FaBFT: Flexible Asynchronous BFT Protocol Using DAG

Yu Song[1], Yu Long[1], Xian Xu[2⋆], and Dawu Gu[1⋆]

[1]Shanghai Jiao Tong University
{sy_121,longyu,dwgu}@sjtu.edu.cn
[2]East China University of Science and Technology
xuxian@ecust.edu.cn

Abstract. The Byzantine Fault Tolerance (BFT) protocol is a long-standing topic. Recently, a lot of efforts have been made in the research of asynchronous BFT. However, the existing solutions cannot adapt well to the flexible network environment, and suffer from problems such as high communication complexity or long latency. To improve the efficiency of BFT consensus in flexible networks, we propose FaBFT. FaBFT's clients can make their own assumptions about the network conditions, and make the most of their networks based on different network assumptions. We also use the BlockDAG structure and an efficient consistent broadcast protocol to improve the concurrency and reduce the number of steps in FaBFT. The comparison with other asynchronous BFT protocols shows that FaBFT has lower complexity and cancels the dependency on the view change. We prove that FaBFT is an atomic broadcast protocol in the flexible networks.

Keywords: Byzantine Fault Tolerance Protocol, Asynchronous Network, Flexible Consensus, DAG

1 Introduction

With the increasing popularity of blockchain and distributed applications, the need for atomic broadcasting protocols that can meet real-world scenarios becomes significantly more urgent. To this end, Byzantine fault tolerance (BFT) and distributed consensus have been studied for more than 40 years. BFT consensus enables a group of parties, who do not trust each other, to reach an agreement in permissioned environments. Compared to the permissionless consensus, the BFT consensus merits high efficiency and good performance, so it suits better for applications in mission-critical infrastructures.

Most of the conventional BFT protocols [4,18] utilize some time-bound assumption, either synchronous or partially synchronous, to guarantee the achievement of the security agreement. Specifically, [14] proposed the concept of *flexible consensus*, which allows clients to make different assumptions about the network condition. In 2021, a flexible BFT protocol, named Ebb-and-Flow protocol [16], was proposed in the partially synchronous network. Roughly, Ebb-and-Flow utilized two types of ledgers. As one ledger is the prefix of the other, these two ledgers can grow independently with different network assumptions (i.e., either synchronous or partially synchronous), without violating the requirements of the secure BFT. Thus, Ebb-and-Flow can flexibly meet the needs of different clients.

Unfortunately, none of these partially synchronous consensus schemes can maintain their security without the time-bound assumption, and all these solutions lack liveness when the network experiences long delays or fluctuations. For parties that are not within close distance and do not share stable communications with each other, these protocols cannot ensure agreement. This leads to barriers against consensus in asynchronous environments such as WAN.

In 2016, the first practical asynchronous Byzantine fault tolerance (aBFT), named HoneyBadger BFT [15], was proposed and proven secure without any time-bound assumptions. HoneyBadger does not suffer from network fluctuation and has stronger robustness and responsiveness. Subsequently, a series of works have been proposed [5,11,17,19]. As far as we are concerned, all the existing aBFT solutions utilize a two-phase mechanism, i.e., the *broadcast phase*, followed by the *agreement phase*. In the broadcast phase, the reliable broadcast (RBC) is used typically to guarantee that valid blocks can be delivered by honest parties. In the agreement phase, to bypass the FLP impossibility theorem [7] (i.e., deterministic consistency cannot be achieved in asynchronous networks), a random procedure is required, such as the common coin method [2,3]. Both phases require multiple rounds of communications and complex protocol flows, which make the aBFT protocol less usable. Thus the aBFT has been viewed as a “theoretical” consensus [10], and the early studies on aBFT protocols are not yet satisfactory from the standpoint of realistic applications.

One most critical issue in aBFT lies in the “unbalance” in the two phases, including both the bandwidth and the time requirements. Roughly, the reliable broadcast phase requires $O(n^3)$ times of communications (n is the number of the BFT parties), which leads to high communication overhead. Meanwhile, the agreement phase needs much less bandwidth but a much longer time. Consequently, the different requirements in the two phases result in a big waste of bandwidth. Most recently, some new methods have been proposed to address these issues, basically by applying different consensus strategies under different network conditions. Some of the aBFT schemes, such as [9,13,17], can deal with the “optimistic case”. When these protocols detect that the real network is stable or performs better than the “pessimistic case”, they can change the consensus strategy to make full use of the network and improve the performance of the consensus. Specifically, BDT [13] utilizes a complex *transformer* in its pace-sync process, which can achieve the switch between the optimistic fast lane and the pessimistic path. Bullshark [17] needs vertexes (blocks) in different networks that have different voting types and uses two types of leaders, i.e., the steady-state leaders and the fallback leaders, to commit blocks. The mechanism of dealing with network switching is highly complex and time-consuming. As such, Bullshark achieves weak liveness only [20]. Ditto [9], which derives from its partially synchronous version named Jolteon [9], uses the asynchronous fallback technique to handle the asynchronous network and view change phase. However, unlike the aforementioned flexible consensus, all these optimistic aBFT solutions maintain only one ledger instead of two, and thus the clients can not make their choice based on their own network assumptions. In other words, all these optimistic aBFT schemes are more suitable to work under the single network assumption.

Concerning the performance, some existing (a)BFT solutions adopt the *single chain* structure to form the ledger. For example, in the HoneyBadger BFT protocol [15] or Dumbo-families protocols [11,10,13], only the block which contains at least $n - f$ transaction batches from different parties can be committed in a round. Instead of assembling transactions in blocks to form a single chain, using the *BlockDAG* structure to organize the *DAG chain* can preserve the parties' concurrent blocks, and fully utilize the bandwidth.

In this work, we extend the idea of flexible consensus to the asynchronous BFT and adapt the BlockDAG structure to this setting. Inspired by Ebb-and-Flow [16], our basic idea is to ask the committee parties to maintain two ledgers, including the *safer ledger* (for conservative clients) and the *faster ledger* (for aggressive clients). The clients can make their own network assumptions: either the " optimistic case" or the "pessimistic case". For the conservative clients who think the network condition is the pessimistic case, i.e., the asynchronous case, the safer ledger can be utilized to guarantee a secure consensus. For aggressive clients who believe that their network conditions are as good as the partially synchronous case, the faster ledger can be used to speed up the committing of transactions. In this way, we effectively cancel the time-bound assumption when dealing with the "pessimistic case", and optimize the utilization of the network in the "optimistic case"".

To realize this high-level idea, however, there are some technical challenges. In particular, in order to guarantee a secure consensus, how can we utilize the BlockDAG structure and reduce the unbalance between phases for both ledgers? Moreover, how can we make sure that the safer ledger is the prefix of the faster ledger?

