Swarm: A High-Performance Asynchronous BFT Protocol Adapted to High Network Delay

Jianyu Han, Xiulong Liu*, Keqiu Li
College of Intelligence and Computing, Tianjin University, China
Tianjin Key Laboratory of Advanced Networking (TANK)
Email: {hanjianyu, xiulong_liu, keqiu}@tju.edu.cn

***Abstract*—Recently, asynchronous byzantine fault tolerance (BFT) consensus has made progress in linearized communication complexity, allowing nodes to broadcast a small number of transactions at their own pace in networks with variable bandwidth. However, this means that network propagation delay poses a more severe bottleneck to the system compared to transmission delay affected by block size and bandwidth. Particularly in networks with high propagation delays, nodes need to wait more for messages to propagate through the network, which severely hinders system performance. We propose Swarm, an asynchronous BFT protocol that is suitable for networks with tight bandwidth and propagation delay. The new protocol structure supports highly concurrent propagation-delay-intensive and propagation-delay-tolerant stages by separating the control and data layers and ensures that nodes achieve an ordered and consistent log for parallel blocks. We also design a group broadcasting strategy that allows nodes to not broadcast proposals from other nodes without compromising safety and liveness while reducing communication cost. Importantly, our work is orthogonal to existing improvements based on communication complexity and bandwidth and does not sacrifice existing results. We built a complete system prototype and conducted a thorough evaluation. The results show that Swarm performs better in networks with higher propagation delays, achieving 2-5x TPS and reducing latency by 50% compared to existing asynchronous protocols for different scales.**

***Index Terms*—Blockchain, Byzantine fault tolerance, Asynchronous consensus**

I. Introduction

A. Background

Blockchain is a decentralized distributed ledger technology that has been widely applied in various fields. Its core is to achieve consistency and trustworthiness among distributed nodes through consensus mechanisms. However, since blockchain is essentially a distributed Byzantine system, it cannot avoid a range of problems, such as unreliable network communication, node failure, and malicious behavior. The FLP theorem [1], which is well-known in the distributed field, theoretically proves that consistent deterministic consensus is impossible in a pure asynchronous environment. Therefore, most practical Byzantine fault tolerance (BFT) protocols compromise on this issue. The (partial) synchronous protocols [2]–[5], which strengthen assumptions about network conditions, have achieved good performance. Unfortunately, due to bandwidth fluctuations, unreliable links, significant delays, and even network attacks, these synchronous assumptions may not always apply to wide-area networks (WANs). When the network is adversarial, depending on synchronous assumptions may cause fatal vulnerabilities. Asynchronous consensus protocols use random sources to avoid this ”impossibility” and do not make any assumptions about the network, thereby exhibiting robustness against adversarial networks. However, since the 1980s, many attempts have mainly focused on theoretical feasibility and performed poorly [6]–[10], so they have not been widely adopted. It was not until recent years that emerging practical solutions, such as HoneyBadgerBFT [11], DispersedLedger [12], and DumboBFT [13] and its latest variant improvements [14], [15], have provided new paradigms for practical asynchronous BFT, bringing asynchronous consensus protocols into the practical phase.

B. Limitations of Existing Studies

HoneyBadger BFT is the first practical asynchronous BFT consensus protocol in WANs. It is based on the classic asynchronous common subset (ACS) protocol [16] proposed by Ben-Or et al. to achieve consensus. An ACS instance consists of N parallel RBCs and N parallel ABAs, where N is the number of nodes. RBC [17] simulates reliable broadcast channels through point-to-point (P2P) communication, allowing a designated sender to propagate a batch of transactions. Due to the asynchrony, it cannot be guaranteed that all honest nodes complete the RBC within a certain time. The ABA selects N - f (where f is the number of byzantine nodes) RBC results that have been jointly completed as the output after consensus.

Subsequent research has improved the cryptography and protocol instances to optimize performance, making it more suitable for different scenarios. Some works propose using MVBA [18] to replace inefficient ABA and reduce the number of ABA instances. Another work proposes the re-proposable RABA, which allows for fully parallelized ABA instances, greatly improving the performance of asynchronous BFT protocols. Researchers also take into account the unevenness of network bandwidth and divide the model into a bandwidth-intensive broadcast phase and a bandwidth-independent agreement phase, supporting concurrent execution of transaction broadcasting and consensus to improve performance.

Overall, current asynchronous BFT protocols mainly focus on studying the communication complexity of different paradigms and the limitations of network bandwidth on system performance. Although they have achieved certain results in specific scenarios, they face two main shortcomings:

- **Insufficient delay awareness**: Existing asynchronous BFT consensus protocols have paid little attention to dealing with propagation delays in wide-area networks. As nodes are dispersed in different regions worldwide, long-distance communication, network congestion, and multiple routing hops can result in a low-quality network conditions, which may lead to high propagation delays and affect the performance of the entire system.
- **Reliable broadcast efficiency issues**: Traditional RBC protocols have high broadcast costs, which limit the performance of the entire system. Some new protocols (such as sDumbo) try to optimize broadcast costs by reducing the strong effectiveness of RBC, but the cost is an increase in the number of protocol rounds, which exacerbates the cost of delay awareness.

