Ipotane: Achieving the Best of All Worlds in Asynchronous BFT

Xiaohai Dai*, Chaozheng Ding*, Hai Jin*, Julian Loss†, and Ling Ren‡
*Huazhong University of Science and Technology
†CISPA Helmholtz Center for Information Security
‡University of Illinois at Urbana-Champaign
{xhdai, chaozhengding, hjin}@hust.edu.cn, loss@cispa.de, renling@illinois.edu

***Abstract*—State-of-the-art asynchronous *Byzantine Fault Tolerance* (BFT) protocols integrate a partially-synchronous optimistic path. The holy grail in this paradigm is to match the performance of a partially-synchronous protocol in favorable situations and match the performance of a purely asynchronous protocol in unfavorable situations. Several prior works have made progress toward this goal by matching the efficiency of a partially-synchronous protocol in favorable conditions. However, their performance compared to purely asynchronous protocols is reduced when network conditions are unfavorable. To address these shortcomings, a recent work, Abraxas (CCS '23), presents the first optimistic asynchronous BFT protocol that retains stable throughput in all situations. However, Abraxas still incurs very high worst-case latency in unfavorable situations because it is slow at detecting the failure of its optimistic path. Another recent work, ParBFT (CCS '23) guarantees good latency in all situations, but suffers from reduced throughput in unfavorable situations due to its use of extra *Asynchronous Binary Agreement* (ABA) instances.**

To approach our holy grail, we propose Ipotane. Ipotane achieves performance comparable to partially-synchronous protocols in favorable situations, and attains performance on par with purely asynchronous protocols in unfavorable situations—in both throughput *and* latency. This is accomplished by our newly introduced primitive *Dual-functional Byzantine Agreement* (DBA), which packs the functions of (biased) ABA and *Validated Asynchronous Byzantine Agreement* (VABA). In the context of Ipotane, it promptly detects the optimistic path's failure and, at the same time, generates blocks on the pessimistic path with little extra work. We conduct extensive experiments to demonstrate that Ipotane achieves high throughput and low latency in all situations.

I. Introduction

The explosive popularity of blockchain technology [47], [7], [56] has reignited significant interest in *Byzantine Fault Tolerant* (BFT) consensus over the past decade [53], [29], [52]. At its core, BFT consensus empowers distributed replicas to reach an agreement on data, even in scenarios where a subset of these replicas, termed Byzantine replicas, may deviate arbitrarily from the protocol.

BFT consensus protocols traditionally fall into three categories based on their underlying network assumptions: asynchronous, partially-synchronous, and synchronous. Asynchronous protocols [42], [19], [31] ensure safety and liveness under arbitrary network conditions, whereas (partially-)synchronous protocols are prone to network attacks [42]. On the flip side, asynchronous protocols are known to be inherently randomized [23], which makes them less efficient and more challenging to design than their synchronous and partially-synchronous counterparts (e.g., PBFT [14] and HotStuff [54] which can be fully deterministic).

A. Asynchronous protocols with an optimistic path

To harness the strengths of both (partially-)synchronous and asynchronous protocols, a line of research has proposed incorporating an optimistic path into an asynchronous protocol [26], [40], [16], [10]. This typically involves using a partially-synchronous protocol, like two-chain HotStuff [34], as the optimistic path, while an asynchronous protocol, often *Validated Asynchronous Byzantine Agreement* (VABA), acts as the pessimistic fall-back path.

This dual-path paradigm considers two situations: favorable and unfavorable. A favorable situation is characterized by a non-faulty leader on the optimistic path and good network conditions, enabling the protocol to make progress through the optimistic path. On the contrary, an unfavorable situation arises when we have a faulty leader or poor network conditions, in which case the protocol will fall back to the pessimistic path to achieve liveness.

The holy grail in this dual-path paradigm is to match the performance of a partially-synchronous protocol in favorable situations and the performance of a purely asynchronous protocol in unfavorable ones. While existing protocols successfully achieve the former, a significant gap remains in achieving the latter. Specifically, earlier works like Ditto [26] and BDT [40] follow a *sequential-path* design where the pessimistic path is launched only after the optimistic path's failure is detected. This delay in launching the pessimistic path results in poor efficiency in unfavorable situations. We give a more thorough comparison with these and other existing works in Section VII.

To deal with the issues of sequential-path protocols, two recent works ParBFT [16][1] and Abraxas [10] follow a *parallel-path* design which operates the two paths simultaneously. By continuously running the pessimistic path in the background, they avoid much of the overhead encountered by the sequential-path design during unfavorable situations.

In spite of these improvements, ParBFT and Abraxas still fall short of fully matching the performance of asynchronous protocols under unfavorable situations. Concretely, ParBFT employs an individual *Asynchronous Binary Agreement* (ABA) instance at each height to detect the potential failure of the

[1]In [16], two versions of ParBFT are proposed. Our focus is on the first one, ParBFT1. Throughout this paper, we will refer to ParBFT1 simply as ParBFT, provided there is no ambiguity.

optimistic path. This achieves low latency in unfavorable situations but introduces an idle period where no new block is generated, resulting in reduced throughput (i.e., number of committed blocks) compared to a purely asynchronous protocol. On the other hand, Abraxas's pessimistic path leverages consecutive VABA instances to continuously generate blocks even during optimistic periods. Since there is no idle time, Abraxas achieves essentially the same throughput as state-of-the-art asynchronous protocols. The downside, however, is that blocks from the pessimistic path can be committed only after a special indicator transaction on the pessimistic path confirms the failure of the optimistic path. This indicator transaction is submitted only after a certain number of blocks (empirically, 20) have been generated on the pessimistic path. Thus, though Abraxas achieves high throughput *on average*, it might incur very high latency *in the worst case*.

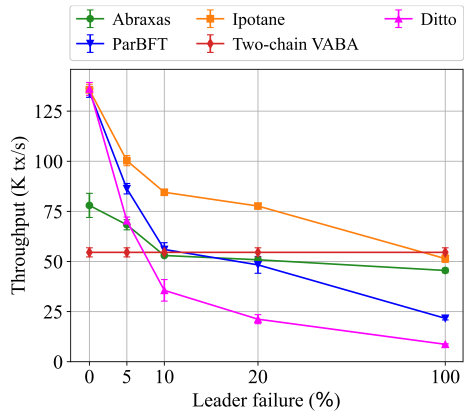
Therefore, we pose the following natural question: *Is there an asynchronous protocol whose throughput and latency are on par with the state-of-the-art of: 1) partially-synchronous protocols under favorable situations, and 2) purely asynchronous protocols in unfavorable situations?*

B. Our solution

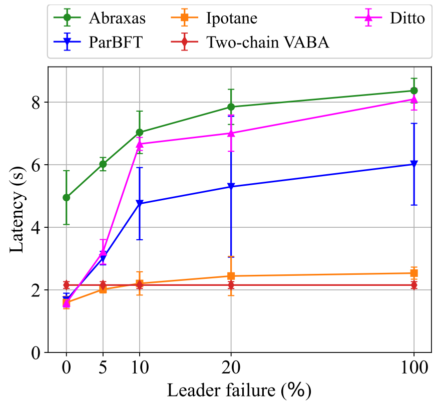
We answer this question affirmatively by proposing Ipotane. Ipotane integrates the advantages of Abraxas and ParBFT. More precisely, Ipotane delivers performance akin to that of partially-synchronous protocols under favorable situations; in unfavorable situations, Ipotane still offers throughput and latency comparable to those of a purely asynchronous protocol. In addition, in situations that fall between favorable and unfavorable, Ipotane consistently exhibits nearly the best throughput and latency among existing protocols.

Ipotane achieves these merits by utilizing a new primitive introduced in this paper: *Dual-functional Byzantine Agreement* (DBA). At its core, DBA combines the functionalities of (biased) ABA and VABA. In addition to the validated block, as required by standard VABA, the input for a DBA instance also includes a binary value. At the end of the protocol, DBA outputs pair, comprising a binary value and a block value, ensuring agreement on the pair, biased validity for the binary value, and external validity for the block value. DBA can be constructed by incorporating a *prepare* phase prior to any existing VABA protocol. This prepare phase, completed in a single communication round, biases the binary input towards 0 by amplifying this value, ensuring the safety of committing a block on the optimistic path. It is important to note that although DBA may seem somewhat similar to Cachin et al.'s VABA [13], they actually differ significantly, which will be discussed in Section III-B3. With the addition of merely one communication round, DBA manages to deliver performance similar to VABA.

With the DBA primitive defined, Ipotane operates the optimistic and pessimistic paths in parallel much like Abraxas and ParBFT. It employs the two-chain HotStuff protocol for the optimistic path and the DBA protocol for the pessimistic path. The binary decision from DBA indicates the success or failure of the optimistic path. Upon detecting a failure in the optimistic path, Ipotane promptly commits blocks on the pessimistic path, thus promising low latency. Moreover,



(a) Throughput under varying failure probability.



(b) Latency under varying failure probability.

Fig. 1: Performance under varying probabilities (ρ) of leader failure. $\rho = 0$ represents a favorable situation, where the optimistic path always operates smoothly. Conversely, $\rho = 100\%$ represents an unfavorable situation, where the optimistic path fails to make any progress.

DBA instances are comparable in efficiency to VABA, thus achieving good throughput.

Experimental results to evaluate Ipotane's performance have been shown in Fig. 1, where the x-axis represents the probability (ρ) of leader failure on the optimistic path. It shows that under favorable situations ($\rho = 0$), Ipotane achieves high throughput and low latency, matching Ditto, which operates as a purely partially-synchronous protocol in such situations. On the other hand, when leaders are always faulty ($\rho = 100\%$), Ipotane demonstrates throughput and latency on par with two-chain VABA, a purely asynchronous protocol. Furthermore, as the probability of leader failure varies, Ipotane consistently achieves almost the highest throughput and lowest latency compared to other protocols.

Table I presents a more detailed and comprehensive comparison between Ipotane and existing protocols, corroborating Ipotane's good performance with theoretical analysis. δ denotes the actual network delay, and c represents the maximum transaction count within a block. Ipotane demonstrates a low latency of 5δ and a high throughput of $c/(2\delta)$ in favorable situations. This matches the performance of a partially-synchronous protocol (specifically, two-chain HotStuff). On the other hand, in unfavorable situations, Ipotane manages to

TABLE I: Consensus performance comparison. δ and Δ denote the actual network delay and timer parameter, respectively. c and L represent the maximum transaction count and block size of a block. κ denotes the computational security parameter, while λ is the lookback parameter (empirically, 20) in Abraxas. Performance under unfavorable situations is measured assuming the adversary mounts arbitrary attacks.