Our Contributions. To achieve secure BFT consensus in the flexible asynchronous network and answer the foregoing questions, we propose FaBFT, an asynchronous BFT protocol using DAG to support flexible consensus. To the best of our knowledge, FaBFT is the first flexible BFT consensus protocol that achieves all of the following properties.

- **Flexible in an asynchronous network.** FaBFT can run in both partially synchronous and asynchronous networks, and the network assumption is totally made by the clients. Thus the clients can make flexible choices independently, without sacrificing security.
- **DAG-based.** FaBFT's committee parties utilize the BlockDAG structure to improve the transaction throughput and network utility.
- **More efficient protocol.** We use the more efficient consistent broadcast (CBC) to design the broadcast phase, which reduces the rounds and thus the time latency. In the partially synchronous / asynchronous case, only 6 / 12 steps are required to commit blocks, respectively, completely eliminating the view change.

Table 1. Comparison of related BFT protocols

Protocol	Network Assumption [1]	Steps[2]	Time Complexity [3]	Communication Complexity[4]	Message Complexity[5]	Structure	DAG Based	View Change
Hotstuff [18]	P-Sync.	6	$O(1)$	$O(n)$	$O(n)$	3-Phase Voting	N	Y
Ebb-and-Flow [16]	Sync.+P-Sync.	N.A.	$O(1)$	$O(n^2\|m\|)$	$O(n^2)$	Longest-Chain+BFT	N	Y
HBBFT [15]	Asy.	Best: 9 Adv.: 13.5	$O(\log n)$	$O(n^2\|m\| + \lambda n^3 \log n)$	$O(n^3)$	RBC+ABA	N	N
Speeding-Dumbo [10]	Asy.	Best: 10 Adv.: 16	$O(1)$	$O(n^2\|m\| + \lambda n^3 \log n)$	$O(n^2)$	RBC+MVBA	N	Y
BDT [13]	Opt-Asy.	Best: 3 Adv.: 63	$O(1)$	$O(n^2\|m\| + \lambda n^3 \log n)$	$O(n^3)$	CAST/RBC+ BA+ MVBA	N	N
BullShark [17]	Opt-Asy.	Best: E(10) [6] Adv.: E(20)	$O(1)$	$O(n^3\|m\| \log n)$	$O(n^3)$	RBC	Y	N
Ditto [9]	Opt-Asy.	Best: 5 Adv.: E(10.5)	$O(1)$	$O(n^2\|m\|)$	$O(n^2)$	Asy-fallback	N	Y
This work	P-Sync.+Asy.	Best: 6 Adv.: 12	$O(1)$	$O(n^2\|m\|)$	$O(n^2)$	CBC+GPC	Y	N

[1] We use “Sync.” for “Synchronous”, “P-Sync.” for “Partially-Synchronous”, “Asy.” for “Asynchronous”, and “O-Asy. ” for “Optimistic Asynchronous”. Ebb-and-Flow is flexible in the partially synchronous network, and our FaBFT is flexible in the asynchronous network.
[2] We use “Steps” to estimate the approximate time from the block generation to the block commitment. Assume that the message from the sender reaches the receiver as one step. “Best” means that all parties are honest and the network is stable. “Adv.” means the asynchronous network with 1/3 Byzantine parties. E(s) means that the expected value of steps is s.
[3] Time Complexity: The expected number of rounds of communication before the protocol terminates. n is the number of the BFT parties.
[4] Communication Complexity: The expected value of the length of the message generated by the honest node for committing a block. m represents the average bit length of the block.
[5] Message Complexity: The expected value of the total messages that an honest nodes has generated.
[6] The “wave blocks” will be committed together after two or four rounds.

Table 1 compares FaBFT with other related consensus protocols.

2 Preliminaries

2.1 BFT Consensus

BFT consensus is a kind of state machine replication protocol that can tolerate Byzantine nodes. Specifically, the BFT consensus divides the committee party nodes into two categories. One is the dishonest Byzantine parties that will deviate from the protocol in some way, such as not sending messages or sending garbage messages. The other is the honest parties, which strictly comply with the protocol. Traditionally, BFT protocols work in the permissioned setting.

2.2 BlockDAG Chain and Basic Operations

Instead of the single (i.e., longest) chain rule used in Bitcoin, BlockDAG consensus organizes blockchain into a *Direct Acyclic Graph* (DAG) which can benefit from all blocks created in parallel by honest parties. Through the DAG structure, BlockDAG consensus makes good use of the node's bandwidth and achieves true concurrency. Since there are multiple paths from the genesis block to one block, more operations need to be introduced to define BlockDAG. We use $\mathsf{parent}(\text{Chain}, \text{B})$ to denote the set of blocks which the block B references directly, $\mathsf{past}(\text{Chain}, \text{B})$ to denote the set of all blocks that B references (directly or indirectly), and $\mathsf{tips}(\text{Chain})$ to denote the set of blocks with 0 in-degree, i.e., end blocks. Also, we call past(Chain, B) the precursor blocks of B, which are blocks traversed on all paths from the genesis block to B. We provide a BlockDAG instance in Figure 1.

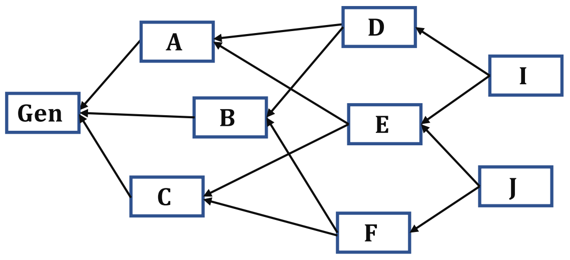

Fig. 1. An example of a DAG chain. For block J, parent(DAG, J)={E, F}, past(DAG, J) = {Gen, A, B, C, E, F}, and tips(DAG) = {I, J}.

2.3 Other Building Blocks

Here we introduce the *consistent broadcast* and the *global perfect coin* protocols as our building blocks to construct ledgers.