C. *Our Contribution*

In this paper, we propose the following solutions to overcome the above shortcomings:

- **Broadcast strategy optimization**: To reduce the cost of the broadcast and tolerate larger blocks without sacrificing overall delay, we found that nodes need to broadcast erasure-coded blocks from other nodes to the entire network. However, any node only needs $f + 1$ blocks to reconstruct the complete proposal. This leads to the "waste" of sending N - $2f$ erasure-coded blocks even if there are f byzantine nodes. Our challenge is to ensure that each node receives at least $f + 1$ erasure-coded blocks from honest nodes to reconstruct the complete proposal without broadcasting to the entire network. To address this, we design a broadcast grouping strategy that borrows the idea of a circular queue to ensure that each node is located in the sending group of at least $f + 1$ honest nodes, thus ensuring the security of the broadcast.
- **Delay-awareness optimization**: To overcome the system bottleneck caused by propagation delay, we abstract the ACS protocol into a control layer and a data layer. Each stage in the data layer is divided into delay-intensive and delay-tolerant stages and perform fine-grained decoupling. Each stage is viewed as an independent working unit "Bee" and records the information of each epoch in its buffer pool. In this way, idle resources can be more fully utilized during data transmission to achieve "compression" of propagation delay by parallel processing of messages from multiple epochs. However, designing the aforementioned steps is not easy. To prevent byzantine nodes from frequently attacking honest nodes' buffer pools, we design a credential mechanism at the control layer so that only allows proposals that meet the QC to be collected in the buffer pools. Due to the existence of random sources, the end time of each epoch cannot be accurately predicted. Therefore, we design a chain update mechanism based on state iteration at the control layer to ensure the consistency of log submissions from different nodes. Through this mechanism, we can handle time differences between different nodes, ensuring the correctness and reliability of consensus.

In summary, our contributions are as follows:

- We propose the GRBC protocol, a design paradigm that can be applied to any broadcast protocol using erasure coding. We optimize the communication redundancy caused by the broadcast phase and innovatively design a circular grouping strategy that reduces the communication complexity by $N/(2f + 1)$ while ensuring security.
- We propose the Swarm Collaboration Mechanism (SCM), which optimizes the ACS protocol by fine-grained decoupling to minimize the impact of propagation delay on the performance. We design a credential mechanism and a chain update mechanism to ensure the safety and consistency of the protocol, and significantly improve the TPS without affecting latency. To the best of our knowledge, we are the first team to consider the impact of propagation delay on asynchronous consensus performance and propose a solution. The solution adopts the design pattern of traditional asynchronous consensus and can be applied to other asynchronous consensus as a feasible paradigm for decoupling the ACS.
- We implement the above system using Go and compare it with a classic ACS protocol. Experimental results show that applying the GRBC protocol can achieve more significant improvements in networks with more obvious transmission delays while applying SCM can achieve excellent performance in networks with higher propagation delay. The final experimental results show that our method can achieve 2-5x improvement.

The next sections will cover the following topics: Section II will summarize related work, Section III will introduce the network model and system design, Section IV will report our experimental results, and finally, Section V will summarize our work and discuss future directions.

II. Related Works

A. *SMR Issues and ACS Protocols*

The main problem addressed in this paper is the issue of Byzantine fault-tolerant state machine replication (SMR) [20] in asynchronous networks. Typically, SMR assumes a server-client model where N servers maintain N replicas of a state machine, with at most F servers being Byzantine, i.e., their behavior can be arbitrary. The BFT protocol must ensure that the state machine is replicated on all correct servers, even in the presence of Byzantine servers. Typically, this is achieved by delivering a consistent, totally ordered transaction log to all servers. In asynchronous consensus, SMR is mainly achieved through the use of ACS. The ACS works as follows: in a single epoch, all nodes propose their proposals, and ACS selects a subset of at least N - f proposals as the common output.

B. Paradigms and Practical Improvements

The first practical ACS was implemented in HoneyBadger, where all N nodes broadcast proposals each epoch, and when a subset of proposals to be committed by the ABA satisfies certain conditions to reach a consensus. This establishes the BKR (Ben-Or, Kelmer, and Rabin) paradigm for constructing ACS using RBC+ABA. Another study, Dumbo, innovatively uses MVBA to implement a faster asynchronous BFT protocol, establishing the CKPS (Cachin, Kusawe, Petzold, and Shoup) paradigm for constructing ACS using RBC+MVBA.

Subsequent research has directly improved the practical implementation of various cryptographic and consensus protocols, thanks to the excellent work of BEAT [21] and Aleph [22]. In particular, BEAT carefully examines different use cases (such as threshold encryption [24], erasure coding algorithm [25], random coin [26], verifiable information dispersal (VID) [27], etc.), and provides recommendations for selecting appropriate deployment components in practice, optimizing performance to make it more flexible to adapt to different scenarios. Some other work has noted that the inefficiency of the ABA severely restricts the performance of the system. Dumbo reduces the number of ABA instances intuitively by electing a committee and using MVBA instances in an innovative way. Another work, PACE [19], proposes a re-proposable protocol RABA that allows fully parallelized ABA, greatly improving the performance.

There are also some research works, such as DispersedLedger and Dumbo-NG [15], that address the unevenness of node bandwidth. They divide the above model into a bandwidth-intensive broadcast phase and a bandwidth-independent agreement phase, making full use of bandwidth, greatly improving performance. Specifically, DispersedLedger allows nodes with lower bandwidth to participate in a proposal and voting in the broadcast phase without downloading the complete proposal, while the agreement phase retrieves and downloads the complete proposal in parallel according to its own bandwidth. Similarly, Dumbo-NG also based on bandwidth inconsistency, decouples the broadcast phase and agreement phase into two independent parts, supporting concurrent execution of proposal broadcast and asynchronous agreement.

