upon receiving at least $2f+1$ signatures $\langle \mathsf{vote}, v, h_{k-1}\rangle_r$, construct the aggregate signature $s^v_{k-1} = \sum \langle \mathsf{vote}, v, h_{k-1}\rangle_r$, batch the transactions with the form of data d, and broadcast $\langle \mathsf{propose}, d, v, h_{k-1}, s^v_{k-1}\rangle_L$ where $h_{k-1} = H(B_{k-1})$.
2) **Vote.** Before the timeout, upon receiving a valid block $B_k = \langle \mathsf{propose}, d, v, h_{k-1}, s^v_{k-1}\rangle_L$ from L where $h_{k-1} = H(m_{k-1})$, send $\langle \mathsf{vote}, v+1, h_k\rangle_r$ to the leader of view $v+1$ where $h_k = H(m_k)$. Otherwise, after the timeout send $\langle \mathsf{vote}, v+1, h_{k-1}\rangle_r$ to the leader of view $v+1$. Move to the next view.
3) **(Non-blocking) Revote.** Let m' be the last block the replica votes for. Upon receiving $\langle \mathsf{recall}, v, m^*\rangle_L$ from L, check if m^* is valid and satisfies i) $height(m^*) > height(m')$, or ii) $view(m^*) > view(m')$ at the same height, or iii) $H(m^*) > H(m')$ at the same height in the same view, then resend $\langle \mathsf{vote}, v, H(m^*)\rangle_r$ to L.
4) **Commit.** When receiving a valid block B_k and such this block at the current height emerging for the first time, commit the block with height $k-2$.

Fig. 3. The rolling state protocol.

Vote. Each replica moves to the next view after sending a vote on the block issued in the current view. Once entering a new view, the replica r starts a timer and waits for a valid block from the new leader. If the block is received before the timeout, r will vote on it by sending $\langle \mathsf{vote}, v+1, h_k\rangle_r$ to the leader. Otherwise, r stays on the last block and sends $\langle \mathsf{vote}, v+1, h_{k-1}\rangle_r$ to the next leader. In any case, the vote always works on the block in the current view but be sent to the next view.

Revote. When the leader L of view v collects multiple different votes from the previous view, making it fails to create a new valid block. L can select one of those referred blocks m^* according to the rank of its height and belonging view. Then, L sends $\langle \mathsf{recall}, v, m^*\rangle_L$ to persuade some replicas to change their vote to the selected block. Note that this step is only executed by the current leader, and it works before the timeout in the current view.

Commit. Once the replica r successfully enters view $v+1$ by voting for the block B_k at a higher height, it means that the referred block has already got at least $2f+1$ votes. Meanwhile, the block before this referred block obtains confirmations from the majority of replicas twice. Thus, r can commit the block at height $k-2$.

In the rolling state protocol, replicas send votes to the leader of the next view, which forces them to change view. When entering a new view, a timer is still set to ensure liveness. If the current leader cannot propose a valid block at a higher height, a vote on the same block is re-created and sent to the next leader. While a new valid block can end the current view in advance, triggering replicas to move into the next view and reset the timer.

Likewise, replicas choose a branch to follow when encountering forks at a certain height of blockchain. Different from the normal state protocol, the rolling fashion restrains each leader to propose at most one block in its view. Apart from the block height and hash value, the view number is also required to locate a unique block. In a possible case, the current leader creates a valid block but fails to spread the block before the timeout. When the next leader issues a valid block, replicas will receive two blocks at the same height but in different views. We redefine the rank of blocks in the Revote phase, letting replicas prefer the block in a higher view number when facing multiple blocks at the same height. The stale block will be abandoned although it is correct. By adopting this way, the leaders in adjacent views compete to broadcast a new block and thus pushing the protocol forward.

B. Leader Strategy

In the rolling state protocol, a leader is allowed to issue a new block if and only if collecting enough votes from the previous view. It constraints that each block in the longest chain must contain expected witnesses from the group of replicas, and thus ultimately reaching consensus. Apart from the proposal and vote message, we use a recall message, which is unrelated to building blocks, to let the leader persuade replicas to change their votes.

When entering view v, the leader may encounter two situations in collecting votes from the previous view. As shown in Fig. 4, each view has only one leader to propose blocks while replicas always follow the longest chain. The first situation is caused by the malicious leader of view 2, which generates two valid blocks B'_3 and B^*_3. The leader of view 3 may receive different votes and fail to construct a valid block. Due to the hash value of block B^*_3 is larger than that of block B'_3, the leader's strategy is to multicast a recall message containing block B^*_3. The non-blocking Revote phase can let the leader get more votes on block B^*_3. The dotted box in blue shows the branch selection process.

The second situation is affected by communication quality. As shown in Fig. 4, the leader of view 4 proposes a valid block B'_5, but most replicas do not receive this block before the timeout because of the poor network quality. Thus, they send the vote on block B_4 to the leader of view 5, which generates a valid block B^*_5 at the same height as block B'_5. The leader of view 6 may receive different votes and fail to issue its own block. Due to the view number of block B^*_5 is larger than that of block B'_5, the leader's strategy is to multicast a recall message containing block B^*_5. The non-blocking Revote phase lets the leader get more votes on block B^*_5. The dotted box in green shows the branch selection process.