	Favorable situations			Unfavorable situations		
	Latency	Throughput	Communication	Latency	Throughput	Communication
Two-chain HotStuff [26]	5δ	$c/(2\delta)$	$O(nL + n\kappa)$	/	/	/
Two-chain VABA [26]	10.5δ	$2c/(7\delta)$	$O(n^2L + n^2\kappa)$	10.5δ	$c/(7\delta)$	$O(n^2L + n^2\kappa)$
Ditto [26]	5δ	$c/(2\delta)$	$O(nL + n\kappa)$	$3\Delta + 10.5\delta$	$c/(2\Delta + 7\delta)$	$O(n^2L + n^2\kappa)$
Abraxas [10]	5δ	$c/(2\delta)$	$O(n^2L + n^2\kappa)$	$3.5\lambda\delta + 14\delta$	$c/(7\delta)$	$O(n^2L + n^2\kappa)$
ParBFT [16]	5δ	$c/(2\delta)$	$O(n^2L + n^2\kappa)$	22δ	$c/(22\delta)$	$O(n^2L + n^2\kappa)$
Ipotane	5δ	$c/(2\delta)$	$O(n^2L + n^2\kappa)$	18.5δ	$3c/(23\delta)$	$O(n^2L + n^2\kappa)$

‡ As a common implementation practice, transactions are constantly packaged and broadcast through an underlying mempool [26]. A consensus block only contains hashes of some transaction packages. Therefore, L is typically very small, approximately κ, and should not negatively impact the performance much [10].

maintain a latency of 18.5δ and a throughput of $3c/(23\delta)$. These are just slightly worse than a purely asynchronous protocol (specifically, two-chain VABA) but significantly better than prior works Abraxas in terms of latency and ParBFT in terms of throughput.

II. Models And Preliminaries

A. Model

The system consists of n replicas, with a maximum of t being Byzantine where $n \geq 3t+1$. Each replica is identified by a unique number and is denoted as p_i ($1 \leq i \leq n$). Byzantine replicas may deviate from the protocol arbitrarily and are presumed to be under the control of an adaptive adversary. This adaptive adversary can corrupt replicas as the protocol progresses and drop a corrupted replica's messages from the network a posteriori.The remaining replicas, termed non-faulty, faithfully adhere to the protocol. Each pair of replicas is connected through a pairwise authenticated communication channel. The system operates in an asynchronous network environment where no assumption is made about network delays. The adversary is assumed to have full control over the network and can arbitrarily delay and reorder any messages as long as it eventually delivers them.

A *Public Key Infrastructure* (PKI) is established across the replicas and digital signatures are used to ensure the authenticity and integrity of transmitted messages. Additionally, we employ two distinct instances of threshold signature schemes [38], [6]: one with a threshold of $n-t$, and the other with a threshold of $t+1$. The algorithm for generating a threshold signature share is denoted as $\mathsf{SignShr}$, while Comb constructs a threshold signature from sufficient shares. To simplify our notation, we omit the use of private or public keys as parameters in $\mathsf{SignShr}$ or Comb. To differentiate between the two threshold signature schemes, we use $\mathsf{SignShr}_r$ and Comb_r to denote calls to these algorithms with the threshold parameter r. We assume the adversary is computationally bounded and cannot break the security of (threshold) signatures.

B. State Machine Replication

We focus on the *State Machine Replication* (SMR) problem. Each replica p_i in SMR locally maintains a growing chain, denoted as $\mathcal{C}_i$, which is modeled as a write-once array. An object in the array is named a block, which consists of multiple transactions. Transactions are continuously generated by clients or upper-layer applications, and are inserted into a buffer of each replica i, denoted as buf_i. Transactions cached in the buffer are sorted based on the times they are received by the replica. When a replica p_i proposes a block, it selects a number of transactions from its buffer buf_i. Without loss of generality, we assume the maximum number of transactions that can be included in a block is c. Therefore, the block proposed by p_i consists of the first c transactions from the buffer, denoted $\mathsf{buf}_i[: c]$.

$\mathcal{C}_i$ is initialized as empty, namely $\mathcal{C}_i[k] = \perp$ for each index k ($k \geq 1$). A block B is said to be committed by p_i when it is written to the chain $\mathcal{C}_i$. All transactions in B are then deleted from p_i's buffer buf_i. At a high level, the SMR protocol serves to maintain a consistent chain among non-faulty replicas. We adopt the definition in Blum et al. [9], [10]:

Definition 1. *Let Π be a protocol executed among replicas $p_1, ..., p_n$, where each non-faulty replica holds a transaction buffer buf_i. We say that Π implements SMR if it satisfies the following properties:*

- ***Completeness:*** *For each index k ($k \geq 1$) and each non-faulty replica p_i, eventually either buf_i remains forever empty or $\mathcal{C}_i[k] \neq \perp$.*
- ***Consistency:*** *For two non-faulty replicas p_i and p_j, if $\mathcal{C}_i[k] \neq \perp$ and $\mathcal{C}_j[k] \neq \perp$, then $\mathcal{C}_i[k] = \mathcal{C}_j[k]$.*
- ***Liveness:*** *If a transaction tx is added to every non-faulty replica's buffer, then every non-faulty replica will eventually commit a block containing tx.*

If Π can achieve the above properties of completeness, consistency, and liveness in the presence of t Byzantine replicas, Π is named a t-BFT SMR protocol.

C. (Biased) Asynchronous Binary Agreement

The *Asynchronous Binary Agreement* (ABA) abstraction [8], [24] represents the most basic form of asynchronous BFT consensus, which serves to agree on a binary value. To be more specific, an ABA protocol is defined as follows:

Definition 2. *Let Π be a protocol executed among replicas $p_1, ..., p_n$, where replica p_i holds a binary input b_i and generates an output. We say that Π achieves ABA if it satisfies* *the following properties in an asynchronous network whenever at most t replicas are corrupt:*

- ***Agreement:*** *If two non-faulty replicas output values b and b', then $b = b'$.*
- ***Termination:*** *All non-faulty replicas eventually output.*
- ***Validity:*** *If all non-faulty replicas input the same bit b, then each non-faulty replica outputs b.*

As mentioned by Cachin et al. [13], the ABA protocol can be adapted to exhibit a bias towards 0 by adding a 'biased validity' property, which is defined as follows:

- ***Biased validity:*** If at least $t + 1$ non-faulty replicas input the bit 0, all non-faulty replicas will output 0.

An ABA protocol achieving the biased validity property is termed a biased ABA. Our design does not utilize a biased ABA directly. Instead, we introduce a new abstraction named DBA that incorporates properties akin to those in biased ABA, which is detailed in Section III.

D. Validated Asynchronous Byzantine Agreement

Different from ABA, the *Validated Asynchronous Byzantine Agreement* (VABA) abstraction facilitates consensus on arbitrary values [13]. VABA introduces an external validation predicate Q, typically defined by the higher-layer applications. To be more specific, VABA is defined as follows.

Definition 3. *Let Π be a protocol executed among replicas $p_1, ..., p_n$, where replica p_i holds input v_i and replicas terminate upon generating output. We say that Π achieves VABA if it satisfies the following properties in an asynchronous network whenever at most t replicas are corrupt:*

- ***Agreement:*** *If two non-faulty replicas output values v and v', then $v = v'$.*
- ***Termination:*** *All non-faulty replicas eventually output.*
- ***External validity:*** *If a non-faulty replica outputs v, then $Q(v)$ must be* True.
- ***Quality:*** *If a non-faulty replica outputs v, then with probability over $1/2$, v is input by some non-faulty replica.*

Various implementations of the VABA abstraction [13], [2], [41], [31] have been developed over the past decades. Note that while the quality property is not explicitly defined in [13], [31], both works implicitly guarantee this property. In our approach, we can employ any existing VABA protocol with an added validation predicate R. Specifics of this additional predicate R will be discussed in Section III-B1.

III. BUILDING BLOCK: DBA

A. Definition of DBA

We propose a new abstraction called *Dual-functional Byzantine Agreement* (DBA), which serves as a fundamental building block in Ipotane. At its core, DBA combines the functionalities of biased ABA and VABA. Through a DBA instance, replicas can simultaneously achieve consensus on a binary value as well as an arbitrary value. In the context of this paper, the arbitrary value is typically a block. Therefore, within the remainder of this paper, we will use the term "block" to represent the arbitrary value in DBA. The input to DBA consists of a triplet, $\langle b, \sigma, B \rangle$, where b is a binary value and B represents a block. If b equals 0, then σ is a proof that b satisfies an external predicate P. Conversely, if b is 1, σ defaults to $\perp$. The protocol's output is a pair $\langle b, B \rangle$. The "bias" is reflected in a preference to 0 in the binary value: as long as $t + 1$ non-faulty replicas input 0, DBA will output 0. Formally, a DBA protocol is defined as follows:

Definition 4. *Let Π be a protocol executed among replicas $p_1, ..., p_n$, where replica p_i holds a binary value b_i, a proof σ, plus a block B_i as input, and generates a binary value b and a block B as output. Replicas terminate upon generating output. We say that Π achieves DBA if it satisfies the following properties whenever at most t replicas are corrupt:*

- ***Agreement:*** *For any two non-faulty replicas outputting $\langle b, B \rangle$ and $\langle b', B' \rangle$, it must hold that $b = b'$ and $B = B'$.*
- ***Termination:*** *All non-faulty replicas eventually output.*
- ***External validity:*** *For any output $\langle *, B \rangle$ from a non-faulty replica, $Q(B) =$* True.
- ***Quality:*** *If a non-faulty replica outputs $\langle *, B \rangle$, then with probability over $1/2$, B is input by some non-faulty replica.*
- ***Biased validity.*** *If at least $t + 1$ non-faulty replicas input $\langle 0, * \rangle$, then all non-faulty replicas will output $\langle 0, * \rangle$.*
- ***Proof validity:*** *If a non-faulty replica outputs $\langle 0, * \rangle$, at least one replica (Byzantine or non-faulty) must have inputted $\langle 0, \sigma, * \rangle$ with $P(\sigma) =$* True.

To aid presentation, in the context of DBA's input, b plus σ is referred to as the "binary input", while B is termed the "block input". Correspondingly, in the output $\langle b, B \rangle$, we refer to b and B as the "binary output" and "block output", respectively. It is important to note that the properties of biased validity and proof validity specifically pertain to DBA's binary values, while quality and external validity are applicable to the block values. Besides, DBA discards the validity property defined in the original (biased) ABA abstraction. This implies that even if all non-faulty replicas input a binary value of 1 but some faulty replica inputs 0, DBA can output 0.

B. Construction of DBA: AlgDBA

Since the combination of binary input and block input can be regarded as a single value, an initial approach to developing the DBA protocol might involve adapting an existing VABA protocol to accept the combined inputs as a singular value. This approach could fulfill most of the properties outlined in Definition 4, but it falls short of meeting the biased validity requirement. To solve this problem, we introduce a *prepare* phase prior to executing the VABA protocol.

At a high level, our construction of the DBA protocol involves two steps. Firstly, we propose a variant of VABA, termed VABA*, which extends any existing VABA protocol with minor modifications. Subsequently, by adding a prepare phase to VABA*, we end up with a construction of DBA, named AlgDBA.