Unfortunately, the current asynchronous BFT protocols do not effectively study the propagation delay, which also affects the performance. Although ResilientDB [23] considers the impact of geographical location on propagation delay in synchronous networks based on PBFT [2], which integrates communication by electing regional leaders and is not suitable for asynchronous environments. This actually stimulates our interest in exploring this direction. As we mentioned briefly before, our approach is orthogonal and compatible with the above research: we reduce the communication cost by narrowing the broadcast range, which is effective for any practical broadcast model. In addition, our focus on the decoupled SCM is to solve the limitations brought by propagation delay, which does not change the essence of existing protocols. It is based on the round design policy to ensure security and consistency, and improve parallelism. Therefore, for experiments, we focus on comparing with HB, and combining the above improvements with our research will be interesting future work.

III. System Design

A. Network Model

Before introduction, we will outline the network model:

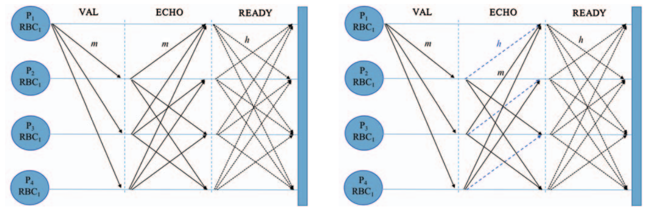
- The model consists of N nodes and f byzantine nodes, where $N \geq 3f + 1$. Honest nodes strictly follow the protocol and faithfully execute the tasks. Byzantine nodes can deceive, they can behave like honest nodes, but may also execute arbitrary destructive behaviors, and can coordinate with other byzantine nodes to carry out malicious strategies. Similar to other BFT consensus, the number of nodes and the number of byzantine nodes are predetermined and remain constant throughout the system. As it is an asynchronous network, we do not make any assumptions about network conditions, and message delivery may be indefinitely delayed.
- Any two nodes communicate through a P2P channel. The transmission delay refers to the time required for a node to send a message, which is determined by the bandwidth. The propagation delay refers to the time it takes for the message to propagate in the network to the target node. The latency of a transaction refers to the time it takes for a contained proposal to be proposed, transmitted through multiple rounds of P2P communication, and finally completes consensus and written to the log.

B. GRBC Protocol

As we all know, P2P communication goes through two stages: the transmission stage which is related to the bandwidth, and the propagation stage which is associated with the network conditions. To improve the throughput, we can let transmission compete with propagation by carrying more data, for example, increasing the block size to accommodate 10,000 or even 100,000 transactions. This will improve the throughput but at the cost of sacrificing the latency. We aim to reduce the broadcasting cost without sacrificing the latency. It can be said that as long as the communication cost is reduced, we can send blocks containing more transactions within the same transmission time. To better understand our work, let's first review the workflow of the RBC protocol as shown in Fig.1a:

- Node P_i proposes some transactions (B/N txs) for agreement based on a random selection rule and generates N data blocks using an erasure coding algorithm as VAL messages, which are broadcast to the N nodes in the network. This process is called the VAL stage.
- Other nodes receive the VAL message from P_i and broadcast ECHO messages containing the data blocks. When they receive ECHO messages from other nodes and verify them, they increment their counter. After receiving $2f + 1$ valid ECHO messages, they broadcast a READY message containing only the proposal hash. This process is called the ECHO stage.

- When a node receives a READY message, it increments its counter. After receiving $2f + 1$ READY messages, it uses $f + 1$ ECHO messages to reconstruct the complete proposal and input it into the corresponding ABA. This process is called the READY stage.



(a) RBC (b) GRBC

Fig. 1: The workflow of reliable broadcast protocol.

In the ECHO stage, each node broadcasts the data block. When $2f + 1$ ECHO messages are collected, the READY stage begins. The purpose of collecting $2f + 1$ ECHO messages is to confirm that at least $f + 1$ honest nodes have received the data block from Node P_i, forming a "definitely recoverable" certificate to ensure that any honest node can collect enough ECHO messages to reconstruct the complete proposal. In the READY stage, only $f + 1$ ECHO messages are needed to reconstruct the complete proposal. This means that f ECHO messages carrying data blocks are redundant, and these ECHO messages can carry only the proposal hash as a certificate to avoid wasting the cost of f data blocks. Therefore, can we design a scheme that allows nodes to receive confirmation from at least $f + 1$ honest nodes and collect at least $f + 1$ data blocks to reconstruct the complete proposal without broadcasting throughout the network?