Definition 1 (Consistent Broadcast, CBC). *The consistent broadcast protocol consists of the c-broadcast and the c-deliver phases [1,6]. The sender first executes c-broadcast with request m and thereby starts the protocol, and all parties terminate the protocol by executing c-deliver with request m. Consistent broadcast ensures only that the delivered requests are the same for all receivers. In particular, when the sender is faulty, it does not guarantee that every party terminates and delivers a request. A protocol for consistent broadcast satisfies:*

- *(Validity) If an honest party P_s c-broadcasts m, then all honest parties eventually c-deliver m.*
- *(Consistency) If an honest party c-delivers m and another honest party c-delivers m', then $m = m'$.*
- *(Integrity) Every honest party c-delivers at most one request from the same sender. Moreover, if the sender P_s is honest, then the request was previously c-broadcast by P_s.*

We also use the *Global Perfect Coin* (GPC) protocol as a blackbox in the construction of the safer ledger for asynchronous networks. The global perfect coin protocol, also called the common coin protocol [15], typically works as a source of randomness for asynchronous BFTs. Many previous works [2,3,11,12,17] use the $(f+1, n)$-threshold signature to instantiate the global perfect coin to solve the Asynchronous Byzantine Agreement (ABA) problem. Specifically, when a party is invoked in a global perfect coin instance ω, this party signs ω with his private key and sends the resulting signature share to all other parties. Once a party receives at least $f+1$ shares, this party can re-construct the full signature σ on ω via the threshold signature's combination algorithm, and then the hash value of ω is viewed as the randomness to choose the "leader" among all parties.

To describe the required properties of the global perfect coin, we say that each committee party P_i uses the $\mathsf{ChooseLeader}_{P_i}(w)$ process to get the leader of the instance ω, and X_ω stands for the random variable that some committee party is chosen as the leader. A secure global perfect coin protocol satisfies the following requirements.

Definition 2 (Global Perfect Coin, GPC). *A global perfect coins protocol running by an n-member committee has the agreement, termination, unpredictability, and fairness properties.*

- *(Termination) If at least $f+1$ parties call the* ChooseLeader *processes on an instance ω respectively, then each of these process can eventually return a coin-choosing result. The choosing result is also called as the leader.*
- *(Agreement) If two honest parties, P_i and P_j, call* ChooseLeader *on instance ω and get the returned values X_ω and X'_ω respectively, then $X_\omega = X'_\omega$.*
- *(Unpredictability) As long as no less than $f+1$ parties call the* ChooseLeader *on instance ω, then the returned value is computationally indistinguishable from the random choosing. Concretely, the probability that anyone can predicate the leader choosing result correctly is $1/n$.*
- *(Fairness) The coin is fairly chosen. Each party shares an equal probability to be chosen as the leader. That is, for P_i and P_j, $\Pr[X_\omega = P_i] = \Pr[X_\omega = P_j]$.*

3 Model and Security Definition

In this section, we provide the security model and definition for FaBFT.

3.1 The Security Model

FaBFT runs in the permissioned setting with an initially fixed number of committee parties. Each authorized committee party uses a public key as the unique pseudonym.

Adversary model. Without loss of generality, we consider that the committee set consists of n parties indexed by $i \in [n]$, where $[n] = \{1, ..., n\}$ and use f to denote the maximum number of the Byzantine faulty parties. Without loss of generality, we assume that $n = 3f+1$. That is, no more than $1/3$ committee parties can be corrupted by the adversary $\mathcal{A}$. For the corrupted parties, $\mathcal{A}$ learns all their internal states and takes full control of them, and thus there is no communication cost within Byzantine faulty parties. In FaBFT, we consider the static adversary only.

Network model. Network communication is point-to-point. In this network, adversary $\mathcal{A}$ is capable of delaying or reordering any messages between all the parties, but it cannot drop or modify the messages broadcast by honest parties. As generally accepted [13,17,16], we define the partially synchronous and asynchronous networks as below.

Assume that every party has a local clock controlled by $\mathcal{A}$ and the time of a "tick" is at the speed of the actual network δ. Also assume that there is a global stabilization time (GST), after which all messages can be delivered within time δ. If $GST > 0$ the network is called to be *partially synchronous*, and if $GST = \infty$ the network is called to be *asynchronous*.

3.2 Definition of Security

Since FaBFT focuses on solving the asynchronous Byzantine Atomic Broadcast (ABC) problem in the flexible network to attain robustness and reduce latency, we utilize the definition of ABC to provide the security definition of FaBFT [15].

Definition 3 (Atomic Broadcast). *In an ABC protocol, there are at most f Byzantine nodes among the n parties. Each of the parties has a transaction buffer and randomly selects transactions from its buffer to package a block as the inputs to this protocol. The ABC protocol will output an ordered transaction ledger. ABC protocol satisfies the following properties with all but negligible probability:*

– *(Total-order) If one honest party has outputted a sequence of transactions $<tx_0, tx_1, ..., tx_j>$ and another one has outputted $<tx'_0, tx'_1, ..., tx'_{j'}>$, then $tx_i = tx'_i$ for $i \leq min(j, j')$.*
– *(Agreement) If any honest party outputs a transaction tx, then every honest party outputs tx.*
– *(Liveness or Censorship Resilience) If a transaction tx has been inputted to $N - f$ honest parties, then it is eventually outputted by every honest party.*

4 Technical Overview

Current asynchronous consensus cannot meet the requirements of flexible networks. Specifically, in order to guarantee the "pessimistic-case" security, clients are unable to fully utilize their network in reality, especially when their real network conditions are better than the pessimistic-case scenario, which can lead to wasted bandwidth and long-term waits. In contrast, FaBFT introduces the flexible consensus in asynchronous networks, providing more resilient options for different clients. In addition, FaBFT uses BlockDAG to organize transactions to improve throughput.

4.1 Flexible Architecture

In FaBFT, each party may be in one of two different network environments. That is, the client can make different assumptions about the state of the network, i.e., partially synchronous or asynchronous. To reach a secure consensus in this environment, BFT parties run our FaBFT protocol to maintain two types of ledgers,