Algorithm 1: AlgDBA_h: A construction of DBA with the instance identity h (for replica p_i)

```
1  Let $\langle b_i, \sigma_i, B_i \rangle$ denote the input of $p_i$.

2  if $b_i = 0$ then
3      broadcast $(\texttt{PREP}, h, 0, \sigma_i, \mathsf{SignShr}_{t+1}(h, 0))$;
4  else
5      broadcast $(\texttt{PREP}, h, 1, \sigma_i, \mathsf{SignShr}_{n-t}(h, 1))$;

6  on receiving $(\texttt{PREP}, h, 0, \sigma_j, *)$ from $p_j$:
       // amplify the bit of 0
7      if $p_i$ has not broadcast 0 then
8          broadcast $(\texttt{PREP}, h, 0, \sigma_j, \mathsf{SignShr}_{t+1}(h, 0))$;
9  on receiving $t+1$ $(\texttt{PREP}, h, 0, \sigma_j, *)$:
10     $S \leftarrow$ all the signature shares from $t+1$ messages;
11     $sig_0 \leftarrow \mathsf{Comb}_{t+1}(h, 0, S)$;
12     input $\langle 0, sig_0, B_i \rangle$ to $\mathsf{VABA}^*_h$ if it has not done;
13 on receiving $n-t$ $(\texttt{PREP}, h, 1, *, *)$:
14     $S \leftarrow$ all the signature shares from $n-t$ messages;
15     $sig_1 \leftarrow \mathsf{Comb}_{n-t}(h, 1, S)$;
16     input $\langle 1, sig_1, B_i \rangle$ to $\mathsf{VABA}^*_h$ if it has not done;

17 on outputting $\langle b, sig, B \rangle$ from $\mathsf{VABA}^*_h$:
18     output $\langle b, B \rangle$;
```

1) VABA:* This modified version of VABA introduces a triplet as the input and output, denoted as $\langle b, sig, B \rangle$. Here, b is a bit, sig is a threshold signature on b, and B is a block. Let Q denote the original validation predicate in VABA. For VABA*, we introduce a new global validation predicate R, which takes $\langle b, sig \rangle$ as input and returns a Boolean value indicating validation success. When $b = 0$, R yields True if sig is a valid $(t+1)$-threshold signature on 0 and False otherwise. Conversely, for $b = 1$, R returns True if sig is a valid $(n-t)$-threshold signature on 1, and False otherwise.

In VABA*, upon receiving a message containing $\langle b, sig, B \rangle$, the replica uses R to validate $\langle b, sig \rangle$ and Q to validate B. The message is deemed valid only if both R and Q return True. Once the message is validated, the triplet $\langle b, sig, B \rangle$ is regarded as a whole, and is input to VABA. Ultimately, a triplet that satisfies both R and Q is output by VABA, which is also the output of VABA*.

2) AlgDBA *protocol:* This protocol, given in Algorithm 1, adds a prepare phase to amplify the bit of 0, prior to executing VABA*. In this phase, each replica broadcasts its binary input accompanied by a threshold signature share, through a PREP message. If the binary input is 0, the threshold parameter for the signature share is set to $t+1$. Conversely, for a binary input of 1, the threshold is set to $n-t$ (see Lines 2-5 in Algorithm 1). If a replica receives a valid PREP message containing 0, it will also broadcast 0 if it has not yet done so (Lines 6-8), which amplifies the broadcast of 0.

At the end of this phase, if a replica gathers $t+1$ PREP messages containing 0, it creates a complete threshold signature sig_0 based on signature shares in these messages, certifying the bit 0. The replica then uses $\langle 0, sig_0, B \rangle$ as input to the VABA* instance, where B is the block input of AlgDBA (Lines 9-12). Alternatively, if it receives $n-t$ valid values of 1, it creates a complete threshold signature sig_1 for the value 1, leading to the input $\langle 1, sig_1, B \rangle$ for the following VABA* instance (Lines 13-16). Finally, VABA* outputs one of these inputs, with the bit and block values forming the output of AlgDBA (Lines 17-18).

3) Relation to Cachin et al. [13]: Cachin et al. [13] first introduced VABA and biased ABA. While we draw inspiration from their work, our work is also significantly different from theirs. The end goal of Cachin et al. [13] is to construct VABA, and they defined and used biased ABA in that process. In contrast, our DBA is a new primitive that simultaneously achieves agreement on a binary value and a block value. To obtain DBA, we modify existing VABA protocols and use them as building blocks.

C. *Correctness analysis of* AlgDBA

In this section, we prove our AlgDBA construction adheres to all the properties outlined in Section III-A. First, we establish that VABA* inherits all properties from VABA. This is because VABA* only adds an additional validation for the binary value of the input. If the input originates from a non-faulty replica, this extra validation will definitely return True, making VABA* functionally identical to VABA.

1) Agreement, quality, and external validity: These three properties of AlgDBA are derived directly from the properties of VABA*.

2) Termination: AlgDBA's termination property is stated in Theorem 2, supported by Lemma 1. Due to space limitations, Lemma 1's proof is provided in Appendix Section A.

LEMMA 1. *Every non-faulty replica will receive either* $t+1$ *values of* 0 *or* $n-t$ *values of* 1 *during the prepare phase.*

THEOREM 2. AlgDBA *fulfills DBA's termination property.*

Proof: Based on Lemma 1, every non-faulty replica can generate a valid input for VABA*. Following the termination property of VABA*, all non-faulty replicas will eventually produce an output from VABA* and thus from AlgDBA, affirming the termination property. ■

3) Proof validity: The proof is established by contradiction. Suppose a non-faulty replica outputs $\langle 0, * \rangle$, but no replica inputs 0 with a valid proof σ. In such a case, no one can receive $t+1$ messages of $(\texttt{PREP}, 0, *)$ to create a valid signature sig_0 or form a valid input of 0 to VABA*, as described in Lines 9-12 in Algorithm 1. Consequently, the binary output from VABA* or AlgDBA cannot be 0, contradicting the initial assumption.

4) Biased validity: If at least $t+1$ non-faulty replicas input $\langle 0, * \rangle$, it implies that at most $n-t-1$ replicas, whether non-faulty or Byzantine, will input $\langle 1, * \rangle$. Hence, the condition in Line 13 of Algorithm 1 will not be met. Even a Byzantine replica cannot forge a valid threshold signature on 1. Therefore, every replica, whether non-faulty or Byzantine, can only input a tuple containing the bit 0 to VABA*. This ensures that the output from VABA* and AlgDBA will contain the bit 0, thus guaranteeing biased validity.

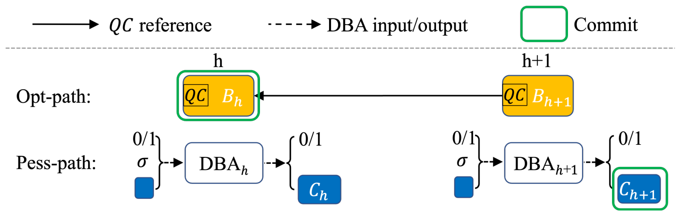
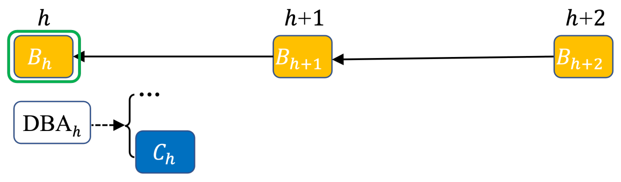
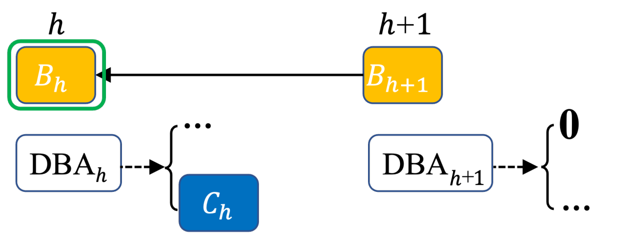


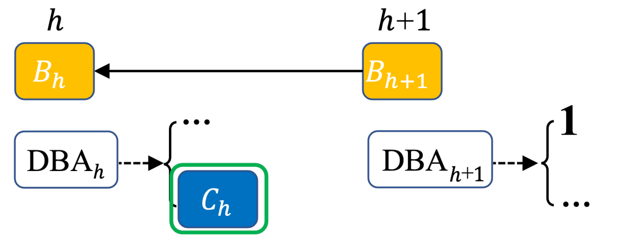
Fig. 2: The structure of an epoch in Ipotane.



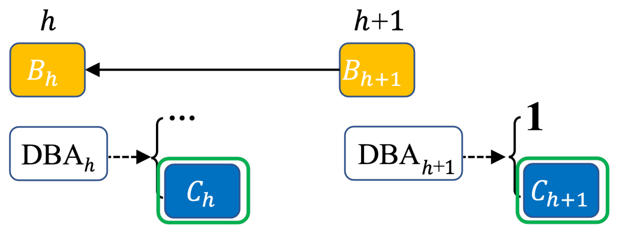
(a) Commit B_h through the two-chain mechanism.



(b) DBA$_{h+1}$ outputs 0 indicating the readiness to commit B_h.



(c) DBA$_{h+1}$ outputs 1 indicating the readiness to commit C_h.



(d) Commit both C_h and C_{h+1} when DBA$_{h+1}$ outputs 1.

Fig. 3: Examples to show the block committing rules. We omit some elements in the figures for conciseness.

IV. Ipotane Design

A. Overview and intuition

Ipotane operates in epochs, designated by incrementing integer identifiers starting from 1. Each epoch comprises an optimistic path and a pessimistic path in parallel, as depicted in Fig. 2. The optimistic path employs a structure of chain-based blocks, where the *Quorum Certificate* (QC) for a block is encapsulated within the next block. The pessimistic path is implemented through consecutive DBA instances, each producing a block and a binary value. Blocks generated in the two paths are referred to as "opt-blocks" and "pess-blocks", respectively. Opt-blocks within an epoch are numbered with heights starting from 1, denoted as B_h. Similar to a partially-synchronous protocol, a leader is designated for each height on the optimistic path, following a round-robin manner. DBA instances and their outputted pess-blocks in an epoch are also numbered starting from 1, denoted as DBA$_h$ and C_h, respectively.

1) Design intuition: In the context of Ipotane, we make a clear distinction between the terms "certify" and "commit" in relation to a block. An opt-block is deemed "certified" when the corresponding QC is obtained, and a pess-block is considered "certified" if it is outputted from a DBA instance. Taking Fig. 2 as an example, the opt-block B_h is certified, since the QC for it is contained in B_{h+1}. Both the pess-blocks C_h and C_{h+1} are certified, as they are outputted from DBA$_h$ and DBA$_{h+1}$, respectively. Due to the quorum intersection argument, the opt-block at a given height will be unique. Additionally, according to the consistency property of DBA, the pess-block at a given height will also be unique. Conversely, "commit" denotes that a block, either an opt-block or a pess-block, is eligible to be written to the SMR chain $\mathcal{C}$.