C. Safety, Liveness and Responsiveness of the Optimized HCA Protocol

A block B_{k-2} is safely committed in view v only if the leader of this view successfully issues a block at height k that extends

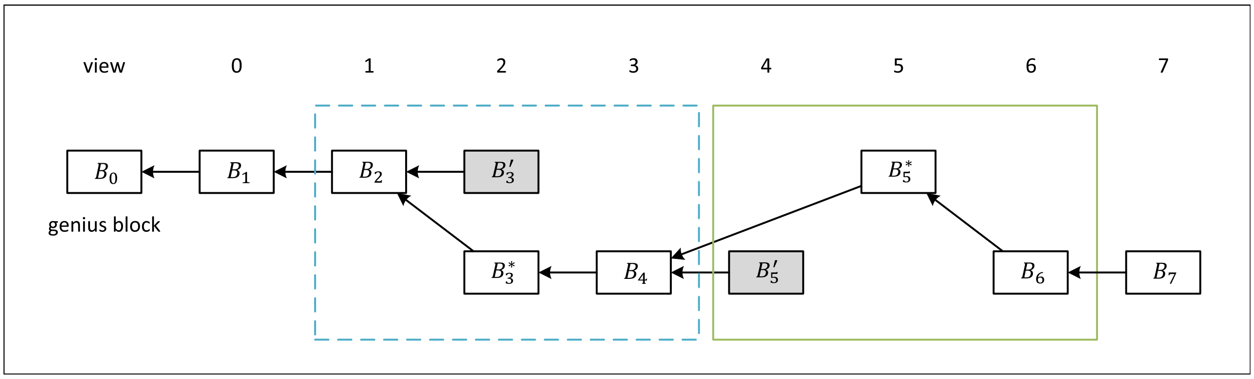


Fig. 4. Two possible situations of blockchain fork in the rolling state protocol by following the longest chain rule. The first situation shown in the dotted box in blue is caused by a malicious leader, and the next leader selects the branch by extending the block with a higher hash value. The second situation shown in the dotted box in green is caused by the poor network, and the next leader selects the branch by extending the block with a higher view number.

from block B_{k-2}. It means that the block B_{k-2} obtains at least $2f+1$ votes in two rounds. Akin to PBFT, a replica can locally commit the block B_{k-2} once getting two blocks confirmation.

Lemma 8. If a new valid block at height k is proposed in view v, then the leaders since view $v+1$ can no longer collect at least $2f+1$ votes on the block at less than height $k-1$.

Proof. Intuitively, because of the proposed block at height k, we have at least $2f+1$ replicas who sign a block at height $k-1$ when entering view v. Thus, at least $f+1$ honest replicas have verified the block at height $k-1$, which extends from the same block at height $k-2$. According to the Revote phase in the protocol, those honest replicas who vote on the block at height $k-1$ do not change their vote to blocks at a lower height. The remaining replicas cannot constitute $2f+1$ votes, and thus the leaders after view v are unable to issue a block that extends from the block at height $k-2$ and even lower.

Suppose for contradiction that the leaders since view $v+1$ collect at least $2f+1$ votes on the block at less than height $k-1$. Without loss of generality, let the leader of view $v+1$ obtains both a valid block at height k and $2f+1$ votes on the block at height $k-2$. In extreme case, we have a set R of $f+1$ honest replicas who vote on the block at height $k-1$ in view $v-1$. It joins with other f malicious replicas to contribute the valid block at height k. There also exists a set R' of f honest replicas who vote on the block at height $k-2$ in view $v-1$. When entering view v, we have $2f+1$ votes on the block at height $k-2$. This implies that a replica r in set R also votes on the block at height $k-2$. However, the protocol restrains replicas to only vote on the block at a higher height. Whatever the replica r receives the new block in view v, it can only vote on the block at height $k-1$. Thus, the leaders since view $v+1$ can no longer get at least $2f+1$ votes on the block at lower than height $k-1$. □

Lemma 9. There is only one main chain of blocks while the length of any fork is less than 2 blocks.

Proof. In any view, the corresponding leader either does nothing or issues a block. For the first situation, replicas will skip this view once the local timer expires and continue to vote in the next view. For the second situation, the leader collects votes.