We designed a grouping strategy to improve the RBC protocol, which we call GRBC (as shown in Fig.1b). Our design idea is to assume that each node does not broadcast, but instead sets a sending group (a subset of all nodes), and it only broadcasts the data block to the nodes in this group. This ensures that all nodes in the group can receive the data block that the node broadcasts. For other nodes outside the group, they only need to send the proposal hash as confirmation. Since nodes need to collect at least $f + 1$ data blocks to reconstruct the complete proposal, it is necessary to ensure that any node is in a sending group of $2f + 1$ nodes to ensure that even if f byzantine nodes behave maliciously, they can still collect $f + 1$ data blocks sent by honest nodes. Thus, we borrow the idea of a circular queue (as shown in Fig.2). We arrange the nodes in a circular order based on ID and design each node's sending group to include itself and the next $2f + 1$ nodes. This ensures that for any node, the sending group of previous $2f + 1$ nodes will contain it. Therefore, even if f byzantine nodes behave maliciously, it is guaranteed to receive $f + 1$ data blocks. With this design, GRBC's communication cost is reduced from the original $O(N^2m)$ to $O(Nfm)$, which can reduce the transmission latency and improve performance. On the other hand, without sacrificing the latency, the saved bandwidth can be used to send more transactions, resulting in a TPS improvement of $N/(2f + 1)$.

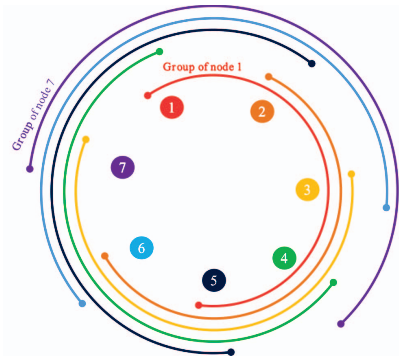


Fig. 2: The circular grouping strategy diagram.

C. Swarm Collaboration Mechanism

Simply increasing the block size and competing with propagation delay not only sacrifices latency but more importantly, cannot have an effect on the propagation delay-intensive stages. To understand this problem, let's take the RBC protocol as an example and carefully consider each stage. The communication complexity of the VAL stage is $O(Nm)$, and it of the ECHO stage is $O(N^2m)$ (which can be reduced to $O(Nfm)$ through GRBC). These two stages can increase the block size to reduce the impact of propagation delay. We call this kind of stage a propagation delay-tolerant stage. However, the communication complexity of the READY stage is $O(\lambda)$. Even if the block size changes, the short messages sent in this stage remains the same. Similarly, the short messages sent in each stage of the ABA protocol are also independent of the block size. We call this kind of stage a propagation delay-intensive stage, and propagation delay-intensive stages will cause bandwidth idle time for each epoch.

We aim to support the concurrent execution of delay-intensive and delay-tolerant stages to reduce the impact of propagation delay and approach optimal performance. To achieve this, we design a new structure that separates the data and control layers(as shown in Fig.3) and decouple each stage into independent units called "Bee" in data layer, focusing only on inputs and outputs. Each stage has its own (unbounded) buffer to record the execution status of each epoch. This way, we can achieve high parallelism and send messages of different stages of different epochs in a single data transmission. Especially in cases where propagation delay is significant, such parallelism can greatly improve throughput.

However, designing the above model is not trivial, and we face the following challenges:

- **Proposal storm**: Controlling the timing of the VAL stage is a key issue because the broadcast in the VAL stage directly affects the parallelism and the load on the instance buffer. If byzantine nodes continuously broadcast their proposals to other nodes, the buffer and downstream

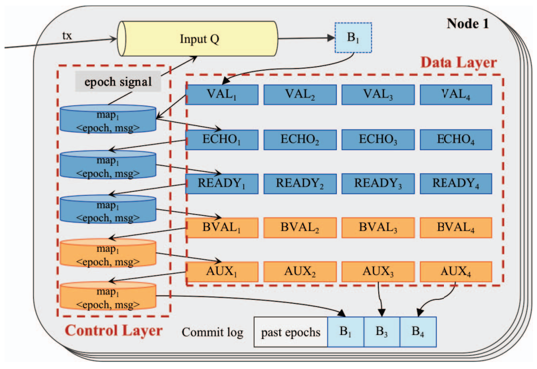

Node 1
tx
Input Q
B_1
Data Layer
epoch signal
VAL_1 VAL_2 VAL_3 VAL_4
$ECHO_1$ $ECHO_2$ $ECHO_3$ $ECHO_4$
$READY_1$ $READY_2$ $READY_3$ $READY_4$
$BVAL_1$ $BVAL_2$ $BVAL_3$ $BVAL_4$
AUX_1 AUX_2 AUX_3 AUX_4
map_i <epoch, msg>
Control Layer
Commit log
past epochs
B_1 B_3 B_4

Fig. 3: Swarm Collaboration Mechanism diagram.

bandwidth of other nodes will be consistently occupied, causing a decrease in performance or even paralysis.

- **Consistency issue**: In asynchronous protocols, due to the random source, the rounds and end times of each epoch cannot be accurately predicted. This may cause the consensus of a later epoch completes before an earlier epoch during the agreement stage, and the consensus result is added to the log. Furthermore, the order in which each epoch is completed on different nodes may be inconsistent, leading to confusion between node.

To address the above challenges at the control layer, we further propose the following solutions:

- To address the issue of proposal storm, we draw on the idea of collecting QC in the ECHO and READY stages. Specifically, when a node initiates a new epoch proposal, it needs to provide credentials for the previous epoch's proposal. When a node collects proposal certs from $2f+1$ nodes for the previous epoch, it indicates that at least $f+1$ honest nodes have sent proposals. At this point, the node can initiate the proposal for the next epoch. This credential mechanism can prevent byzantine nodes from launching attacks by continuously sending proposals. Therefore, we redesigned the broadcast protocol (refer to Fig.4), where a node needs to carry the certs of $2f+1$ nodes it received for the previous epoch's proposal and sign the proposal for the current epoch as a credential for other nodes to initiate the next epoch's proposal.