Within this parallel-path structure, it is possible to have two certified blocks at the same height, h: an opt-block B_h and a pess-block C_h. A primary task is to decide which block to commit. This is precisely the reason why we augmented VABA to DBA to make a binary decision. In particular, we will leverage the binary output from the DBA instance **at the next height**, namely DBA$_{h+1}$, to commit the block at height h. On the other hand, to attain performance comparable to a partially-synchronous protocol under favorable situations, Ipotane must be capable of rapidly committing blocks through the optimistic path, particularly employing the two-chain mechanism akin to two-chain HotStuff [26]. Thus, two distinct rules for block committing co-exist: one using binary outputs on the pessimistic path, and the other using the two-chain mechanism on the optimistic path.

The next challenge is to ensure consistency between these two commit rules. Specifically, for a given height, if a replica commits an opt-block using the two-chain mechanism, we must ensure that another replica will also commit this opt-block even if it follows the pessimistic path's binary output. This consistency is achieved through the biased-validity property of DBA. In short, if a replica commits B_h via the two-chain mechanism, then at least $t+1$ non-faulty replicas have inputted 0 to DBA$_{h+1}$, signaling their intention to commit B_h. Due to the biased-validity property, DBA$_{h+1}$ will output 0, which indicates committing B_h.

2) Overall design: Each replica participates in both the optimistic and pessimistic paths. The optimistic path, resembling the two-chain HotStuff protocol, involves designated leaders proposing opt-blocks, which are then voted by replicas using threshold signature shares. The pessimistic path, on the other hand, consists of consecutive DBA instances. The input for a DBA instance (DBA_{h+1}) depends on which block at the preceding height h—either opt-block B_h or pess-block C_h—gets certified first. An opt-block is certified by a QC contained in the subsequent opt-block, whereas a pess-block is certified upon being outputted from a DBA instance. Therefore, a replica's binary input for DBA_{h+1} hinges on which of these two events occurs first: (1) receipt of B_{h+1} or (2) output from DBA_h. If B_{h+1} is received earlier, it inputs 0 to DBA_{h+1}; otherwise, it inputs 1.

Committing an opt-block or a pess-block is based on either the two-chain mechanism or the output from the DBA instance. As depicted in Fig. 3a, upon receiving an opt-block B_{h+2}, a replica can immediately commit the opt-block from two heights prior (B_h) via the two-chain mechanism. On the other hand, if a replica receives 0 from DBA_{h+1}, as illustrated in Fig. 3b, it can commit the opt-block **at the preceding height** (B_h). Otherwise (namely if DBA_{h+1} outputs 1), the replica commits the pess-block **at the preceding height** (C_h), as shown in Fig. 3c.

Additionally, the 1 output from DBA_{h+1} indicates a failure in the optimistic path. In this scenario, each replica concludes the current epoch and progresses to the next. To enhance throughput, the pess-block C_{h+1} generated from DBA_{h+1} is also committed together with C_h. As demonstrated in Fig. 3d, both C_h and C_{h+1} are committed when DBA_{h+1} outputs 1.

In favorable situations, Ipotane continuously commits blocks through the two-chain mechanism, achieving performance akin to partially-synchronous protocols. In contrast, under unfavorable situations, Ipotane remains capable of committing blocks using the pessimistic path, thereby ensuring liveness. Since the DBA protocol can be effectively constructed based on a VABA protocol with efficient modifications, the DBA instance can offer performance comparable to VABA, enabling Ipotane to match the performance of purely asynchronous protocols in unfavorable situations.

B. Data structures and utilities

We describe data structures and utilities in this section, which are summarized as Algorithm 2. An opt-block B_h on the optimistic path is characterized by the data structure $\{h, QC, d\}$, where h represents its height number, QC is a certificate for the preceding block B_{h-1}, and d denotes a transaction batch from the buffer buf.

On the pessimistic path, each replica can generate a transaction batch at a height h, serving as the block input to the DBA_h instance. From these block inputs, only one is outputted from DBA_h and is referred to as "certified", denoted as C_h. Consequently, a replica's input I to the DBA_h instance follows the format $\{b, \sigma, C\}$, where b is a binary value indicating its opinion on which block at height $h-1$ is certified earlier, and C denotes the block input. If $b = 0$, the replica believes the opt-block B_{h-1} is certified earlier, and σ is set to QC of B_{h-1}. Otherwise ($b = 1$), the replica believes that the pess-block C_{h-1} is certified earlier, leaving $\sigma = \perp$. The output from DBA_h is consistent across replicas, and has the format $\{b, C\}$,

Algorithm 2: Data structures & utilities for p_i

```
struct Opt-Block:
    $\{h, QC, d\}$
struct DBAInput:
    $\{b, \sigma, C\}$
struct DBAOutput:
    $\{b, C\}$

define $GenOptBlk(h, QC)$:
    $d \leftarrow$ GenTxBatch();
    $B.h \leftarrow h$; $B.QC \leftarrow QC$; $B.d \leftarrow d$;
    return $B$;
define $InvokeDBA(h, b, \sigma)$:
    $d \leftarrow GenTxBatch()$;
    $I.b \leftarrow b$; $I.\sigma \leftarrow \sigma$; $I.C \leftarrow d$;
    invoke $\mathsf{DBA}_h$ with $I$;
define $GenTxBatch()$:
    $d \leftarrow$ a batch of transactions from $\mathsf{buf}_i$;
    return $d$;
define $CommitBlk(blk)$:
    $len \leftarrow \mathcal{C}_i.len()$;
    $\mathcal{C}_i[len+1] \leftarrow blk$;
    delete $\mathsf{tx}$ from $\mathsf{buf}_i$ for each $\mathsf{tx} \in blk$;
```

where b is a bit indicating the agreed-upon result regarding which block at height $h-1$ is certified earlier. C is a block output derived from one of the block inputs. For convenience, we omit the height numbers in data structures of DBA inputs and outputs. Instead, their heights are implied by the height numbers of DBA instances. For example, "invoking DBA_h with I" implies I has a height number h, and "DBA_h outputs O" implies O has a height number h.

We also define some functions that are utilized in Ipotane, including $GenOptBlk$, $InvokeDBA$, and $CommitBlk$. Both $GenOptBlk$ and $InvokeDBA$ need to extract a batch of transactions from the replica's transaction buffer buf, which is achieved by calling the $GenTxBatch$ function.

C. Detailed design when $h > 1$

Algorithm 3 outlines an epoch in Ipotane, which operates in consecutive heights[2]. This subsection describes the general protocol for heights greater than 1, while special considerations for the first height will be discussed in the next subsection.

The external validity function P in DBA is defined as a $(n-t)$-threshold signature verification function. The binary input or output in the DBA instance is 0 if the corresponding opt-block is certified earlier than the pess-block, and is 1 otherwise. In other words, a replica inputs 0 if it believes the optimistic path is functioning well and inputs 1 if it perceives lack of progress with the optimistic path. An output of 0 from a DBA instance indicates agreement among replicas that the optimistic path performs well, while an output of 1 indicates agreement that the optimistic path has encountered a failure.

[2]We put termination and invocation of DBA (Lines 19-20 of Algorithm 3) as part of the optimistic path, as these actions are triggered by receiving an opt-block. Similarly, we put committing an opt-block (Lines 26-28) as part of the pessimistic path, as these actions are triggered by receiving a pess-block.

Algorithm 3: Protocol of an epoch in Ipotane for p_i

```
1  Let $L_h$ denote the leader of height $h$ on the opt. path.
2  $h \leftarrow 1$, $prevPessBlk \leftarrow \perp$.

   // optimistic path
3  if $p_i$ is $L_1$ then
4      $B_1 \leftarrow GenOptBlk(1, \perp)$;
5      broadcast $B_1$;

   // pessimistic path
6  $InvokeDBA(1, 0, \perp)$;

   // optimistic path
7  on receiving $B_1$:
8      send $\mathsf{SignShr}_{n-t}(B_1)$ to $L_2$;
9  on receiving n-t sign. shares on $B_k$ (denoted as $S$):
10     if $p_i$ is $L_{k+1}$ then
11         $QC \leftarrow \mathsf{Comb}_{n-t}(B_k, S)$;
12         $B_{k+1} \leftarrow GenOptBlk(k+1, QC)$;
13         broadcast $B_{k+1}$;

14 while the epoch is not concluded:
15     wait until $B_{h+1}$ is received or $\mathsf{DBA}_h$ outputs $O$;
16     if $B_{h+1}$ is received earlier then
           // optimistic path
17         $CommitBlk(B_{h-1})$ if $h \geq 2$;
18         send $\mathsf{SignShr}_{n-t}(B_{h+1})$ to $L_{h+2}$;
19         stop participating in $\mathsf{DBA}_{h-1}$ if $h \geq 2$;
20         $InvokeDBA(h+1, 0, B_{h+1}.QC)$;
21         broadcast $B_{h+1}$ if it has not done yet;
22     else
           // pessimistic path
23         if $O.b = 0$ then
24             stop participating in the optimistic path;
25             $InvokeDBA(h+1, 1, \perp)$;
26             if $h \geq 2$ and $B_{h-1}$ is not committed then
27                 wait until $B_{h-1}$ is received;
28                 $CommitBlk(B_{h-1})$;
29             $prevPessBlk \leftarrow O.C$;
30         else
31             $CommitBlk(prevPessBlk)$;
32             $CommitBlk(O.C)$;
33             conclude the epoch;
34     $h \leftarrow h+1$;
```

For a height, consider two time points for replica p_i:

- t_1**:** the time when the opt-block B_h is certified, indicated by receiving an opt-block B_{h+1}
- t_2**:** the time when the pess-block C_h is certified, indicated by receiving the output from the DBA_h instance.

If the optimistic path operates effectively, the DBA instance at height h, namely DBA_h, will be launched upon the reception of the opt-block B_h. Subsequently, it takes 2δ for B_h to be certified by the QC contained in the subsequent opt-block B_{h+1}, whereas a minimum of 7δ is required for DBA_h to output the certified pess-block C_h. Therefore, we should have $t_1 < t_2$ when the optimistic path is functioning well. The comparison between t_1 and t_2 hence serves as an indicator of whether the optimistic path is working well. p_i takes different actions based on this comparison.