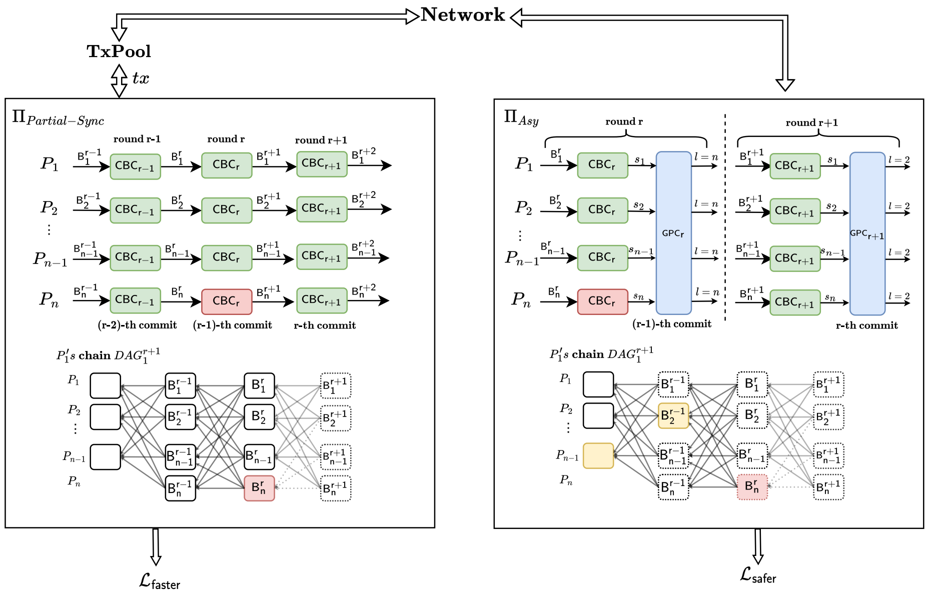

Fig. 2. The FaBFT architecture. FaBFT runs two parallel protocols. 1) The left box denotes the running of the partially synchronous protocol $\Pi_{\text{Partial-Sync}}$. The upper is $\Pi_{\text{Partial-Sync}}$'s running process in round $r+1$. Each green box denotes one CBC instance running smoothly, while the red box denotes one CBC instance with some errors. $P_1's$ chain DAG_1^{r+1} is shown at the bottom. Each solid box denotes a committed block and each dashed box denotes one block which is not arriving or in P_1's block buffer. Since $P_n's$ CBC in round r fails to arrive in P_1, blocks generated in round $r+1$ are added to P_1's block buffer, and then the dashed arrows help P_1 to retrieve B_n^r at round $r+1$. 2) The right box denotes the running of the asynchronous protocol Π_{Asy}. The blue boxes denote the GPC processes. In DAG_1^{r+1}, each yellow box denotes a leader blocks. Since B_n^r reaches lately and is elected as the leader in round $r+1$, blocks from round $r-1$ to $r+1$ are not committed. When B_n^r arriving, B_n^r and $\text{past}(\text{DAG}_1^r, \text{B}_n^r)$ can be committed.

faster ledger for partially synchronous networks and *safe ledger* for asynchronous networks. In particular, a safer ledger is a prefix to a faster ledger. The client chooses one of the two ledgers based on the client's network assumptions.

FaBFT guarantees that both ledgers can meet the security requirements of the BFT protocol regardless of the real network. In other words, clients can take advantage of a flexible network environment for better performance. In addition, by adopting the BlockDAG structure, the throughput of our consensus protocol depends only on the network speed.

To illustrate the FaBFT protocol at a high level, we assume that there are n parties, where up to t could be broken by a static adversary $\mathcal{A}$.

Flexible protocol. The FaBFT committee runs two parallel protocols as Figure 2 shows. The left ledger (i.e., the faster ledger $\mathcal{L}_{\text{faster}}$) has aggressive partial synchronous network assumption and runs faster. In this setting, every party's clock is consistent with the network speed and almost unanimously. The protocol is propelled by “step” and every step consists of time “ticks”. The right one (i.e., the safer ledger $\mathcal{L}_{\mathsf{safer}}$) has a conservative asynchronous network assumption and runs slower but safer. Protocol operates at the network flow rate and advances in sufficient “quantities”. Both $\mathcal{L}_{\mathsf{faster}}$ and $\mathcal{L}_{\mathsf{safer}}$ are constantly increased in each round. A two-phase process is required to commit blocks, including the *broadcast phase* and the *agreement phase*. It is worth noting that the two-phase process under the two ledgers is not the same, and the flexibility of FaBFT relies on the second phase, where the client chooses a committing strategy based on his own network assumption.

(1) *The broadcast phase.* During the broadcast phase, every honest party generates one block and broadcasts it. It collects votes from other parties for their blocks to form a broadcast certificate set and also votes for other blocks. Only the block with a legal certificate may be recorded on DAG.

(2) *The agreement phase.* In the agreement phase, an honest party will / will not generate a new block for the $\mathcal{L}_{\mathsf{faster}}$ / $\mathcal{L}_{\mathsf{safer}}$. Roughly,

– The aggressive clients assuming that they are working in the partial synchronization network use the timeout mechanism similarly to their broadcast phase. That is, the honest parties create new blocks to guarantee that enough blocks can be generated in the current round, to ensure that all blocks in the previous round can be committed safely.
– Conservative clients, who assume themselves in an asynchronous network environment, use a counting mechanism instead of creating new blocks. They start a random process only when enough blocks have been collected, to decide the committed blocks.

BlockDAG structure. Like [8,17], FaBFT adopts the BlockDAG structure. Specifically, each legal block references at least $n-f$ “parent blocks” generated in the previous round. This way, once a new block is recorded on one party’s DAG chain, this party has received not only the certificate of this block but also all the parent blocks referenced by this block. Therefore, the recording of this new block implies that the party has agreed on all the referenced blocks. If this new block is committed, all the referenced ones are committed too. We note that in FaBFT, there are $2f+1$ (i.e., instead of only 1 as in Figure 1) initial *genesis blocks* in the first round, and these genesis blocks appear in the past set of any valid blocks in FaBFT. If there are only the genesis blocks on the FaBFT ledgers, then parties only reference them. Otherwise, the parties will reference all (i.e., no less than $n-f$) tips of the DAG in their view.

Transaction packaging and block constructing. When receiving transactions from clients, one party puts them into its own buffer pool TransactionPool and packages the new transactions TX into the blocks. TX’s Merkle tree root, denoted as MKroot, is recorded in the head of the block. The resulting blocks, which contain the references to all tips (i.e., end blocks in DAG), are submitted by the parties to get recorded on the ledger. FaBFT runs in rounds, and each honest committee generates exactly one block in one round. A block B_i^r submitted by party P_i at the beginning of the $r-$th round contains the identity information of

P_i, the round number r, the transactions TX contained in this block, and the references to parent blocks in the previous round PaB_i^{r-1}. We emphasize that only blocks with certificates can be recorded on the ledger. Specifically, a block's qualified certificate consists of at least $n-f$ *vote*s for this block. Each vote of B_i^r is generated by one party's *voting* in the r-th round. For simplicity, we denote a block authorized B_i^r generated by P_i as $B_i^r = \{\text{MKroot}, P_i\text{'s identity}, r, \mathsf{TX}, \mathsf{PaB}_i^{r-1}\}$, and denote the qualified certificate as QC. We also call a block B_i^r with QC as a "legal block".

Block checking and voting. When a party receives blocks, it adds them to its local buffer, checks them, and votes for valid ones. Especially, if one party finds a received block containing references to some parent blocks which this party has not received yet, this party starts the "call for help" process for other parties and retrieves the missing blocks if only they really existed. If all checks pass, this party waits for blocks' QCs, and removes a block from the local buffer to the end of the DAG when the corresponding QC arrives. It worth noting that in the partially synchronous network, valid missing blocks will be sent in this step with the QC.