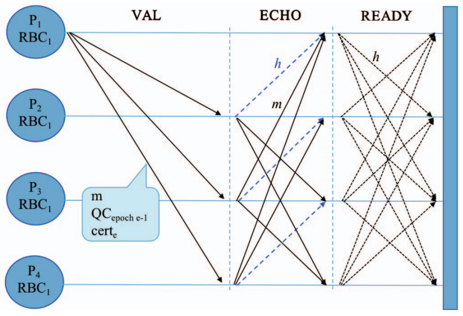


Fig. 4: VAL message content diagram.

We need to explain the setting of the security threshold of $2f+1$. If the threshold is set to $f+1$, f byzantine nodes can send proposals to an honest node to meet its QC threshold. In this case, the proposal broadcast by the honest node will reach the QC threshold of the byzantine nodes, and they will initiate the next epoch of proposals, repeating the process and allowing the byzantine nodes to collude to attack a single honest node. Therefore, we need to set the threshold to $2f+1$ to prevent such attacks from byzantine nodes. That is, for each epoch proposal, at least $2f+1$ credentials from nodes are required, where at least $f+1$ of them are honest nodes. This setting ensures the security and reliability of the system.

- To address the consistency issue, we propose a chain update mechanism based on state iteration (as shown in Fig.5). We require each node to strictly determine the order of log writes according to the epoch. For each epoch block, there are three states: Agreeing, Decide, and Finish, with the initial state being Agreeing. When ACS confirms an epoch block, we do not immediately write it to the log, but update its status to Decide. If the previous epoch block is in the state of Finish, we will update the state of the epoch and subsequent blocks to Finish, and write them to the log in order. This mechanism ensures that each epoch block is written to the log in the correct order, thereby maintaining consistency between nodes.

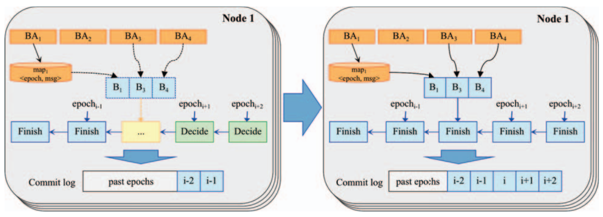


Fig. 5: The chain update mechanism diagram.

IV. Evaluation

A. Implementation

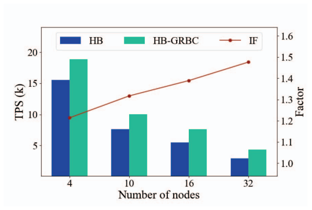
We implemented a prototype of a HoneyBadger system using 4,000 lines of Go language code and further implemented our Swarm system on top of it. We refer to the system using the GRBC protocol on top of the HoneyBadger as HB-GRBC and the system using the SCM on top of the HoneyBadger as HB-SCM. The entire system is macroscopically modeled into three modules: one module continuously generates a batch of transactions and sends them into the nodes' transaction pool, another module is responsible for message transmission between nodes, and the most important module is responsible for running a corresponding protocol instance (HB, HB-GRBC, HB-SCM, Swarm). In addition, we used Reed-Solomon to encode and decode blocks of erasure coding algorithm and use merkle tree to organize data blocks.

B. Experimental Setup

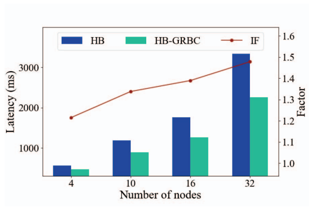
We deployed the system in a real LAN environment with a server cluster, where the machines in the cluster had stable communication delay and the same bandwidth. Multiple nodes were deployed on the servers, forming a fully connected graph with a network link between each pair of nodes. Each server used a controlled deployment environment and underwent a series of simulation tests. Specifically, each node was equipped with 8GB of running memory and 4 CPU cores, with a total upstream and downstream bandwidth of 100Mbps. We observed the effects on TPS and latency by setting block size, number of nodes, and propagation delay between each pair of nodes (simulating typical delays between nodes in different geographical locations and network conditions). This controlled setup allowed us to precisely define changes in network conditions and achieve fair, repeatable evaluations and comparisons. To measure TPS, our transaction generation module created a high-load system with a guaranteed backlog, and the consensus module reported the ratio of confirmed transactions on each node for each epoch. Moreover, to prevent competition from message broadcasting caused by parallelization, we set higher priorities for messages of lower epochs to fine-tune the protocol and ensure that concurrency does not compromise system stability. Additionally, each transaction was specified as 128 bytes in size with a 32-byte hash.

C. Evaluation Results of GRBC Protocol

We observed the improvements in TPS and latency when replacing RBC with GRBC. Since GRBC aims to improve performance from the perspective of optimizing communication costs, it is mainly affected by the number of nodes and block size. Therefore, we set the system security threshold to $1/3$ and propagation delay $D = 20ms$ and evaluated the TPS, latency, and improvement factor (IF) in scenarios with different numbers of nodes and fixed block size $B = 10,000$ (as shown in Fig.6), and in scenarios with different block sizes and fixed number of nodes $N = 32$ (as shown in Fig.7).