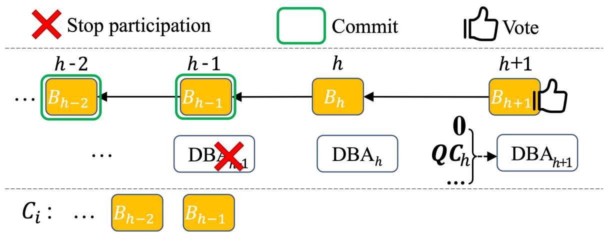


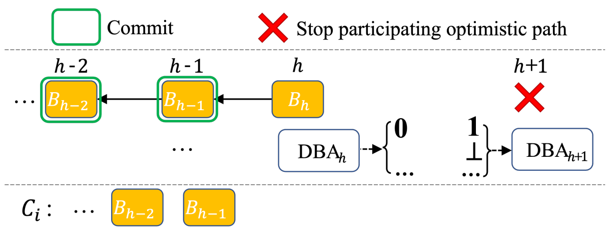
Fig. 4: Actions taken when $t_1 < t_2$ in Ipotane.

1) Case 1: $t_1 < t_2$: In this case, p_i receives B_{h+1} earlier than the output from DBA_h, indicating that the optimistic path works well as expected. p_i leverages the two-chain mechanism to commit block B_{h-1} (when $h \geq 2$) and casts a vote for the received block B_{h+1}, as described in Fig. 4 and Lines 17-18 in Algorithm 3. Additionally, p_i stops participating in the DBA_{h-1} (when $h \geq 2$) instance (Line 19). Furthermore, p_i inputs to DBA_{h+1} its opinion that B_h is certified earlier than C_h. To be concrete, its binary input to DBA_{h+1} is 0 plus QC of B_h contained in B_{h+1} (Line 20). We denote QC of B_h as QC_h. This ensures the consistency of committed blocks. Intuitively, if a non-faulty replica commits B_h after receiving B_{h+2}, at least $t+1$ non-faulty replicas must have received B_{h+1} earlier and inputted 0 to DBA_{h+1}. Therefore, the biased validity of DBA guarantees that DBA_{h+1} will output 0, directing any non-faulty replica to commit B_h if it has not done so already. Besides, p_i will also broadcast B_{h+1} to make sure other replicas receive this block (Line 21).

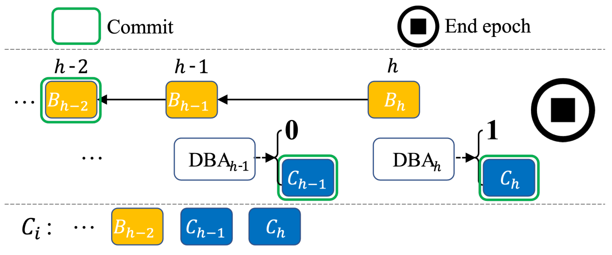
2) Case 2: $t_1 \geq t_2$: In this case, DBA_h outputs before the reception of B_{h+1}. When DBA_h outputs 0, it indicates an agreement that the optimistic path has been functioning well until height $h-1$. However, from this one replica's perspective, something is wrong with the optimistic path at height h. So the replica conveys this opinion by inputting 1 to the next DBA instance, namely DBA_{h+1}, and stops participating in the optimistic path. On the contrary, if the binary output from DBA_h is 1, signifying agreement among replicas that a failure has occurred with the optimistic path, the replica concludes the current epoch after committing pess-blocks. To delve into more details, we consider two sub-cases.

Case 2.1: DBA_h outputs 0. As illustrated in Fig. 5a and detailed in Lines 22-24 of Algorithm 3, p_i promptly stops participating in the optimistic path of this epoch. Additionally, it inputs 1 to the subsequent DBA instance expressing its opinion that the optimistic path has failed (Line 25). It can also commit the block at height $h-1$ (when $h \geq 2$), namely B_{h-1}. If it has not received B_{h-1} yet, it will wait for the reception of B_{h-1} and then commit B_{h-1}. The pseudocode for this case is described in Lines 26-28 in Algorithm 3. Furthermore, the pess-block outputted from DBA_h, namely C_h, will be cached for now (Line 29) and will be committed later if the subsequent DBA instance outputs 1.

Case 2.2: DBA_h outputs 1. This sub-case indicates agreement among replicas that the optimistic path has failed. Consequently, every replica within this sub-case commits two pess-blocks and then concludes the current epoch. Actions taken by p_i are presented in Fig. 5b and Lines 30-33 of Algorithm 3. After consecutively committing the opt-blocks



(a) DBA$_h$ outputs 0.



(b) DBA$_h$ outputs 1.

Fig. 5: Actions taken when $t_1 \geq t_2$ in Ipotane.

until B_{h-2}, p_i commits two pess-blocks, C_{h-1} and C_h. Notably, C_{h-1} has been cached in the variable $prevPessBlk$, and C_h is outputted from the current DBA, namely DBA$_h$. Subsequently, p_i concludes its participation in the current epoch and progresses to the next epoch.

D. Detailed design when $h = 1$

In the initial opt-block B_1, QC for the preceding block is set to an empty value $\perp$ (Line 4 in Algorithm 3), following the approach in two-chain HotStuff [26]. The first DBA instance, DBA$_1$, is invoked with a binary input of 0 and QC set to the empty value $\perp$, as outlined in Line 6 in Algorithm 3. Any replica that receives a message in the form of $(\texttt{PREP}, 0, \perp, \mathsf{SignShr}_{t+1}(0))$ during the DBA$_1$ instance will straightforwardly recognize this binary input of 0 as valid.

V. Analysis of Ipotane

Our analysis of Ipotane covers two main aspects: correctness and efficiency. The correctness analysis examines whether Ipotane fulfills the three properties of SMR: completeness, consistency, and liveness. Due to space constraints, detailed proofs of helper lemmas are deferred to Appendix Section A.

A. Completeness analysis

To aid presentation, we denote an iteration of the loop (Lines 15-33 in Algorithm 3) with the parameter h as $iter_h$. Theorem 6 addresses the completeness property, supported by Lemmas 3, 4, and 5.

Lemma 3. *If a non-faulty replica concludes an epoch in the iteration $iter_h$, all non-faulty replicas will also conclude that epoch in $iter_h$.*

Lemma 4. *If a non-faulty replica p_i does not conclude the epoch in or before the iteration $iter_h$, p_i will receive either an opt-block B_{h+1} or an output from* DBA$_h$.

Lemma 5. *A replica will commit at least two blocks in an epoch, as long as all non-faulty replicas enter this epoch.*

Theorem 6 (Completeness). *For each index k ($k \geq 1$), every non-faulty replica p_i eventually has $\mathcal{C}_i[k] \neq \perp$.*

Proof: If there is an epoch where p_i does not conclude the epoch in or before the iteration $iter_{k+1}$, according to Lemma 4, p_i will receive either B_{j+1} or O_j for each j (where $1 \leq j \leq k+1$). Besides, $O_j.b$ must be 0. When either B_{j+1} is received or O_j is received, an opt-block B_{j-1} will be committed. Therefore, p_i will commit at least k opt-blocks in this epoch, establishing this theorem.

Otherwise, according to Lemma 3, when a replica concludes an epoch, all non-faulty replicas will successfully conclude the current epoch and advance to the next epoch. Besides, based on Lemma 5, a replica can commit at least two blocks in an epoch as long as all non-faulty replicas enter this epoch. Therefore, after at most $\lceil k/2 \rceil$ epochs, p_i will commit at least k blocks, implying $\mathcal{C}_i[k] \neq \perp$. ■

B. Consistency analysis

The consistency property is outlined in Theorem 9, whose proof relies on Lemmas 7 and 8.

Lemma 7. *Within an epoch, if two non-faulty replicas commit two blocks at the same height, these two blocks must be identical.*

Lemma 8. *If two non-faulty replicas conclude the same epoch, they must commit the same number of blocks within that epoch.*

Theorem 9 (Consistency). *For two non-faulty replicas p_i and p_j, if $\mathcal{C}_i[k] \neq \perp$ and $\mathcal{C}_j[k] \neq \perp$, then $\mathcal{C}_i[k] = \mathcal{C}_j[k]$.*

Proof: Lemma 8 states that the epoch in which p_i commits $\mathcal{C}_i[k]$ must be the same as the epoch where p_j commits $\mathcal{C}_j[k]$, and we denote this epoch as e. Furthermore, within epoch e, the specific height at which p_i commits $\mathcal{C}_i[k]$ must be the same as the height where p_j commits $\mathcal{C}_j[k]$, denoted as h. In other words, p_i and p_j commit $\mathcal{C}_i[k]$ and $\mathcal{C}_j[k]$ at the same height within the same epoch. According to Lemma 7, it is also established that $\mathcal{C}_i[k]$ and $\mathcal{C}_j[k]$ must be identical. ■

C. Liveness analysis

We say the protocol has concluded an epoch if any non-faulty replica concludes it. According to Lemma 3, if a non-faulty replica concludes an epoch, then all non-faulty replicas will also conclude it within the same iteration. In other words, non-faulty replicas agree on the number of iterations in each epoch. We say a transaction is committed if it is included in a committed block.

As described in Section II-B, each replica's buffer organizes pending transactions in the order of their reception times. Therefore, a unique index k, starting from 1, is assigned to each transaction within buf_i. Each time a block is committed, any transaction included in this block will be removed from buf_i, and the indices of remaining transactions are adjusted downwards. Recall that in Section II-B, the maximum transaction count in a block is denoted as c. For a given transaction tx in replica p_i's buffer, the committing of p_i's newly proposed block results in one of two outcomes for tx: if tx is included within the block, it becomes committed; otherwise, the index of tx decreases by c. To unify the two cases, we define that when tx's index becomes 0 or negative, tx is committed.

Consider the moment when tx enters the buffer of every non-faulty replica and suppose tx is placed at index k_i in replica p_i's buffer. Whenever an index k_i falls to 0 or below, tx is committed by p_i. Let K represent the sum of tx's indices in the buffers of all non-faulty replicas, expressed as $K = \sum_{p_i \in H} k_i$, where H is the set of non-faulty replicas. It follows naturally that each time a block from a non-faulty replica is committed, K decreases.

THEOREM 10 (LIVENESS). *If a transaction* tx *is added to every non-faulty replica's buffer, every non-faulty replica will eventually commit a block containing* tx.

Proof: Let T_0 denote the moment when tx is added to every non-faulty replica's buffer. Let $u = \lceil K/c \rceil$. Two situations unfold:

Situation 1: At least u non-faulty opt-blocks proposed after T_0 are committed within an epoch. We assume by contradiction that tx is not committed after u non-faulty opt-blocks being committed. Each time a non-faulty opt-block is committed, K will be reduced by c. Therefore, after u non-faulty opt-blocks being committed, K will be reduced by $c \cdot u$. Since $u = \lceil K/c \rceil$, then $K - u \cdot c \leq 0$. As K represents the sum of all indices of tx in non-faulty replicas' buffers, at least one index is negative or 0, indicating that tx is committed and leading to a contradiction.