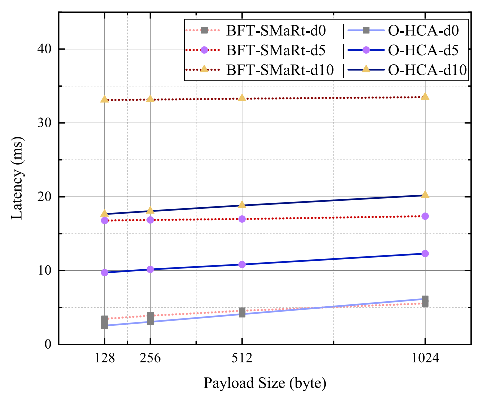
We suppose that the current block with height k is proposed before the current view. By Lemma 8, the leader can only collect enough votes on the block at height $k-1$ or k, where the blocks are denoted by B_{k-1} and B_k respectively. If the leader extends the block B_{k-1} using a new block B_k', then a fork is created. The leader essentially replaces the original block B_k with B_k'. Although both the block are at the same height, the view number of B_k' obviously larger than that of B_k. According to the Revote phase in the protocol, honest replicas will change vote to blocks at the same height but a higher view number. Once the block B_k' is issued, which means that there are at least $f+1$ honest replicas who change vote to the current view. That is to say, the original block B_k is relinquished and can no longer collecting enough votes on it. The new block comes into the main chain, waiting to be substituted or be extended. Thus, any forks in the main chain have less than 2 blocks. □

Theorem 10 (Safety of the Optimized HCA Protocol). No two replicas commit different blocks at the same height.

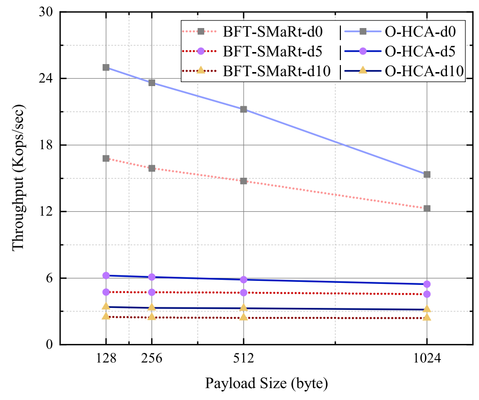
Proof. As described in the protocol, replicas only commit the block with depth 2. To commit different blocks at the same height, there exists a fork from the main chain having at least two successor blocks. By Lemma 9, any fork is less than 2 blocks, and thus committed blocks certainly belong to the unique main chain. □

Lemma 11. After GST, there exists a time period that all honest replicas remain in the same view. A honest leader in such a correct view can always broadcast a new block at a higher height.

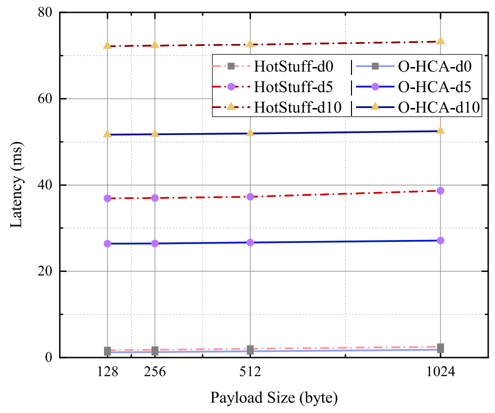
Proof. Due to the introduction of the Revote phase in a non-blocking way, the leader of each view can synchronize the current blockchain with replicas. The replicas only change their vote to the block at a higher height and a larger view number, resulting in a unique main chain in the protocol. Moreover, any malicious leader in the previous view may propose several blocks to make forks in the blockchain. The current leader chooses the block with the largest hash value from the previous view. According to the Revote phase in the protocol, honest replicas will change vote to blocks with a larger hash value at the same height and in the same view. If the leader is honest,



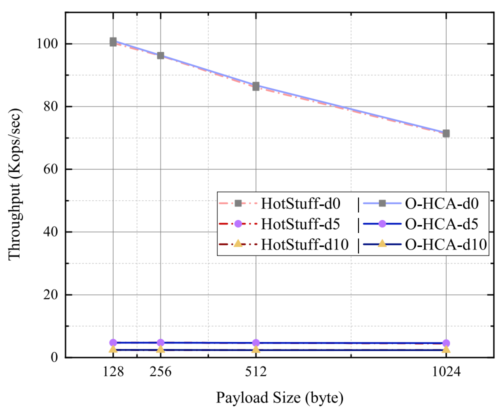
(a) Latency (Optimized HCA vs BFT-SMaRt)



(b) Throughput (Optimized HCA vs BFT-SMaRt)



(c) Latency (Optimized HCA vs HotStuff)



(d) Throughput (Optimized HCA vs HotStuff)

Fig. 5. Performance with different choices of payload size under 4 replicas limiting the batch size of 50.

it can always collect $2f+1$ votes from other honest replicas. Once obtaining enough votes on a predecessor block, the correct leader is able to construct a new valid block and broadcast it at a higher height. □

Theorem 12 (Liveness of the Optimized HCA Protocol). All honest replicas keep committing new blocks in the correct view.

Proof. Each view has only one leader in a round-robin way, and the leader drives the consensus protocol by proposing a new block. Replicas start a local timer after entering a view. When the timer counting down to zero or receiving a valid block from the leader of the current view, replicas change the view and run the protocol again in the next view. By Lemma 11, the honest leader can always issue a new block at a higher height to push replicas making progress. That is because 1) this leader generates a new block by following the longest chain, and 2) the proposed block can be uniquely received by all replicas. Upon passing over such a correct view, the proposed block certainly gets enough votes and thus being added to the main chain. There is only one valid block in each view, the majority of replicas will change to vote on the block with priority rank, which is impossible substituted by other blocks at the same height. As long as a new block at a higher height is generated, the replicas can keep committing new blocks. □

Theorem 13 (Responsiveness of the Optimized HCA Protocol). When network becomes synchronous, the optimized HCA can generate the correct leader to push the protocol to reach consensus within the actual value of network delay rather than the maximum value.