4.2 Two-phase Consensus in Partially Synchronous Networks

In the partially synchronous network, we can assume that there is a uniform clock. In this environment, the protocol runs round-by-round, and the protocol's rounds are driven by a timer. Each round consists of 3 steps, and the timer starts at the beginning of each step. A block B_i^r created by P_i at the beginning of the r-th round (i.e., the broadcast phase of B_i^r) will be committed at the end of the $(r+1)$-th round (i.e., the agreement phase of B_i^r), if B_i^r is correctly formed. Therefore, it takes 6 steps (or 2 rounds) to commit a block.

Broadcast phase. Generally, the broadcast phase consists of n parallel consistent broadcast (CBC) being run by every party. Each CBC instance consists of 3 steps: (1) Each honest party generates exactly one block at the beginning of the broadcast phase and broadcasts it among the committee. (2) Other parties check the received blocks and return the votes if all the checks pass. (3) Once at least $n-f$ votes for one block have been received, the block's QC has been formed and the creator of this block broadcasts QC. At the end of the broadcast phase, the honest parties add blocks to its DAG chain only if (i) each added block has a qualified certificate QC, and (ii) all the reference blocks appear in this party's DAG view. Otherwise, the blocks are saved in the party's block buffers. Under the partially synchronous assumption, at least $n-f$ new blocks will be added to $\mathcal{L}_{\mathsf{faster}}$ at the end of this phase.

We use an honest party P_1 running at round r as an example to illustrate the CBC phase in Figure 3. Here the block B_1^r together with its QC are received by other parties at the end of round r. It is worth noting that B_1^r can always be committed until the end of the $(r+1)$-th round.

The agreement phase. The agreement phase works in a way similar to the broadcast phase. At least $n-f$ new blocks will be added to $\mathcal{L}_{\mathsf{faster}}$ at the end of this phase, and each of the new block references to at least $n-f$ legal blocks (i.e., with their QCs) in the previous round. Therefore, at the end of this phase,

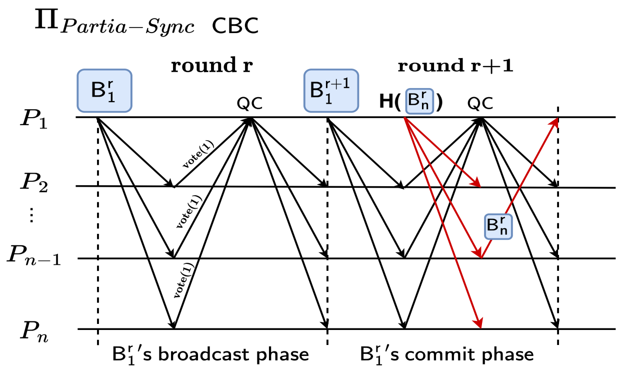

Fig. 3. P_1's CBC phase in $\Pi_{\mathsf{Partial-Sync}}$. 1) In round r, P_1 broadcasts a block B_1^r and collects unambiguous votes to form QC, then P_1 broadcasts QC to other parties. Round $r+1$ is B_1^{r+1}'s broadcast phase and also B_1^r's commit phase. When other parties receive B_1^{r+1}'s QC, B_1^{r+1} is added to their DAG chain, B_1^r can get committed. 2) The red lines denote that P_1 has received a block from some other party, but one of the parent blocks B_n^r is not contained in DAG_1^r, then P_1 calls help from others. At the end of round $r+1$, all blocks belonging to round r can be committed.

all blocks before this batch of blocks can be committed. In Figure 3, block B_1^r is committed when block B_1^{r+1} is recorded on $\mathcal{L}_{\mathsf{faster}}$.

4.3 Two-phase consensus in asynchronous network

Unlike the previous partially synchronous networks, asynchronous networks do not have the assumption of time bounds, and the advancement of each party depends entirely on the speed of their respective networks. Although the protocol still operates on a round-by-round basis, iterations of rounds are guaranteed by a counting mechanism rather than a timeout mechanism.

The broadcast phase. Each honest party advances into a new round via the CBC process by generating and broadcasting a new block. During each round, each honest party can generate exactly one block, and thus no less than $n-f$ blocks will be recorded on DAG. The most crucial difference lying in the CBC process under the asynchronous network is that there is no timer. Thus, parallel CBCs belonging to the same round can *not* start at the meantime. When a new block is generated, not only more than $n-f$ parent blocks but also all $\mathsf{tips}(\mathsf{DAG})$ blocks with no subsequent references should be referenced. Then broadcasts the block. After that, honest parties will keep waiting for more than $n-f$ consistent votings to form and broadcast the QC, instead of utilizing the timeout mechanism. When an honest party adds at least $n-f$ blocks on the DAG, this party will not only advance into the agreement phase but also the next CBC and organize a new block.

The agreement process. When a party admits $n-f$ blocks into its DAG during the CBC process, it will immediately boost the global perfect coin by broadcasting its random signature share, and $f+1$ random share will invoke GPC. Then a random leader P_l can be elected. If the leader's block B_l has been on DAG and fortunately it is a parent block for more than $n-f$ next-round blocks, B_l block and its history can be committed without ambiguity, which means more than $n-f$ parties has acknowledged B_l's QC and recorded B_l on DAG. Otherwise, there will be no progress in this round. We can prove that once the subsequent available leader's block appears, all these blocks can still be committed. The fact that this protocol can move forward satisfies the liveness and is censorship resistance.

5 Detailed Protocol Description

5.1 Block Generation.

At the start of a round, to generate a new block and add it to the BlockDAG, a party needs to refer to all end blocks (no less than $n-f$) in its BlockDAG chain (Here we emphasize that in the asynchronous network, the party needs to traverse its BlockDAG to reference all end blocks reached lately too). Moreover, the party needs to select transactions from its local transaction pool. The selected transactions constitute the body of a block, and the Merkle-tree root of these transactions is recorded in the head of the block. Then the party signs it to create a block. After that, the party broadcasts it instantly to others and then waits to collect enough votes to form a broadcast certificate QC for this block.