Situation 2: Less than u non-faulty opt-blocks proposed after T_0 are committed within each epoch. In this situation, each epoch is concluded after some opt-blocks and two pess-blocks are committed. DBA's quality property ensures that the probability of the outputted pess-block being proposed by a non-faulty replica is over $1/2$. Similar to Situation 1, each time a non-faulty pess-block is committed, K will be reduced by c. As the epochs advance, the probability that at least u non-faulty pess-blocks being committed will approach 1. In other words, K will keep decreasing and eventually become negative or 0. Thus, tx will eventually be committed.

To sum up, the liveness property is established. ■

D. Efficiency analysis

Recall that δ denotes the actual network delay, while c and L represent the maximum transaction count and block size of a block, respectively. Besides, we assume the size of shares and signatures to all have length κ. Our analysis focuses on the efficiency of Ipotane when employing sMVBA [31] to construct AlgDBA. Inspired by AMS-VABA [4] and two-chain VABA [26], we introduce two improvements to sMVBA. Firstly, we reduce its view-change phase from two communication rounds to just one, in a manner akin to the AMS-VABA protocol [4], effectively reducing its expected worst-case latency to 10.5 communication rounds. Secondly, we require each replica to broadcast a block within the second *Provable Broadcast* (PB) instance. For clarity, we refer to these blocks as PB2-blocks. Accordingly, original pess-blocks proposed in the first PB instance are termed PB1-blocks. When a replica commits the PB1-block proposed by the view leader (distinct from the leader of Ipotane's optimistic path), it must have received a QC for this leader's PB2-block. The replica will include this QC in its PB1-block in the subsequent sMVBA/AlgDBA instance, leading to a chain of blocks across sMVBA/AlgDBA instances, similar to two-chain VABA [26]. This way, committing a PB1-block in an AlgDBA instance will also commit a PB2-block from the preceding AlgDBA, thereby improving DBA's throughput.

1) Favorable situation: In a favorable situation, blocks are continuously committed through the optimistic path. Every 2δ interval, a new opt-block is produced, and a block from two heights prior is committed. This process results in a throughput of $c/(2\delta)$ and a latency of 5δ. Even in this favorable situation, both two paths are executed. On the optimistic path, each replica will send a signature share to leaders and broadcast its received opt-block, leading to a communication overhead of $O(n^2L + n\kappa)$. The pessimistic path consists of consecutive AlgDBA instances, leading to a communication overhead of $O(n^2L + n^2\kappa)$. Therefore, the total communication overhead in a favorable situation is $O(n^2L + n^2\kappa)$.

2) Unfavorable situation: In an unfavorable situation, only two AlgDBA instances produce outputs per epoch. For simplicity, we refer to them as AlgDBA$_1$ and AlgDBA$_2$, respectively. As AlgDBA is constructed as an extension of sMVBA with an additional prepare phase, its expected worst-case latency is 11.5 rounds. At the end of an epoch, three blocks are committed: two PB1-blocks generated in AlgDBA$_1$ and AlgDBA$_2$, respectively, and one PB2-block generated in AlgDBA$_1$. Consequently, the throughput is calculated as $3c/(11.5\delta \cdot 2) = 3c/(23\delta)$. The latency for the first PB1-block is 23δ, corresponding to the duration of two AlgDBA instances. The PB2-block, proposed two rounds later than the first PB1-block, has a latency of 21δ. The second PB1-block, committed immediately upon the output of AlgDBA$_2$, has a latency of 11.5δ. Therefore, the average latency across these blocks is $(23\delta + 21\delta + 11.5\delta)/3$, which equals 18.5δ. As for the communication overhead, the optimistic path in the unfavorable situation fails to make progress. Therefore, its communication overhead is that of the pessimistic path, which is also $O(n^2L + n^2\kappa)$.

VI. IMPLEMENTATION AND EVALUATION

In this section, we present the implementation of the Ipotane prototype and conduct a comparison with other protocols. Our chosen baselines include Abraxas and ParBFT, both of which employ the parallel-path paradigm similar to Ipotane. We include Ditto as another baseline that represents the sequential-path paradigm. In favorable situations, Ditto's performance matches that of a partially-synchronous protocol. We also include two-chain VABA, a purely asynchronous protocol, as a baseline for the evaluation of unfavorable situations.

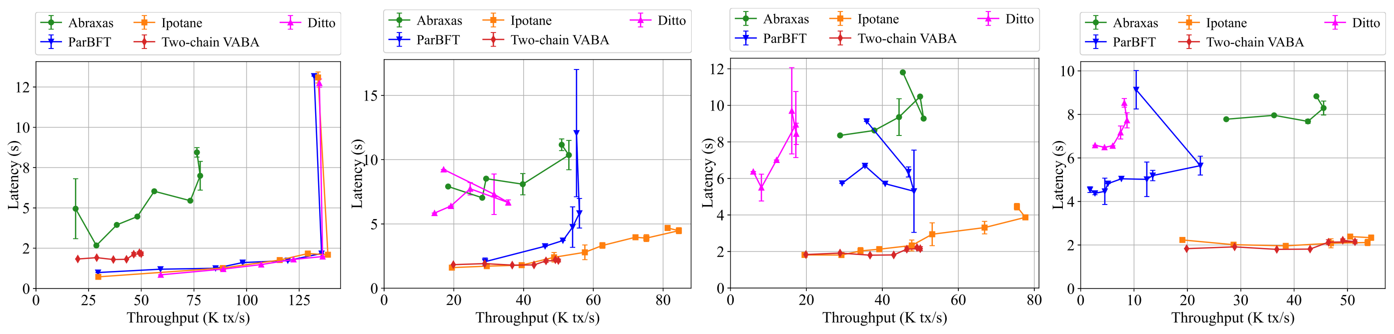
A. Implementation and experimental setup

1) Implementation: We directly adopt the available open-source codes of our baselines ParBFT[3] and Abraxas[4]. Two-chain VABA and Ditto share the same repository[5]. All these implementations are built on the same code framework in Rust, which typically includes a mempool to decouple transaction

[3]https://github.com/ac-dcz/parbft-parbft1-rust

[4]https://github.com/sochsenreither/abraxas

[5]https://github.com/danielxiangzl/Ditto



(a) $\rho = 0$ (b) $\rho = 10\%$ (c) $\rho = 20\%$ (d) $\rho = 100\%$

Fig. 6: Latency vs. throughput.

transmission from consensus messages. Through the mempool, each replica continuously packages a batch of transactions into a payload, which is then broadcast to other replicas. In the consensus message, a block contains only hashes of these payloads, effectively reducing the size of consensus messages and enhancing performance.

To ensure a fair comparison, we implement Ipotane using the same framework as the baselines. The VABA protocol employed in DBA is instantiated with sMVBA [31]. To improve performance, we introduce minor modifications to the sMVBA protocol. These include adding a block in each PB instance and chaining blocks across different sMVBA instances, a structure adopted by two-chain VABA [26]. Additionally, the view-change phase has been streamlined to a single round following the AMS-VABA protocol [4].

2) Experimental setup: For all protocols, we set the size of a transaction to 512 bytes. The size of a payload and the queue capacity in the mempool are configured to be 500 kilobytes and 100,000, respectively. The maximum number of payloads contained in a block is limited to 32, and the minimum interval to propose a block is set to 100 milliseconds. In Ditto, the timing parameter Δ is configured to 5 seconds, while the lookback parameter λ in Abraxas is fixed at 20.

Except for two-chain VABA, each protocol employs predetermined leaders for optimistic paths. Depending on how often leaders crash, we consider following three scenarios, akin to those defined in Abraxas. Each scenario is characterized by the parameter ρ, signifying the probability of leader crashes.

1) $\rho = 0$: This implies that leaders operate without any crashes. In this scenario, all protocols, except two-chain VABA, are expected to commit blocks through the optimistic paths.
2) $\rho = 100\%$: In this scenario, leaders always crash, and all protocols commit blocks through the pessimistic paths.
3) $\rho = 10\%$ or $\rho = 20\%$: Each leader has a 10% or 20% probability of crashing in this scenario, representing an intermediate point between the previous two scenarios. Optimistic protocols commit blocks through their optimistic paths intermittently.

Our experiments are conducted on Amazon Web Service, where each replica is deployed as an m5d.2xlarge instance. Each instance is equipped with 8 vCPUs and 32GiB memory, running Ubuntu 20.04 as the operating system. Replicas are connected through a network link with up to 10 Gbps bandwidth. These replicas are spread in a geographically distributed manner, uniformly across five regions: N.Virginia (us-east-1), Stockholm (eu-north-1), Tokyo (ap-northeast-1), Sydney (ap-southeast-2), and N.California (us-west-1).

3) Performance metrics: Our evaluation primarily focuses on two key metrics: end-to-end latency and throughput. End-to-end latency is assessed as the average time taken for a transaction to be committed, measured from the moment it is submitted by the client to the moment it is committed. Throughput is calculated as the number of committed transactions per second. Each experiment is conducted over a duration of 5 minutes to report a stable performance. We repeat each experiment three times and utilize the error bars or averages to mitigate experimental errors.

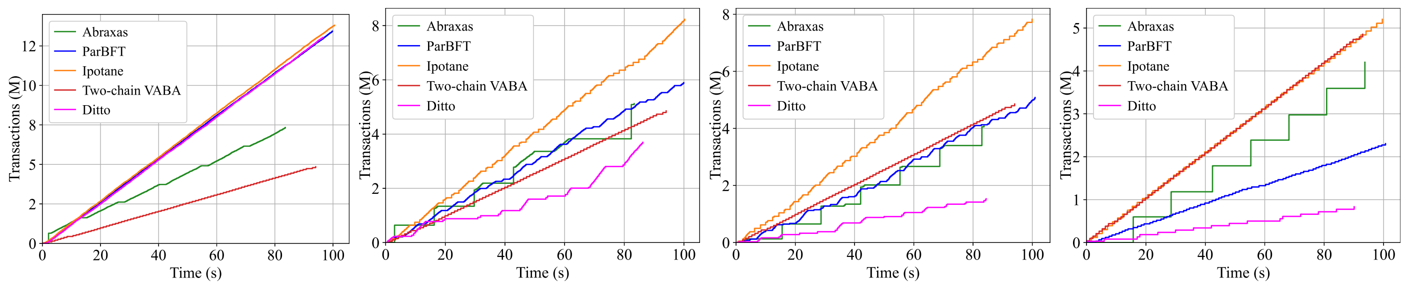
B. *Trade-off between throughput and latency*

In all experiments in this section, we set the number of replicas to 16. By progressively increasing the rate at which clients submit transactions, the system eventually becomes saturated. Plotting each pair of latency and throughput produces a figure that simultaneously demonstrates the latency under unsaturated conditions and the peak throughput under saturated conditions. Experimental results are illustrated in Fig. 6, with throughput and latency on the x-axis and y-axis, respectively. Each data point in the figure is marked with an error bar, representing both the average and standard deviation of the experimental results.