Proof. Due to the introduction of the Revote phase in a non-blocking way, the honest leaders can reach agreements on proposals before timeout in synchronized networks under the actual network delay by making the replica re-vote when they cannot collect enough votes. Once at least $2f+1$ votes are collected through revoting, the leader can broadcast the block to replicas, and then prepare to end the current view, achieving Optimistically Responsive. □

VII. Theoretical Analysis of the Performance

In this section, we examine two strongly relevant concensus protocols, namely, PBFT and HotStuff, to compare with the Optimized HCA protocol. The reason is that the proposed protocol

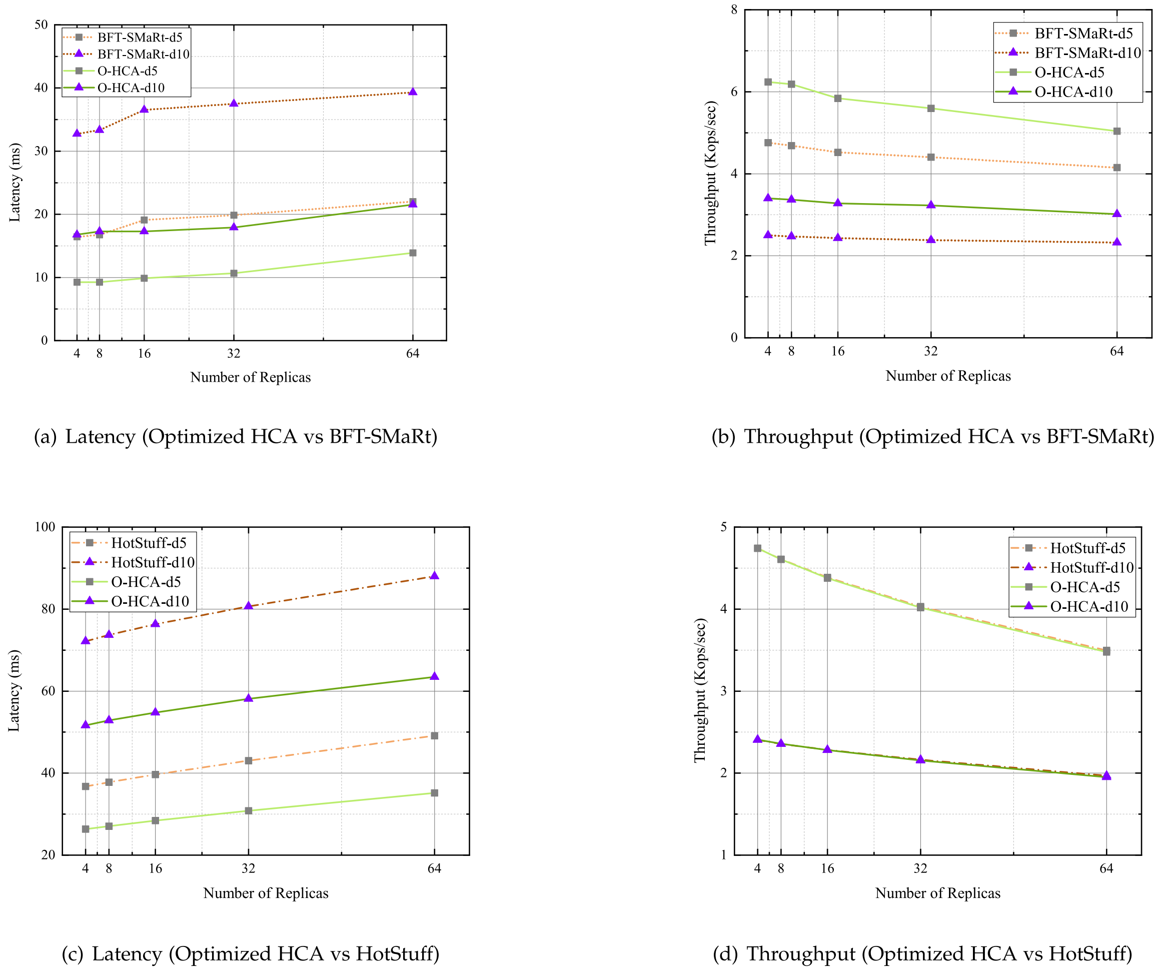


Fig. 6. Scalability under different number of replicas setting payload size to 0.

adopts two rounds of voting to decide a block, which is the same as PBFT, and it also follows a rolling view to pipeline the protocol phases, which is similar to the chained HotStuff. The theoretical comparison explicitly shows the advantage of the proposed protocol.