Algorithm 1: Generation of P_i's Block B_i^r

```
  Input: $\mathsf{tips}(\mathsf{DAG}_i^{r-1})$, $P_i$'s public key $\mathsf{P}_i$, $\mathsf{TransactionPool}$,
         $\mathsf{CurrentRound} = r$, $\mathsf{NetworkAssumption}$, $\mathsf{PaB}_i^{r-1} = \emptyset$
  Output: Block $B_i^r$
1 for $B \in \mathsf{tips}(\mathsf{DAG}_i^{r-1})$ do
2     add $\mathsf{HashPointer}(B)$ to $\mathsf{PaB}_i^{r-1}$
3 if $\mathsf{NetworkAssumption} = \mathit{Asynchronous}$ then
4     TRAVERSE($\mathsf{DAG}_i^{r-1}$) and add all end blocks to $\mathsf{PaB}_i^{r-1}$
5 $\mathsf{TX} \leftarrow$ SELECTRANDOMLY($\mathsf{TransactionPool}$)
6 $\mathsf{TransactionPool} \leftarrow \mathsf{TransactionPool} \setminus \mathsf{TX}$
7 $\mathsf{B}_i^r$.MKroot$\leftarrow \mathsf{MerkleRoot}(\mathsf{TX})$
8 $\mathsf{B}_i^r \leftarrow <\mathrm{MKroot}, \mathsf{P_i}, r, \mathsf{TX}, \mathsf{PaB}_i^{r-1}>$
9 BROADCAST($B_i^r$) and starts the timer
```

5.2 Block Transmission.

CBC guarantees that a new block can reach every party consistently. As one of the senders, P_i first broadcasts its newly generated block B_i^r to every other party and collects votes to form QC. More than $n-f$ unanimous votes form a broadcast QC for B_i^r. This QC proves that most of the parties have received B_i^r and acknowledged the correctness of it. Then P_i also sends the QC to all parties.

When P_j receives the new block B_i^r from P_i, it needs to check its validity, including the transaction correctness and the signature legality. If all checks pass, P_j returns a vote of this block to P_i. Meanwhile, P_j needs to check if all of the referenced blocks are in PaB_i^{r-1} in DAG_j. If not, P_j needs to call for help from others. In this case, other honest parties will send the corresponding parents' block(s) to P_j.

Specifically, the received blocks are first included in the receiver's block buffer. A block B_i^r in P_j's block buffer will be moved out and added to the P_j's DAG chain only if (1) B_i^r is correct, (2) B_i^r's QC has arrived, and (3) P_j has received all blocks stated in PaB_i^{r-1}.

Algorithm 2: P_i's Block Transmission

```
Input: $B_i^r$, CurrentRound $= r$, NetworkAssumption
Output: QC, $\mathsf{DAG}_i^r$
1  $\Sigma^r \leftarrow \emptyset$          ▷ $\Sigma^r$ denotes the votes' signature set recorded in QC
2  $P_i$ as a sender :
3  upon reveive vote ¡value, signature, info¿:
4  if CheckVoteValid(vote) $= TRUE$ and vote.$value = 1$ then
5      add vote.signature to $\Sigma_i^r$
6  if NetworkAssumption $= PartiallySynchronous$
     $\wedge$ timer $= TIMEOUT$ then
7      if $|\Sigma_i^r| \geq n - f$ then
8          QC $\leftarrow \{B_i^r, \Sigma_i^r\}$
9          broadcast(QC) and restart timer
10     else
11         MESSAGE $\leftarrow \{TIMEOUT, r\}$
12         send MESSAGE to the client
13 if NetworkAssumption $= Asynchronous$ then
14     wait until $|\Sigma_i^r| \geq n - f$
15         QC $\leftarrow \{B_i^r, \Sigma_i^r\}$
16     broadcast(QC)
17 $P_j$ as a receiver:
18 upon receive $B_i^r$:
19 if CheckValid($B_i^r$) $= TRUE$ then
20     send vote $< 1, \sigma_j, Hash(B_i^r) >$ to $P_i$
   // check the reference relationship of $B_i^r$
21 for $B \in B_i^r.\mathsf{PaB}_i^{r-1}$ do
22     if $B \notin \mathsf{DAG}_j^{r-1}$ then
23         add $B$ to $P_j$'s BlockBuffer and call for $B$ from others
24         break
25 upon receive QC:
26 wait until all $\mathsf{PaB}_i^{r-1} \in \mathsf{DAG}_j$, add $B_i^r$ to $\mathsf{DAG}_j^r$
```

5.3 Optimistic Case Block Confirmation

As Figure 3 shows, when the current CBC_r ends, there are at least $n-f$ blocks on the honest party P_i's DAG chain DAG_i^r, we denote these blocks as B^r. All blocks referenced by B^r on DAG_i^r have corresponding QC and have passed at least two voting processes. They can all be committed and recorded on the faster ledger.

Algorithm 3: Optimistic case Block Confirmation

```
Input: $\mathsf{DAG}^r$, $\mathsf{NetworkAssymption} = PartiallySynchronous$,
       $\mathsf{CurrentRound} = r$, parties $P_i (i \in [n])$
Output: $\mathcal{L}_{\mathsf{faster}}$
// The CBC process of round $r$ is running concurrently
1 for $i \in [n]$ do
2     while $P_i$ adds a block $B_j^r$ to $\mathsf{DAG}_i^r$ do
3         for $B \in B_j^r.\mathsf{PaB}_j^{r-1}$ do
4             COMMIT($B$)
5     if $\mathsf{timer} = TIMEOUT$ then
6         $\mathcal{L}_{\mathsf{faster}} \leftarrow$ INTERPRET($\mathsf{DAG}_i^r$)
7         restart the timer to advance into round $r+1$
8         break
9 return $\mathcal{L}_{\mathsf{faster}}$
```

5.4 Pessimistic Case Block Confirmation

If P_i received more than $n-f$ blocks in round $r+1$, it will broadcast its signature share and the agreement phase starts. $f+1$ shares from different parties in round r will invoke the random common coin GPC. A leader P_l will be elected. If there was leader's block B_l^r in round r on the view of DAG_i^{r+1} and B_l^r if referenced by more than $n-f$ blocks from $r+1$, B_l^r and all its casual history (i.e., blocks belong to $\mathsf{past}(\mathsf{DAG}_\mathsf{i}^{\mathsf{r+1}}, \mathsf{B}_\mathsf{l}^\mathsf{r})$) will be committed legally. Otherwise, if there is progress in round r, we can prove that the next available leader will also commit to these blocks.

6 Security Analysis

6.1 Security Theorem

In this section, we prove that FaBFT satisfies all the properties of the atomic broadcast (ABC), as defined in Definition 3, no matter what the network environment is. Besides, we discuss the censorship resistance of FaBFT.