As shown in Fig. 6a, when the optimistic path always operates well, Ipotane attains low latency and high throughput, comparable to Ditto and ParBFT.[6] Notably, Ditto matches the performance of a partially-synchronous protocol, as it adopts the sequential-path paradigm and only runs the optimistic path in this scenario. Thus, in favorable situations, Ipotane's performance is on par with a partially-synchronous protocol.

At the other end of the spectrum, when the optimistic path always fails, as shown in Fig. 6d, Ipotane still maintains good performance, slightly inferior to the purely asynchronous protocol (two-chain VABA) but significantly better than Ditto or ParBFT. In this scenario, Ditto takes a considerable amount of time to switch between the failed optimistic path and the pessimistic path, resulting in poor performance. While ParBFT

[6]Abraxas reports lower performance than expected, possibly due to its implementation being based on an earlier version of Ditto.



(a) $\rho = 0$ (b) $\rho = 10\%$ (c) $\rho = 20\%$ (d) $\rho = 100\%$

Fig. 7: Throughput over time.

runs two paths concurrently, it requires additional ABA instances to commit pess-blocks. These ABA instances do not generate new blocks by themselves, leading to idle periods and reduced performance. In contrast, Ipotane commits pess-blocks by consecutively running DBA instances, which can promptly detect the optimistic path's failure without introducing extra consensus instances, thereby delivering superior performance.

In the intermediate scenarios, a protocol with an optimistic path intermittently commits blocks through this path, leading to a blended result between the scenarios of $\rho = 0$ and $\rho = 100\%$. Regarding peak throughput, Ipotane consistently outperforms other protocols as illustrated in Fig. 6b and Fig. 6c. In terms of latency, Ipotaneand two-chain VABA consistently demonstrate the lowest among these protocols.

To sum up, across all scenarios, Ipotane consistently attains the (near-)best performance among all protocols. Specifically, it achieves performance on par with partially-synchronous protocols under favorable situations and on par with purely asynchronous protocols under unfavorable situations.

C. *Throughput stability*

In this section, we evaluate the throughput stability of protocols. We continue to use 16 replicas. Our experiments are specifically conducted at each system's saturation point, where a system achieves its peak throughput without significant deterioration in latency. At this point, the system can reliably sustain high throughput.

Starting from the moment the system reaches the saturation point, we record the accumulated number of committed transactions over time. More precisely, each time a new block is committed, we record the current time and calculate the number of committed transactions by counting in transactions included in this block. We also explore four scenarios with varying values of ρ, whose results are depicted in Fig. 7.

In the $\rho = 0$ scenario, all protocols exhibit stable throughput, as evidenced by the smooth curves in Fig. 7a. This is expected, as the two-chain VABA protocol continuously commits blocks through the two-chain instances[7], while other protocols steadily commit blocks through the optimistic path. On the other hand, in the $\rho = 100\%$ scenario, two-chain VABA, ParBFT, and Ipotane can maintain stable throughput, as shown in Fig. 7d. However, Ditto and Abraxas display unstable throughput, as indicated by the jagged curves. This instability arises from the extended periods required for Ditto to complete the path switch and for Abraxas to wait for a minimum of λ pess-blocks, during which no blocks are being committed.

In the $\rho = 10\%$ or $\rho = 20\%$ scenario, all protocols except VABA exhibit less stable throughput as they alternate between committing blocks through the optimistic path and the pessimistic path. Nevertheless, Ipotane continues to showcase superior stability than Abraxas and Ditto.

D. *Latency stability*

Latency stability holds significant importance for upper-layer applications, as unstable latency can result in poor user experience. We evaluate latency stability by recording latency of each transaction. These experiments are also conducted with 16 replicas and at each system's saturation point.

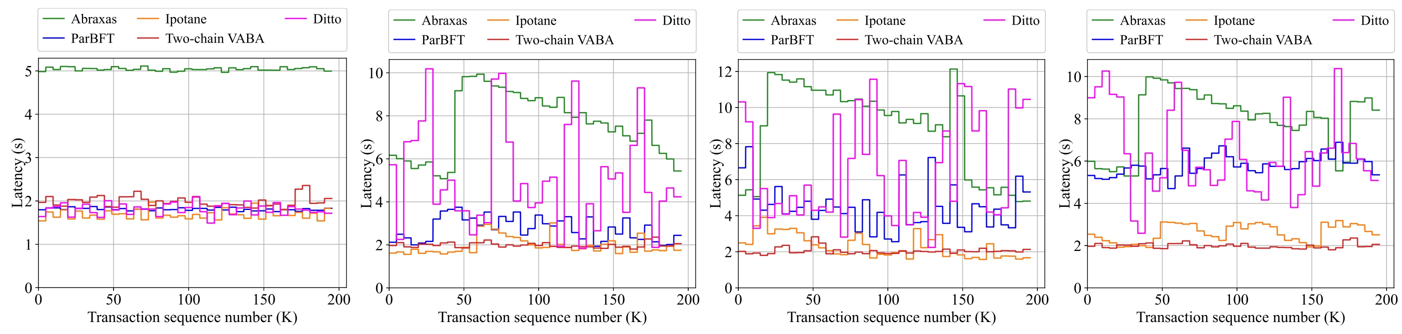
Experimental results are depicted in Fig. 8. In the $\rho = 0$ scenario (Fig. 8a), all protocols exhibit stable latency. In the $\rho = 100\%$ scenario (Fig. 8d), Ipotane maintains relatively stable latency by committing pess-blocks through successive DBA instances. While Ipotane's latency deviation is slightly larger than that of two-chain VABA, it is more stable than other protocols. Ipotane's slightly larger deviation than VABA can be attributed to the fact that, within an epoch, pess-blocks generated in the second-to-last DBA instance are not committed until the final DBA instance outputs, resulting in higher latency for these pess-blocks. In contrast, Abraxas displays significant latency fluctuations due to its lookback mechanism, where blocks generated in the initial two-chain instance must wait for the production of at least λ subsequent blocks. ParBFT and Ditto both exhibit notable latency instability, because their respective ABA instances and path-switch mechanisms introduce considerable latency variations.

In the $\rho = 10\%$ (Fig. 8b) and $\rho = 20\%$ (Fig. 8c) scenarios, Ipotane and VABA still maintain stable latency, with fluctuation significantly lower than other protocols.

E. *Scalability evaluation*

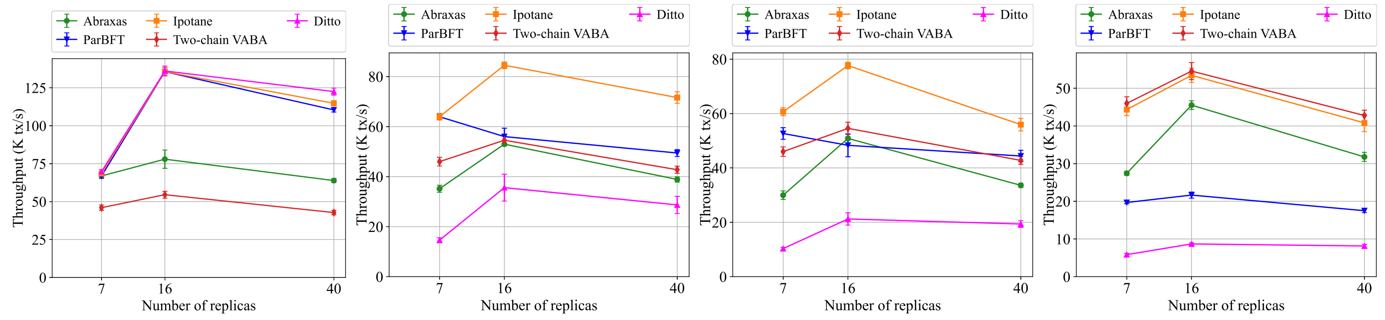
We conducted a comprehensive evaluation of scalability across various protocols, analyzing their throughput under varying numbers of replicas: 7 replicas, 16 replicas, and 40 replicas. Throughput measurements were specifically taken at the saturation point. Additionally, experiments were carried out with different probabilities of leader replicas, and the results are illustrated in Fig. 9.

[7]A two-chain instance refers to two consecutive blocks plus a leader nomination [26].



(a) $\rho = 0$ (b) $\rho = 10\%$ (c) $\rho = 20\%$ (d) $\rho = 100\%$

Fig. 8: Latency over the transaction sequence numbers.



(a) $\rho = 0$ (b) $\rho = 10\%$ (c) $\rho = 20\%$ (d) $\rho = 100\%$

Fig. 9: Throughput vs. system size.

As depicted in Fig. 9a, Ipotane, alongside ParBFT, achieves a high throughput when leaders on the optimistic path keep performing well. Notably, in the case of 40 replicas, they exhibit a slightly lower throughput compared to Ditto, potentially attributed to their elevated communication overhead $O(n^2)$, stemming from the parallel pessimistic path. In scenarios where the optimistic path fails to function, as illustrated in Fig. 9d, Ipotane consistently maintains high throughput comparable to two-chain VABA, across different replica counts. When leaders on the optimistic path fail with a probability of 10% or 20%, Ipotane outperforms all other protocols under varying system sizes, as evidenced by Fig. 9b or Fig. 9c. In summary, Ipotane consistently demonstrates excellent scalability across diverse probabilities of leader failures.

VII. Related Work

We summarize asynchronous BFT protocols in this section and defer the discussion of (partially-)synchronous protocols to Section B in the Appendix.

The simplest form of asynchronous BFT is ABA, which reaches an agreement on binary values [48], [51], [43], [1]. VABA and MVBA instead focuses on agreeing on arbitrary values [13], [2], [41], [25]. Building upon ABA or VABA, *Asynchronous Common Subset* (ACS) and SMR can be constructed [42], [32], [55], [20].

Despite efforts to enhance the performance of asynchronous protocols, a performance gap persists when compared to partially-synchronous protocols. To address this gap, a series of works introduce an optimistic path to asynchronous protocols, categorized into two paradigms: sequential-path and parallel-path. The sequential-path paradigm executes the optimistic and pessimistic path in sequence [5], necessitating path switches [26], [40]. These switches delay the launch of the pessimistic path and affect performance in unfavorable situations. To overcome this, the parallel-path paradigm, exemplified by Abraxas [10] and ParBFT [16], launches two paths simultaneously, avoiding the need for path switches. However, while Abraxas achieves high throughput in all situations, it suffers from high latency under unfavorable situations. In contrast, ParBFT consistently delivers low latency but suffers from reduced throughput in unfavorable situations. Ipotane proposed in this paper achieves both high throughput and low latency in both favorable and unfavorable situations.