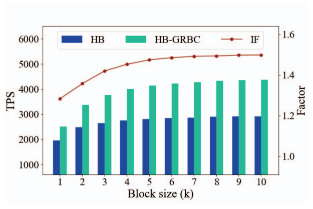
(a) TPS



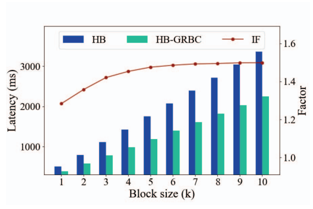
(b) Latency

Fig. 6: The performance of HB and HB-GRBC, with different number of nodes.

From the results, we can see that as the number of nodes and block size increase, HB-GRBC shows increasingly better performance in terms of TPS and latency, gradually approaching a 1.5x improvement. This is because as the number of nodes and block size increase, there is a higher communication cost, and the transmission delay relative to propagation delay



(a) TPS

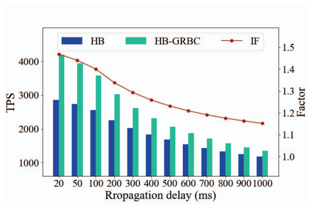


(b) Latency

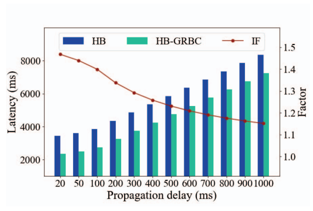
Fig. 7: The performance of HB and HB-GRBC, with different block sizes.

increases in the overall delay, which leads to more significant performance improvement with the GRBC protocol. We also notice that when the block size is increased by 1.5x, such as $3,000$ and $2,000$, $6,000$ and $4,000$, $9,000$ and $6,000$, the latency of both HB and HB-GRBC is approximately equal, but the throughput improvement is approximately 1.5x, which is consistent with the theoretical value of $N/(2f + 1)$ for performance improvement.

To further verify the above conclusion (the higher transmission cost, the more obvious the improvement of HB-GRBC), we also set the security threshold to $1/3$ and evaluated the impact of different propagation delays on TPS, latency, and IF with a fixed number of nodes $N = 4$ and a fixed block size $B = 1,000$. The results are shown in Fig.8.



(a) TPS

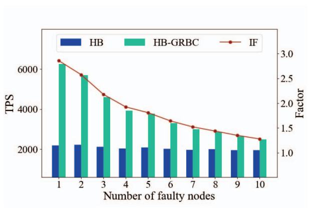


(b) Latency

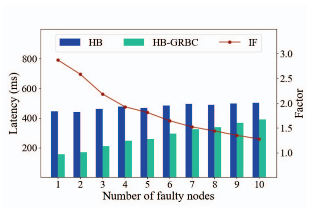
Fig. 8: The performance of HB and HB-GRBC, with different propagation delay.

We found that as the propagation delay increases, the TPS and latency of HB-GRBC and HB gradually converge, which is consistent with our previous conclusion. As the propagation delay increases, the weight of the transmission delay in the overall delay decreases, and the improvement of the GRBC protocol based on communication costs gradually becomes less significant, leading to the gradual convergence of the performance of HB-GRBC and HB.

Finally, we return to the discussion on the security threshold. Calculations show that, without affecting the overall delay, the theoretical IF of performance is $N/(2f + 1)$, which means that the performance improvement is also affected by the number of byzantine nodes f. Therefore, we evaluated the impact of f on system throughput, latency, and IF with a fixed propagation delay $D = 20ms$, the number of nodes $N = 32$, and block size $B = 10,000$, as shown in Fig.9.

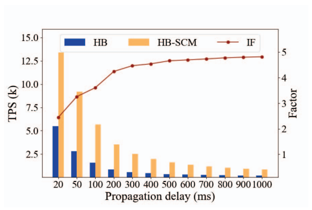


(a) TPS

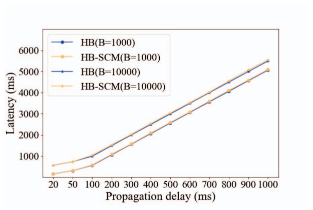


(b) Latency

Fig. 9: The performance of HB and HB-GRBC, with different number of byzantine nodes.



(a) TPS



(b) Latency

Fig. 10: The performance of HB and HB-SCM, with different propagation delay.

We found that as the number of byzantine nodes decreases, the factor of performance improvement will be greater. In the case of only one byzantine node, the performance improvement reaches an astonishing 3x, and even under the security threshold of $1/3$ that does not compromise system security, the performance improvement is close to 1.5x.

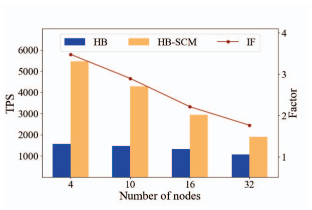
In summary, we can conclude that the GRBC reduces the transmission delay by lowering the communication cost, and its improvement in performance becomes more significant in networks with larger numbers of nodes and blocks. With a security threshold of $1/3$, the performance improvement gradually approaches the theoretical value of 1.5. Additionally, the GRBC protocol saves sending bandwidth and can pack more transactions without sacrificing latency, resulting in an approximate 1.5x increase in TPS. What's more, the performance improvement of GRBC fluctuates with the proportion of byzantine nodes. In networks of the same scale, the performance improvement gradually increases as the number of byzantine nodes decreases. Therefore, HB-GRBC is suitable for networks with a large number of nodes, low propagation delay, and a small security threshold, and the block size can be flexibly chosen based on the tolerance for latency.