Theorem 1. *Assuming $n = 3f + 1$, FaBFT can meet the total-order, agreement, and liveness properties defined in Definition 3, except for a probability exponentially small in n.*

Algorithm 4: P_i's Pessimistic Case Block Confirmation

```
Input: $\mathsf{DAG}_i^{r+1}$, $\mathsf{NetworkAssumption} = \mathit{Asynchronous}$, $\mathsf{CurrentRound} = r+1$
Output: $\mathcal{L}_{\mathsf{safer}}$
1  cnt $= 0$, $\mathbb{S} \leftarrow \emptyset$          ▷ $\mathbb{S}$ denotes the set of parties' signature share
2  if $|\{B \in \mathsf{DAG}_i^{r+1} \wedge B.round = r+1\}| \geq (n-f)$ then
3      send CurrentRound,signature share $s_i$ to GPC, $s_i$ is added to $\mathbb{S}$
4  if $|\mathbb{S}| \geq f+1$ then
5      $w \leftarrow$ SignSig($s_i$) for $s_i \in \mathbb{S}$
6      $l \leftarrow$ GPC(w)          ▷ $P_l$ is the selected leader
   // The number of blocks reference to $B_l^r$ in round $r+1$
7  for $B \in \mathsf{DAG}_i^{r+1} \wedge B.round = r+1 \wedge B_l^r \in \mathsf{PaB}_i^r$ do
8      $cnt \leftarrow cnt + 1$
9  if $cnt \geq n-f$ then
10     $\mathbb{B} \leftarrow \mathsf{parent}(\mathsf{DAG}_i^{r+1}, B_l^r)$
11     for $B \in \mathsf{past}(\mathsf{DAG}_i^{r+1}, B_l^r)$ do
12         if $B \notin \mathbb{B} \wedge B$ has not been committed then
13             add $B$ to $\mathbb{B}$          ▷ Commit the late blocks
14     Commit($\mathbb{B}$)
15     $\mathcal{L}_{\mathsf{safer}} \leftarrow$ Interpret($\mathsf{DAG}_i^{r+1}$)
16 else
17     advance into round $r+2$ with $\mathcal{L}_{\mathsf{safer}}$ not increases for round $r$
```

Proof. We proof Theorem 1 via the following lemmas. More concretely, FaBFT's total-order property follows from Lemma 1 and Lemma 2, the agreement property follows from Lemma 3, and the liveness is implied by Lemma 4.

Lemma 1 (Total-order of $\mathcal{L}_{\mathsf{faster}}$). *Suppose the network is partially synchronous. For any two honest parties P_i and P_j in round r, if P_i commits all new blocks generated in DAG_i^{r-1}, then P_j's $(r-1)$-th DAG view (i.e., DAG_j^{r-1}) will be the same as P_i's, within δ time after GST.*

Proof. Based on the integrity of CBC and the design of FaBFT, each honest sender P_s will generate one and only one block B_s^{r-1} in the $(r-1)$-th round and form the corresponding QC. Thus, all the other honest parties will get B_s^{r-1} and its QC at the end of the $(r-1)$-th round. That is, $B_s^{r-1} \in \mathsf{DAG}_i^{r-1}$ if P_i is honest. In the next round, B_s^{r-1} will be referenced by both honest parties P_i and P_j, and QC for B_i^r implies the existence of QC for B_s^{r-1}, and thus B_s^{r-1} is committed by P_i. Similarly, P_j will commit B_s^{r-1} before advancing into the $(r+1)$-th round.

The dishonest sender $P_{s'}$ can form a QC in the $(r-1)$-th round but only broadcast it to a subset of the parties. For instance, $P_{s'}$ sends QC to P_i but not P_j in round $r-1$. Thus, DAG_i^{r-1} and DAG_j^{r-1} are different, since the former contains $B_{s'}^{r-1}$ while the latter does not. In the next round, P_i will reference $B_{s'}^{r-1}$ to build the next block, while P_j will not. After P_i broadcasting B_i^r, P_j will find that it does not have all the referenced blocks of B_i^r, thus P_j cannot add B_i^r to DAG_j^r. Fortunately, based on the validity and consistency of CBC protocol, every honest party will agree on B_i^r which is generated by the honest P_i. In this way, P_j can agree on $B_{s'}^{r-1}$ and deliver it eventually.

In conclusion, under the partially synchronous setting, based on the reference relationship defined by the BlockDAG structure, two consecutive CBC executions guarantees that no matter what the sender's action is, every honest party will agree on the first CBC's outputs in the second running. Thus, honest parties will have the same DAG view in each round. In other words, in the partially synchronous network, honest parties always have the same DAG view in each round.

Lemma 2 (Total order of $\mathcal{L}_{\mathsf{safer}}$). *Suppose the network is asynchronous. If one honest party P_i commits the leader block B_l^r of round r, then through constant rounds, for another leader block $B_{l'}^{r'}$ of any round $r' > r$ committed by any other honest party P_j, there will be a path between B_l^r and $B_{l'}^{r'}$. In fact, every honest party's DAG^r will converge into the same in round r' within no more than constant rounds.*

Proof. Since each honest party can only generate one block in one round, blocks generated by the same honest party in different rounds form a path. After the CBC phase, all honest parties will c-deliver the same set of blocks through QC.

In round $r+1$, after the coin-tossing process, a unique leader P_l is selected for round r. For the good case (i.e., an honest leader is selected, with probability 2/3), the leader block is generated by the honest leader and with no less than $2f+1$ reference paths from round $r+1$. Since the honest parties will commit the same leader and its leader block B_l^r, all casual history blocks of B_l^r can be committed together. For the bad case (i.e., a dishonest leader is selected, with probability 1/3), since less than $2f+1$ reference "votes" can be collected in round $r+1$, this causes B_l^r to be unavailable as a leader block to commit blocks, and thus there is no progress in round r.

When a leader's block $B_{l'}^{r'}$ is committed in round r', if there is a path between these blocks B_l^r and $B_{l'}^{r'}$, then B_l^r and its casual history can still be committed. From round r to round r', suppose that there is no progress in at most D rounds ($D \leq r'-r$), then it happens with the possibility $\Pr[D] = (1/3)^D$, which decreases with D exponentially.

In conclusion, in the asynchronous setting, the honest parties will commit to the same leader in the same order. Based on the CBC and DAG reference relationship in FaBFT, before committing the leader block, all casual history has been already in the DAG. Once an honest party commits a leader B, it atomically interprets all of $B's$ causal history in a deterministic order, which is identical for all other honest parties.

Lemma 3 (Agreement of $\mathcal{L}_{\mathsf{faster}}$ and $\mathcal{L}_{\mathsf{safer}}$). *In the partially synchronous and the asynchronous networks, suppose that an honest party P_i commits a block B_s in round r. Then any other honest party will also commit B_s.*

Proof. From Lemma 1, in the partially synchronous network, honest parties always share the same DAG view round by round. Thus there will be no conflicting blocks. So the proof of total order implies the agreement.

In the asynchronous network, when P_i commits B_s in round r, from Lemma 2, we can find that B_s is in the casual history of the legal leader block B_l^r. Then P_i will commit B_l^r and B_s within a constant number of rounds. Thus honest parties can make the agreement on the DAG chain.