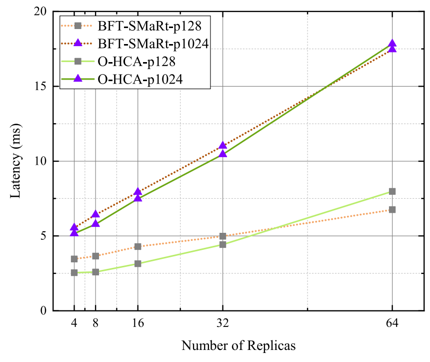
PBFT. This protocol contains three phases, pre-prepare, prepare, and commit. We have n replicas running the protocol. The first phase broadcasts a block proposed by the specific leader, where the communication complexity is $O(n)$. The last two phases broadcast vote among all replicas, where the communication complexity is $2 \cdot O(n^2)$. Thus, the protocol has $O(n^2)$ in total communication complexity. Every three phases decide a block, and we let each phase consume unit time. When there is a total execution time of p, we can get $p/3$ decided blocks.

HotStuff. This protocol designs four phases, prepare, precommit, commit, and decide. But these phases are chained by the chain structure of blockchain. We also have n replicas maintaining the blockchain. Each phase just broadcasts a block by the current leader and collects quorum certificates in the next phase, where the communication complexity is $3 \cdot O(n)$. Thus, the protocol has $O(n)$ in total communication complexity. A new proposed block decides the block at the depth 3 of blockchain, and we let each block need unit time to spread. If the total execution time is p, we can get $p - 3$ decided blocks.

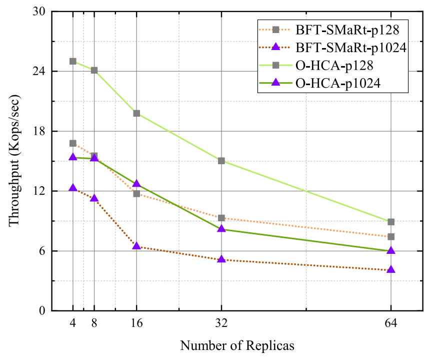
HCA. Our protocol simplifies the procedure in HotStuff. It just includes propose and vote phases to build a block. Two extended block confirmations can decide the current block. Due to the non-blocking recall phase, we can reduce one broadcast operation in the protocol, where the communication complexity is $2 \cdot O(n)$. Thus, our protocol also has $O(n)$ in total communication complexity. Only the block at depth 2 can be safely committed, and we still set unit time for spreading one block. The total execution time p can let us get $p - 2$ decided blocks.

VIII. Experimental Analysis of the Performance

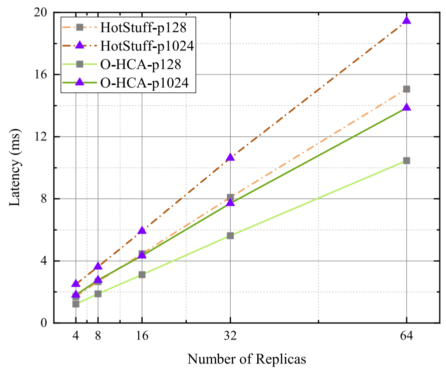
In this section, we evaluate the proposed Optimized HCA protocol based on a state-of-art system BFT-SMaRt [34], an open-source library implementing BFT state machine replication (SMR) in Java. We have forked the code from GitHub repository (https://github.com/bft-smart/library) and rewrite the consensus module using the logic of our protocol. Then, we examine the performance and scalability of Optimized HCA and BFT-SMaRt, respectively, with the same parameters to show our advantages in the blockchain consensus. In addition, we have



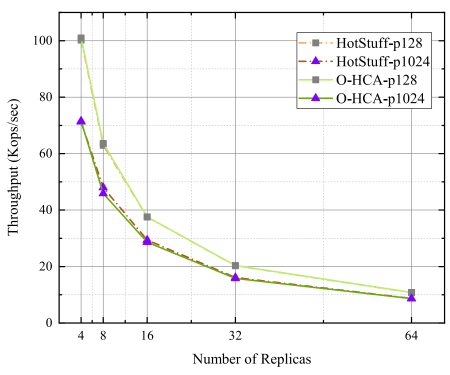
(a) Latency (Optimized HCA vs BFT-SMaRt)



(b) Throughput (Optimized HCA vs BFT-SMaRt)



(c) Latency (Optimized HCA vs HotStuff)



(d) Throughput (Optimized HCA vs HotStuff)

Fig. 7. Scalability under different number of replicas setting network delay to 0.

also implemented the Rust version of Optimized HCA separately, and replaced the comparison protocol with HotStuff to conduct the above experiment again to illustrate the advantages of our protocol in terms of communication complexity as we described in Section VII.

A. Setup

The experiments are conducted in Aliyun's cloud computing services. We run each replica in a single virtual machine (VM), using ECS.g6.3xlarge instance with 48 GiB memory and 12 vCPUs. All CPU cores are supported by Intel Xeon(Cascade Lake) Platinum 8269CY processors, which sustains a clock 2.5 GHz and speeds up to 3.2 GHz. We installed Ubuntu 18.04 operating system and deployed jre1.8 and rustc 1.67.0-nightly to run the program. Each replica connects others to build a consensus network, while the client runs in another VM with the same configuration and creates multiple threads to concurrently send requests to the network.