D. Evaluation Results of SCM

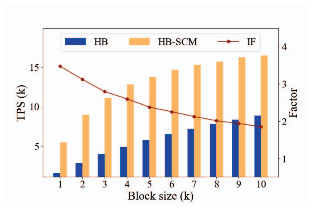
We observed the changes in TPS and latency with the use of SCM. The principle of SCM is to achieve parallelization by decoupling, reducing the restrictions of propagation delay on performance, and thus improving TPS. Therefore, we also set the security threshold to $1/3$ and evaluated the impact of different propagation delays on TPS, latency, and IF with the fixed number of nodes $N = 4$ and block size $B = 1,000$, as shown in Fig.10.

We found that as the propagation delay increases, the improvement in TPS of HB-SCM becomes more significant, with an increase of 2-5x, while the latency remains unchanged. This is because as the propagation delay increases, the proportion of propagation delay to transmission delay in the overall delay increases, and the optimization of SCM becomes more significant, resulting in an improvement in TPS. However, SCM does not change the rounds for block consensus completion, so it does not affect the latency.

Similarly, the performance of HB-SCM is also affected by communication costs. To further verify this conclusion, we set the propagation delay $D = 100ms$ and evaluated the TPS and IF with different numbers of nodes for the block size $B = 10,000$ (as shown in Fig.11a), as well as the TPS and IF with different block sizes for the number of nodes $N = 32$ (as shown in Fig.11b).



(a) TPS with different N.



(b) TPS with different B.

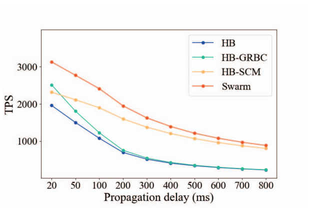
Fig. 11: The TPS of HB and HB-SCM.

We found that the TPS improvement of HB-SCM gradually decreases as the number of nodes and block size increases. HB's TPS gradually approaches that of HB-SCM, but HB-SCM always performs better than HB. This is because as the number of nodes and block size increase, the communication cost becomes higher, and the effect of propagation delay on performance becomes less significant, resulting in a smaller improvement in TPS by SCM.

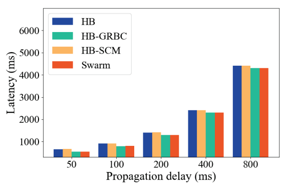
In summary, we can conclude that SCM greatly improves TPS by reducing the impact of propagation delay. This means that in environments with poor network conditions, long distances between nodes, or high propagation delay caused by other factors, as well as in networks with small numbers of nodes and small block sizes where transmission delay is not significant, SCM performs very well.

Finally, since GRBC can reduce communication costs to alleviate transmission delay, and SCM can reduce the impact of propagation delay, we integrated the GRBC protocol and SCM into our Swarm, expecting to observe a more significant overall performance improvement. We set the security threshold to $1/3$, took $N = 32$ as an example, and evaluated the TPS and latency under different propagation delays for a block size $B = 1,000$ (as shown in Fig.12), as well as the TPS and latency under different block sizes for a propagation delay $D = 100ms$ (as shown in Fig.13).

We found that when the number of nodes and block size are large and communication costs are high, the performance

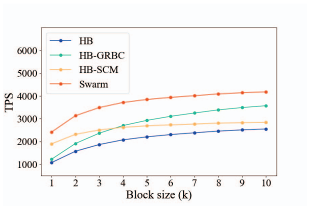


(a) TPS

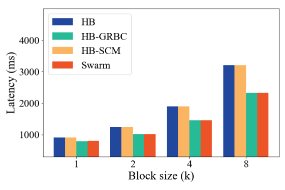


(b) Latency

Fig. 12: The performance of HB, HB-GRBC, HB-SCM and Swarm, with different propagation delay.



(a) TPS



(b) Latency

Fig. 13: The performance of HB, HB-GRBC, HB-SCM and Swarm, with different block sizes.

of HB-GRBC is better than HB-SCM, and as the propagation delay increases, the performance of HB-SCM gradually surpasses HB-GRBC. Similarly, when the propagation delay is large, the performance of HB-SCM is better than HB-GRBC, and as the communication costs increase, the performance of HB-GRBC gradually surpasses HB-SCM. However, the Swarm integrated with GRBC and SCM performs very well in any case.

V. Conclusion and Future Work

We proposed Swarm, an efficient asynchronous BFT consensus protocol that performs well in networks with tight communication costs and propagation delays. Specifically, we proposed the GRBC protocol, which allows nodes to ensure system security and liveness without forwarding proposal contents to the entire network by designing grouping strategies, thus reducing communication costs. We also proposed the Swarm Collaboration Mechanism that supports parallelization while considering both security and consistency, achieving a significant increase in throughput in networks with high propagation delays. Although we considered the impact of propagation delay on actual asynchronous BFT performance, there are still some interesting issues. In asynchronous networks, propagation delay is affected by geographic location and network conditions, and there is unevenness in propagation delay between different nodes, which has a great impact on nodes collecting sufficient QC. Therefore, our main focus in future work is to study how to overcome the unevenness of network delay, which is also a meaningful problem.