Lemma 4 (Liveness of $\mathcal{L}_{\mathsf{faster}}$ and $\mathcal{L}_{\mathsf{safer}}$). *Each legal block broadcast by an honest party P_i can eventually be added to the* DAG *chain of any honest party, which means that FaBFT guarantee liveness.*

Proof. The validity of CBC guarantees the liveness of FaBFT in the partially synchronous network. Concretely, two consecutive CBCs ensure that FaBFT can advance into a new round without getting stuck by the timeout mechanism.

In the asynchronous network, based on the termination property of the global perfect coin, FaBFT can continuously make progress with all but negligible probability, which decreases exponentially with D.

Theorem 2. *FaBFT is censorship resistance.*

Proof. Here we consider the case that the adversary utilizes the hostile network and holds an honest block for some time for censorship. A legal QC can help this block be recorded on the DAG by honest parties when it is released. Specifically one honest party who has received the QC but not the block can ask for others' help, and this will not increase the communication burden. The next legal leader will commit this block. Therefore, FaBFT is censorship resistance.

6.2 More Discussion

When the network assumption is inconsistent with the real environment, FaBFT can still guarantee security. In an optimistic network, clients can make any assumptions. Two consecutive CBCs are sufficient to guarantee the safety and liveness of honest blocks. As each honest party generates exactly one block in each round and Theorem 1, the slower $\mathcal{L}_{\mathsf{safer}}$ is the prefix of $\mathcal{L}_{\mathsf{faster}}$. In particular, when the clients assume that the network is partially synchronous but the real network is pessimistic, the faster ledger may stop increasing, and clients can adopt the safer ledger instead, i.e., $\mathcal{L}_{\mathsf{faster}} = \mathcal{L}_{\mathsf{safer}}$ in this case. When the network is stable and optimistic, honest parties' DAG views can be synchronized quickly and will continuously increase $\mathcal{L}_{\mathsf{faster}}$.

7 Performance Analysis and Conclusion

As shown in Table 1, FaBFT has the merits of low communication complexity and message complexity in flexible asynchronous networks, due to the elegant combination of BlockDAG, CBC, and GPC. Compared with [9], which currently requires the lowest bandwidth and smallest number of messages in the literature, FaBFT costs only one more step in the best case and comparably steps in the asynchronous case, through cancelling the costly asynchronous-fallback and view change operations. Even better, FaBFT supports BlockDAG and a flexible network, which makes it more applicable in practice.

References

1. Byzantine broadcasts and randomized consensus (2009), https://dcl.epfl.ch/site/education/secure_distributed_computing
2. Cachin, C., Shoup, V.: Random oracles in constantinople: Practical asynchronous byzantine agreement using. In: Proceedings of the 19th ACM Symposium on Principles of Distributed Computing, no. pp. 1–26 (2000)
3. Canetti, R., Rabin, T.: Fast asynchronous byzantine agreement with optimal resilience. Association for Computing Machinery, New York, NY, USA (1993)
4. Castro, M., Liskov, B., et al.: Practical byzantine fault tolerance. In: OsDI. vol. 99, pp. 173–186 (1999)
5. Danezis, G., Kokoris-Kogias, L., Sonnino, A., Spiegelman, A.: Narwhal and tusk: a dag-based mempool and efficient bft consensus. In: Proceedings of the Seventeenth European Conference on Computer Systems. pp. 34–50 (2022)
6. Duan, S., Zhang, H., Sui, X., Huang, B., Mu, C., Di, G., Wang, X.: Dashing and star: Byzantine fault tolerance using weak certificates. Cryptology ePrint Archive (2022)
7. Fischer, M.J., Lynch, N.A., Paterson, M.S.: Impossibility of distributed consensus with one faulty process. Journal of the ACM (JACM) **32**(2), 374–382 (1985)
8. Gagol, A., Leśniak, D., Straszak, D., Świetek, M.: Aleph: Efficient atomic broadcast in asynchronous networks with byzantine nodes. In: Proceedings of the 1st ACM Conference on Advances in Financial Technologies. pp. 214–228 (2019)
9. Gelashvili, R., Kokoris-Kogias, L., Sonnino, A., Spiegelman, A., Xiang, Z.: Jolteon and ditto: Network-adaptive efficient consensus with asynchronous fallback. In: Financial Cryptography and Data Security: 26th International Conference, FC 2022, Grenada, May 2–6, 2022, Revised Selected Papers. pp. 296–315. Springer (2022)
10. Guo, B., Lu, Y., Lu, Z., Tang, Q., Xu, J., Zhang, Z.: Speeding dumbo: Pushing asynchronous bft closer to practice. Cryptology ePrint Archive (2022)
11. Guo, B., Lu, Z., Tang, Q., Xu, J., Zhang, Z.: Dumbo: Faster asynchronous bft protocols. In: Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security. pp. 803–818 (2020)
12. Keidar, I., Kokoris-Kogias, E., Naor, O., Spiegelman, A.: All you need is dag. In: Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing. pp. 165–175 (2021)
13. Lu, Y., Lu, Z., Tang, Q.: Bolt-dumbo transformer: Asynchronous consensus as fast as the pipelined bft. In: Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security. pp. 2159–2173 (2022)

14. Malkhi, D., Nayak, K., Ren, L.: Flexible byzantine fault tolerance. In: Proceedings of the 2019 ACM SIGSAC conference on computer and communications security. pp. 1041–1053 (2019)
15. Miller, A., Xia, Y., Croman, K., Shi, E., Song, D.: The honey badger of bft protocols. In: Proceedings of the 2016 ACM SIGSAC conference on computer and communications security. pp. 31–42 (2016)
16. Neu, J., Tas, E.N., Tse, D.: Ebb-and-flow protocols: A resolution of the availability-finality dilemma. In: 2021 IEEE Symposium on Security and Privacy (SP). pp. 446–465. IEEE (2021)
17. Spiegelman, A., Giridharan, N., Sonnino, A., Kokoris-Kogias, L.: Bullshark: Dag bft protocols made practical. In: Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security. pp. 2705–2718 (2022)
18. Yin, M., Malkhi, D., Reiter, M.K., Gueta, G.G., Abraham, I.: Hotstuff: Bft consensus with linearity and responsiveness. In: Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing. pp. 347–356 (2019)
19. Zhang, H., Duan, S.: Pace: Fully parallelizable bft from reproposable byzantine agreement. Cryptology ePrint Archive (2022)
20. Zhou, Y., Zhang, Z., Zhang, H., Duan, S., Hu, B., Wang, L., Liu, J.: Dory: Asynchronous bft with reduced communication and improved efficiency. Cryptology ePrint Archive (2022)