The BFT-SMaRt library provides two micro-benchmark programs named ThroughputLatencyServer and ThroughputLatencyClient to measure the latency and throughput. The server-side program records the time overhead in each phase of the protocol, while the client-side program outputs the average response time of requests. We obtained the consensus latency results from servers and computed the request throughput results from the client's view. We also implemented a similar mechanism under Rust to better conduct the experiment between HotStuff and Optimized HCA. To make credible comparisons, we used the mentioned test programs and adopted the same parameters to conduct the experiments. We let the client create multiple threads where each thread constantly sends requests. The execution latency of the protocols are recorded and the average results are output from 100 records.

B. Performance

We first measure the base performance for Optimized HCA, BFT-SMaRt and HotStuff, deploying 4 replicas in a configuration that tolerates a single Byzantine fault, i.e. $f = 1$. The consensus latency is shown in Fig. 5(a) and (c) with the unit millisecond, while the request throughput is shown in Fig. 5(b) and (d) with the unit thousand operations per second. The label BFT-SMaRt-d0, O-HCA-d0 and HotStuff-d0 denotes experimental results of BFT-SMaRt, Optimized HCA and HotStuff, respectively, without any extra network delay (represented by 'd').

Fig. 5(a) and (b) show that our Optimized HCA protocol has lower latency and higher throughput than BFT-SMaRt when payload size less than 512 bytes. But with payload size increasing,Optimized HCA seems to present poor performance faster. We reviewed the code and found the reason is that BFT-SMaRt makes a lot of engineering optimization and also simplifies the implementation of PBFT protocol. It realizes less communication cost in the message exchange, and thus being slightly affected by payload size. Fig. 5(c) and (d) show that compared with HotStuff, Optimized HCA can achieve lower latency while maintaining even a slightly higher request throughput.

To present the advantages of Optimized HCA, we use the traffic control tool tc in Linux to simulate network delay. As shown in Fig. 5, we add 5 ms and 10 ms delay to the specific network interface and repeat the above experiments. The results prove that Optimized HCA outperforms BFT-SMaRt and HotStuff because of simplified phases. As the network delay increase, the consensus latency of Optimized HCA is far lower than that of BFT-SMaRt and HotStuff. The request throughput of them has a similar effect under the same situation.

C. Scalability

Next, we measure the scalability of Optimized HCA and compare it to BFT-SMaRt and HotStuff, still presenting the consensus latency and corresponding request throughput. The number of replicas is set from 4 to 64, where the system can tolerate Byzantine faults $f = 1, 2, 5, 10, 21$. We conduct the experiments under the effect of network delay and payload size, respectively.

As shown in Fig. 6, we let the network delay be 5 ms and 10 ms to run the test program, where the payload is set to 0. The experimental results show that Optimized HCA has lower consensus latency than BFT-SMaRt and HotStuff. Fig. 6(a) and (c) further indicates that the gap of latency will be larger as the network delay increasing. Fig. 6(b) also shows the better performance of Optimized HCA in requests throughput than BFTSMaRt. However, compared with HotStuff, Optimized HCA has no obvious advantage in throughput, as shown in Fig. 6(d). Conversely, we set the network delay to 0 while letting the payload size be 128bytes and 1024bytes. The experimental results are shown in Fig. 7. Optimized HCA has advantage in consensus latency than BFT-SMaRt when the number of replicas is less than 32, as shown in Fig. 7(a). However, The increase in the number of replicas makes Optimized HCA present temporary bad performance. Fig. 7(b) presents the corresponding request throughput, which indicates the better performance of Optimized HCA in requests throughput than BFTSMaRt. Fig. 7(c) and (d) show that as the number of replicas increases, the latency advantage of Optimized HCA becomes more and more obvious when compared with HotStuff, and its request corresponding throughput keep pace with HotStuff.

The results basically match the performance analysis. Due to both HotStuff and BFT-SMaRt has three phases and BFT-SMaRt involves one more round of broadcast, which are more sensitive for network delay. Thus, Optimized HCA shows better performance at both latency and throughput. The payload size seems like an obstacle for Optimized HCA but not for BFT-SMaRt. The reason is that the latter does a lot of code optimization and also simplifies the implementation of PBFT, which leads to efficient communication. This reason can also be seen from the experiment between HotStuff and Optimized HCA, because neither of them has been optimized in this respect. On the whole, Optimized HCA indeed enhances consensus performance. We verify that it is meaningful to integrate chain structure into traditional consensus protocol.

IX. Conclusion

In this paper, we propose a new consensus protocol that integrates the features of chain structure from Nakamoto consensus and message exchange from Byzantine consensus, and builds a BFT-supported blockchain. In this blockchain, each block contains a quorum of valid votes, making the block at depth 2 be safely committed. The extra revote phase solves temporary forks, in which the leader can collect more votes by sending a recall message to select one branch for convergence. To construct such a blockchain, we first design a normal state protocol and a view-change protocol. The former protocol creates the blockchain and guarantee the safety of consensus, while the latter protocol transfers the state from a failure view to the next view for liveness. Then, we also present a rolling state protocol that integrates the view-change part for simplifying the original HCA protocol. We prove the safety, liveness and responsiveness, respectively. Evaluation proves that the proposed protocol has good performance.