References

[1] Michael J Fischer, Nancy A Lynch, and Michael S Paterson. 1985. Impossibility of distributed consensus with one faulty process. Journal of the ACM (JACM) 32, 2 (1985), 374–382.

[2] Miguel Castro, Barbara Liskov, et al. 1999. Practical byzantine fault tolerance. In Proc. OSDI 1999. 173–186.

[3] Pierre-Louis Aublin, Sonia Ben Mokhtar, and Vivien Quéma. 2013. Rbft: Redundant byzantine fault tolerance. In Proc. ICDCS 2013. 297–306.

[4] Gilad Y, Hemo R, Micali S, et al. Algorand: Scaling byzantine agreements for cryptocurrencies[C]//Proceedings of the 26th symposium on operating systems principles. 2017: 51-68.

[5] M. Yin, D. Malkhi, M. K. Reiter, G. Golan-Gueta, and I. Abraham. HotStuff: BFT consensus with linearity and responsiveness. In Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, pages 347–356. ACM, 2019.

[6] Ittai Abraham, Danny Dolev, and Joseph Y Halpern. 2008. An almost-surely terminating polynomial protocol for asynchronous byzantine agreement with optimal resilience. In Proc. PODC 2008. 405–414.

[7] Michael Ben-Or and Ran El-Yaniv. 2003. Resilient-optimal interactive consistency in constant time. Distributed Computing 16, 4 (2003), 249–262.

[8] Christian Cachin, Klaus Kursawe, and Victor Shoup. 2005. Random oracles in Constantinople: Practical asynchronous Byzantine agreement using cryptography. Journal of Cryptology 18, 3 (2005), 219–246.

[9] Achour Mostefaoui, Hamouma Moumen, and Michel Raynal. 2014. Signature-free asynchronous Byzantine consensus with t¡ n/3 and O (n2) messages. In Proc. PODC 2014. 2–9.

[10] Arpita Patra, Ashish Choudhary, and Chandrasekharan Pandu Rangan. 2009. Simple and efficient asynchronous byzantine agreement with optimal resilience. In Proc. PODC 2009. 92–101.

[11] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. 2016. The honey badger of BFT protocols. In Proc. CCS 2016. 31–42.

[12] Lei Yang, Seo Jin Park, Mohammad Alizadeh, Sreeram Kannan, and David Tse. 2022. DispersedLedger: High-Throughput Byzantine Consensus on Variable Bandwidth Networks. In Proc. NSDI 2022.

[13] Bingyong Guo, Zhenliang Lu, Qiang Tang, Jing Xu, and Zhenfeng Zhang. 2020. Dumbo: Faster asynchronous bft protocols. In Proc. CCS 2020. 803–818.

[14] Bingyong Guo, Yuan Lu, Zhenliang Lu, Qiang Tang, Jing Xu, and Zhenfeng Zhang. 2022. Speeding Dumbo: Pushing Asynchronous BFT Closer to Practice. In Proc. NDSS 2022.

[15] Gao Y, Lu Y, Lu Z, et al. Dumbo-ng: Fast asynchronous bft consensus with throughput-oblivious latency[C]//Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security. 2022: 1187-1201.

[16] Michael Ben-Or, Boaz Kelmer, and Tal Rabin. 1994. Asynchronous secure computations with optimal resilience. In Proc. PODC 1994. 183–192.

[17] Gabriel Bracha. 1987. Asynchronous Byzantine agreement protocols. Information and Computation 75, 2 (1987), 130–143.

[18] Cachin, C., Kursawe, K., Petzold, F., Shoup, V.: Secure and efficient asynchronous broadcast protocols. In: Annual International Cryptology Conference. pp. 524–541. Springer (2001)

[19] Zhang H, Duan S. Pace: Fully parallelizable bft from reproposable byzantine agreement[C]//Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security. 2022: 3151-3164.

[20] L. Lamport, R. Shostak, and M. Pease. The Byzantine generals problem. In Concurrency: The Works of Leslie Lamport, pages 203–226. ACM, 2019.

[21] S. Duan, M. K. Reiter, and H. Zhang. BEAT: Asynchronous BFT made practical. In Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, pages 2028–2041. ACM, 2018.

[22] Gagol A, Swietek M. Aleph: A leaderless, asynchronous, byzantine fault tolerant consensus protocol[J]. arXiv preprint arXiv:1810.05256, 2018.

[23] Gupta S, Rahnama S, Hellings J, et al. Resilientdb: Global scale resilient blockchain fabric[J]. arXiv preprint arXiv:2002.00160, 2020.

[24] J. Baek and Y. Zheng. Simple and efficient threshold cryptosystem from the gap Diffie-Hellman group. GLOBECOM '03, pp. 1491–1495, 2003.

[25] J. Plank and L. Xu. Optimizing Cauchy Reed-Solomon codes for fault-tolerant network storage applications. NCA 2006.

[26] A. Boldyreva. Efficient threshold signature, multisignature and blind signature schemes based on the gap-Diffie-Hellman-group signature scheme. PKC 2003.

[27] C. Cachin and S. Tessaro. Asynchronous verifiable information dispersal. In 24th IEEE Symposium on Reliable Distributed Systems, pages 191–201. IEEE, 2005.