References

[1] Z. Zheng, S. Xie, H. Dai, X. Chen, and H. Wang, “An overview of blockchain technology: Architecture, consensus, and future trends,” in *Proc. IEEE Int. Congr. Big Data*, 2017, pp. 557–564.

[2] G.-T. Nguyen and K. Kim, “A survey about consensus algorithms used in blockchain,” *J. Inf. Process. Syst.*, vol. 14, no. 1, pp. 101–128, 2018.

[3] S. Bano et al., “SOK: Consensus in the age of blockchains,” in *Proc. 1st ACM Conf. Adv. Financial Technol.*, 2019, pp. 183–198.

[4] S. Nakamoto and A. Bitcoin, “A peer-to-peer electronic cash system,” 2008. [Online]. Available: https://bitcoin.org/bitcoin.pdf

[5] A. M. Antonopoulos, “Mastering bitcoin: Programming the open blockchain,” Philadelphia, PA, USA: O’Reilly Media, Inc, 2017.

[6] V. Gramoli, “From blockchain consensus back to Byzantine consensus,” *Future Gener. Comput. Syst.*, vol. 107, pp. 760–769, 2020.

[7] M. Castro et al., “Practical Byzantine fault tolerance,” *Operating Syst. Des. Implementation*, vol. 99, pp. 173–186, 1999.

[8] E. Buchman, “Tendermint: Byzantine fault tolerance in the age of blockchains,” Ph.D. dissertation, 2016.

[9] G. Wood et al., “Ethereum: A secure decentralised generalised transaction ledger,” *Ethereum Project Yellow Paper*, vol. 151, pp. 1–32, 2014.

[10] D. Schwartz et al., “The ripple protocol consensus algorithm,” *Ripple Lab. Inc. White Paper*, vol. 5, no. 8, 2014.

[11] E. Buchman, J. Kwon, and Z. Milosevic, “The latest gossip on BFT consensus,” 2018, *arXiv:1807.04938*.

[12] E. K. Kogias, P. Jovanovic, N. Gailly, I. Khoffi, L. Gasser, and B. Ford, “Enhancing bitcoin security and performance with strong consistency via collective signing,” in *Proc. 25th {Usenix} Secur. Symp.*, 2016, pp. 279–296.

[13] I. Abraham, D. Malkhi, K. Nayak, L. Ren, and A. Spiegelman, “Solidus: An incentive-compatible cryptocurrency based on permissionless Byzantine consensus,” 2016, *arXiv:1612.02916*.

[14] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, “Algorand: Scaling Byzantine agreements for cryptocurrencies,” in *Proc. 26th Symp. Operating Syst. Princ.*, 2017, pp. 51–68.

[15] L. Luu, V. Narayanan, C. Zheng, K. Baweja, S. Gilbert, and P. Saxena, "A secure sharding protocol for open blockchains," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2016, pp. 17–30.

[16] M. Al-Bassam, A. Sonnino, S. Bano, D. Hrycyszyn, and G. Danezis, "Chainspace: A sharded smart contracts platform," 2017, *arXiv: 1708.03778*.

[17] M. Yin, D. Malkhi, M. K. Reiter, G. G. Gueta, and I. Abraham, "HotStuff: BFT consensus with linearity and responsiveness," in *Proc. ACM Symp. Princ. Distrib. Comput.*, 2019, pp. 347–356.

[18] I. Abraham, D. Malkhi, K. Nayak, L. Ren, and M. Yin, "Sync hotstuff: Simple and practical synchronous state machine replication," in *Proc. IEEE Symp. Secur. Privacy*, 2020, pp. 106–118.

[19] B. Y. Chan and E. Shi, "Streamlet: Textbook streamlined blockchains," in *Proc. 2nd ACM Conf. Adv. Financial Technol.*, 2020, pp. 1–11.

[20] N. Giridharan, H. Howard, I. Abraham, N. Crooks, and A. Tomescu, "No-commit proofs: Defeating livelock in BFT," *Cryptol. ePrint Arch.*, 2021. [Online]. Available: https: //eprint.iacr.org/2021/1308

[21] X. Sui, S. Duan, and H. Zhang, "Marlin: Two-phase BFT with linearity," *Cryptol. ePrint Arch.*, 2022. [Online]. Available: https://eprint.iacr.org/2022/551

[22] A. Spiegelman, N. Giridharan, A. Sonnino, and L. Kokoris-Kogias, "Bullshark: DAG BFT protocols made practical," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2022, pp. 2705–2718.

[23] G. Danezis, L. Kokoris-Kogias, A. Sonnino, and A. Spiegelman, "Narwhal and tusk: A dag-based mempool and efficient BFT consensus," in *Proc. 17th Eur. Conf. Comput. Syst.*, 2022, pp. 34–50.

[24] M. Conti, E. S. Kumar, C. Lal, and S. Ruj, "A survey on security and privacy issues of bitcoin," *IEEE Commun. Surv. Tut.*, vol. 20, no. 4, pp. 3416–3452, Oct.-Dec. 2018.

[25] Y. Sompolinsky and A. Zohar, "Secure high-rate transaction processing in bitcoin," in *Proc. Int. Conf. Financial Cryptogr. Data Secur.*, 2015, pp. 507–527.

[26] L. Baird, "The swirlds hashgraph consensus algorithm: Fair, fast, Byzantine fault tolerance," Swirlds, Inc.Technical Report SWIRLDS-TR-2016, 2016.

[27] X. Boyen, C. Carr, and T. Haines, "Graphchain: A blockchain-free scalable decentralised ledger," in *Proc. 2nd ACM Workshop Blockchains Cryptocurrencies Contracts*, 2018, pp. 21–33.

[28] Y. Sompolinsky and A. Zohar, "Phantom," *IACR Cryptol. ePrint Arch.*, Report 2018/104, 2018. [Online]. Available: https://eprint.iacr.org/2018/104

[29] A. S. Tanenbaum, and M. Van Steen, *Distributed Systems: Principles and Paradigms*. Englewood Cliffs, NJ, USA: Prentice-Hall, 2007.

[30] L. Lamport, R. Shostak, and M. Pease, "The Byzantine generals problem," in *Proc. Concurrency: Works Leslie Lamport*, 2019, pp. 203–226.

[31] C. Decker, J. Seidel, and R. Wattenhofer, "Bitcoin meets strong consistency," in *Proc. 17th Int. Conf. Distrib. Comput. Netw.*, 2016, pp. 1–10.

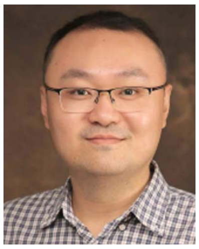
[32] V. Buterin and V. Griffith, "Casper the friendly finality gadget," 2017, *arXiv:1710.09437*.

[33] Y. Xiao, N. Zhang, W. Lou, and Y. T. Hou, "A survey of distributed consensus protocols for blockchain networks," *IEEE Commun. Surv. Tut.*, vol. 22, no. 2, pp. 1432–1465, Feb. 2020.

[34] A. Bessani, J. Sousa, and E. E. Alchieri, "State machine replication for the masses with BFT-smart," in *Proc. IEEE/IFIP 44th Annu. Int. Conf. Dependable Syst. Netw.*, 2014, pp. 355–362.

Zijian Zhang (Member, IEEE) is an associate professor in the School of Cyberspace Science and Technology with Beijing Institute of Technology. He was also a research fellow with the School of Computer Science with the University of Auckland, and was a visiting scholar in the Computer Science and Engineering Department of the State University of New York, Buffalo. His research interests include authentication and key exchange, user identification and behavior inference, consensus mechanism.

Xuyang Liu (Student Member, IEEE) received the BE degree in computer science and technology from the Beijing Institute of Technology, in 2022. He is currently working toward the PhD degree in the School of Cyberspace Science and Technology with the Beijing Institute of Technology. His research interests include applied cryptography, blockchain technology, distributed consensus.



Meng Li (Senior Member, IEEE) is an associate researcher and the dean assistant with the School of Computer Science and Information Engineering, Hefei University of Technology (HFUT), China. His research interests include security, privacy, fairness, vehicular networks, applied cryptography, privacy computing, cloud computing, edge computing, blockchain.

Hao Yin received the BE and MS degrees in information security and software engineering from the University of Science and Technology Beijing, in 2015 and 2018, respectively, and the PhD degree in computer science and technology from the Beijing Institute of Technology. His research interests include applied cryptography, information security, privacy preservation, blockchain technology, distributed consensus.

Liehuang Zhu (Senior Member, IEEE) is a professor in the School of Computer Science and Technology with the Beijing Institute of Technology. He is selected into the Program for New Century Excellent Talents in University from Ministry of Education, P.R. China. His research interests include cryptographic algorithms and secure protocols, Internet of Things security, cloud computing security, Big Data privacy, mobile and Internet security, and trusted computing.

Bakh Khoussainov (Member, IEEE) received the PhD degree in mathematics from the Algebra and Logic Department, Novosibirsk University, USSR. He is currently a professor with the Computer Science Department, The University of Auckland, New Zealand. His research interests include computable algebraic systems and model theory, automata and automatic structures, games on finite graphs and complexity, abstract data types and algebraic specifications, computably enumerable reals and randomness. He is also an editor of the Journal for Symbolic Logic.

Keke Gai (Senior Member, IEEE) received the PhD degree in computer science from Pace University, New York, NY, USA. He is currently a professor with the School of Cyberspace Science and Technology, Beijing Institute of Technology, Beijing, China. His research interests include cyber security, blockchain, and privacy computation. He has published more than 150 peer-reviewed papers in recent years and has been granted 10 Best Paper Awards. He is serving as an editor-in-chief of the journal Blockchains and an Editorial Board member of a few journals, including TDSC, JPDC, FGCS, etc. He is currently serving a few academic organizations, e.g., a Co-Chair of IEEE Technology and Engineering Management Society (TEMS)'s Technical Committee on Blockchain and Distributed Ledger Technologies.