Another class of protocols [35], [49], [17] leverages a *Directed Acyclic Graph* (DAG)-based approach. These protocols, however, inherently suffer from $O(n^2L+n^3\kappa)$ communication overhead, rendering them less scalable compared to many previously discussed protocols. Besides, these approaches generally depend on multiple rounds of *Reliable Broadcast* (RBC) to achieve consensus, resulting in high latency. For instance, DAGRider and Tusk require latencies of 12δ and 9δ, respectively, even under favorable situations. BullShark [50], a noteworthy DAG-based protocol, also introduces an optimistic path to enhance performance. In favorable situations, it requires two sequential RBCs to commit, incurring a latency of 6δ, slightly larger than the 5δ offered by a partially-synchronous protocol (e.g., two-chain HotStuff) or our Ipotane. However, it has a complex process of transitioning to the pessimistic path in unfavorable situations, resulting in an expected latency of 30δ due to 10 sequential RBCs. This is significantly higher than the 10.5δ typical of purely asynchronous protocols (e.g., two-chain VABA) or 18.5δ offered by Ipotane.

VIII. Conclusion

Existing dual-path asynchronous BFT protocols exhibit either low throughput or high latency under unfavorable situations. To address this, we propose a novel protocol named Ipotane, which executes consecutive DBA instances on the pessimistic path. DBA operates as a fusion of biased ABA and VABA, which can be implemented through low-cost modifications to existing VABA protocols. On one hand, DBA promptly detects optimistic path failures, ensuring low latency under unfavorable situations. On the other hand, Ipotane leverages DBA instances to continuously produce blocks without idle periods, thereby achieving high throughput in unfavorable situations. In summary, Ipotane attains performance on par with partially-synchronous protocols under favorable situations and comparable to purely asynchronous protocols in unfavorable situations, as demonstrated by our experiments.

References

[1] I. Abraham, N. Ben-David, and S. Yandamuri, "Efficient and adaptively secure asynchronous binary agreement via binding crusader agreement," in *Proceedings of the 2022 ACM Symposium on Principles of Distributed Computing*. ACM, 2022, pp. 381–391.

[2] I. Abraham, S. Devadas, D. Dolev, K. Nayak, and L. Ren, "Synchronous byzantine agreement with expected o (1) rounds, expected communication, and optimal resilience," in *Proceedings of the 23rd International Conference on Financial Cryptography and Data Security*. Springer, 2019, pp. 320–334.

[3] I. Abraham, D. Malkhi, K. Nayak, L. Ren, and M. Yin, "Sync hotstuff: Simple and practical synchronous state machine replication," in *Proceedings of the 41st IEEE Symposium on Security and Privacy*. IEEE, 2020, pp. 106–118.

[4] I. Abraham, D. Malkhi, and A. Spiegelman, "Asymptotically optimal validated asynchronous byzantine agreement," in *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*. ACM, 2019, pp. 337–346.

[5] P.-L. Aublin, R. Guerraoui, N. Knežević, V. Quéma, and M. Vukolić, "The next 700 bft protocols," *ACM Transactions on Computer Systems*, vol. 32, no. 4, pp. 1–45, 2015.

[6] R. Bacho and J. Loss, "On the adaptive security of the threshold bls signature scheme," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, 2022, pp. 193–207.

[7] I. Bashir, *Mastering Blockchain*. Packt Publishing Ltd, 2017.

[8] M. Ben-Or, "Another advantage of free choice: Completely asynchronous agreement protocols," in *Proceedings of the 2nd Annual ACM Symposium on Principles of Distributed Computing*. ACM, 1983, pp. 27–30.

[9] E. Blum, J. Katz, and J. Loss, "Tardigrade: An atomic broadcast protocol for arbitrary network conditions," in *Proceedings of the 27th International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2021, pp. 547–572.

[10] E. Blum, J. Katz, J. Loss, K. Nayak, and S. Ochsenreither, "Abraxas: Throughput-efficient hybrid asynchronous consensus," in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2023, p. 519–533.

[11] E. Buchman, "Tendermint: Byzantine fault tolerance in the age of blockchains," Ph.D. dissertation, University of Guelph, 2016.

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

[13] C. Cachin, K. Kursawe, F. Petzold, and V. Shoup, "Secure and efficient asynchronous broadcast protocols," in *Proceedings of the 2001 Annual International Cryptology Conference*. Springer, 2001, pp. 524–541.

[14] M. Castro and B. Liskov, "Practical byzantine fault tolerance," in *Proceedings of the 1999 USENIX Symposium on Operating Systems Design and Implementation*. USENIX, 1999, pp. 173–186.

[15] X. Dai, L. Huang, J. Xiao, Z. Zhang, X. Xie, and H. Jin, "Trebiz: Byzantine fault tolerance with byzantine merchants," in *Proceedings of the 38th Annual Computer Security Applications Conference*. ACM, 2022, pp. 923–935.

[16] X. Dai, B. Zhang, H. Jin, and L. Ren, "Parbft: Faster asynchronous bft consensus with a parallel optimistic path," in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2023, p. 504–518.

[17] G. Danezis, L. Kokoris-Kogias, A. Sonnino, and A. Spiegelman, "Narwhal and tusk: A dag-based mempool and efficient bft consensus," in *Proceedings of the 17th European Conference on Computer Systems*. ACM, 2022, pp. 34–50.

[18] D. Dolev, M. J. Fischer, R. Fowler, N. A. Lynch, and H. R. Strong, "An efficient algorithm for byzantine agreement without authentication," *Information and Control*, vol. 52, no. 3, pp. 257–274, 1982.

[19] 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*. ACM, 2018, pp. 2028–2041.

[20] S. Duan, X. Wang, and H. Zhang, "Fin: Practical signature-free asynchronous common subset in constant time," in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, 2023, pp. 815–829.

[21] C. Dwork, N. Lynch, and L. Stockmeyer, "Consensus in the presence of partial synchrony," *Journal of the ACM*, vol. 35, no. 2, pp. 288–323, 1988.

[22] M. Fischer, R. Fowler, and N. Lynch, "A simple and efficient byzantine generals algorithm," in *Proceedings, Second Symposium on Reliability in Distributed Software and Database Systems, Pittsburgh*, 1982.

[23] M. J. Fischer, N. A. Lynch, and M. S. Paterson, "Impossibility of distributed consensus with one faulty process," *Journal of the ACM*, vol. 32, no. 2, pp. 374–382, 1985.

[24] R. Friedman, A. Mostefaoui, and M. Raynal, "Simple and efficient oracle-based consensus protocols for asynchronous byzantine systems," *IEEE Transactions on Dependable and Secure Computing*, vol. 2, no. 1, pp. 46–56, 2005.

[25] Y. Gao, Y. Lu, Z. Lu, Q. Tang, J. Xu, and Z. Zhang, "Dumbo-ng: Fast asynchronous bft consensus with throughput-oblivious latency," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2022, pp. 1187–1201.

[26] R. Gelashvili, L. Kokoris-Kogias, A. Sonnino, A. Spiegelman, and Z. Xiang, "Jolteon and ditto: Network-adaptive efficient consensus with asynchronous fallback," in *Proceedings of the 2022 International Conference on Financial Cryptography and Data Security*. Springer, 2022, pp. 296–315.

[27] N. Giridharan, F. Suri-Payer, I. Abraham, L. Alvisi, and N. Crooks, "Motorway: Seamless high speed bft," *arXiv preprint arXiv:2401.10369*, 2024.

[28] N. Giridharan, F. Suri-Payer, M. Ding, H. Howard, I. Abraham, and N. Crooks, "Beegees: stayin'alive in chained bft," in *Proceedings of the 2023 ACM Symposium on Principles of Distributed Computing*, 2023, pp. 233–243.

[29] V. Gramoli, "From blockchain consensus back to byzantine consensus," *Future Generation Computer Systems*, vol. 107, pp. 760–769, 2020.

[30] G. G. Gueta, I. Abraham, S. Grossman, D. Malkhi, B. Pinkas, M. Reiter, D.-A. Seredinschi, O. Tamir, and A. Tomescu, "Sbft: A scalable and decentralized trust infrastructure," in *Proceedings of the 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. IEEE, 2019, pp. 568–580.

[31] B. Guo, Y. Lu, Z. Lu, Q. Tang, J. Xu, and Z. Zhang, "Speeding dumbo: Pushing asynchronous bft closer to practice," *Cryptology ePrint Archive*, 2022.

[32] B. Guo, Z. Lu, Q. Tang, J. Xu, and Z. Zhang, "Dumbo: Faster asynchronous bft protocols," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2020, pp. 803–818.

[33] T. Hanke, M. Movahedi, and D. Williams, "Dfinity technology overview series, consensus system," *arXiv preprint arXiv:1805.04548*, 2018.

[34] M. M. Jalalzai, J. Niu, C. Feng, and F. Gai, "Fast-hotstuff: A fast and robust bft protocol for blockchains," *IEEE Transactions on Dependable and Secure Computing*, 2023.

[35] I. Keidar, E. Kokoris-Kogias, O. Naor, and A. Spiegelman, "All you need is dag," in *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*. ACM, 2021, pp. 165–175.

[36] R. Kotla, L. Alvisi, M. Dahlin, A. Clement, and E. Wong, "Zyzzyva: Speculative byzantine fault tolerance," in *Proceedings of 21st ACM SIGOPS Symposium on Operating Systems Principles*. ACM, 2007, pp. 45–58.

[37] L. Lamport, R. Shostak, and M. Pease, "The byzantine generals problem," *ACM Transactions on Programming Languages and Systems*, vol. 4, no. 3, pp. 382–401, 1982.

[38] B. Libert, M. Joye, and M. Yung, "Born and raised distributively: Fully distributed non-interactive adaptively-secure threshold signatures with short shares," in *Proceedings of the 2014 ACM symposium on Principles of distributed computing*, 2014, pp. 303–312.

[39] J. Liu, W. Li, G. O. Karame, and N. Asokan, "Scalable byzantine consensus via hardware-assisted secret sharing," *IEEE Transactions on Computers*, vol. 68, no. 1, pp. 139–151, 2018.

[40] Y. Lu, Z. Lu, and Q. Tang, "Bolt-dumbo transformer: Asynchronous consensus as fast as the pipelined bft," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2022, pp. 2159–2173.

[41] Y. Lu, Z. Lu, Q. Tang, and G. Wang, "Dumbo-mvba: Optimal multi-valued validated asynchronous byzantine agreement, revisited," in *Proceedings of the 39th ACM Symposium on Principles of Distributed Computing*. ACM, 2020, pp. 129–138.

[42] A. Miller, Y. Xia, K. Croman, E. Shi, and D. Song, "The honey badger of bft protocols," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2016, pp. 31–42.

[43] A. Mostéfaoui, H. Moumen, and M. Raynal, "Signature-free asynchronous byzantine consensus with t¡ n/3 and o (n2) messages," in *Proceedings of the 2014 ACM Symposium on Principles of Distributed Computing*. ACM, 2014, pp. 2–9.

[44] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," *Decentralized Business Review*, p. 21260, 2008.

[45] J. Niu, F. Gai, M. M. Jalalzai, and C. Feng, "On the performance of pipelined hotstuff," in *IEEE INFOCOM 2021-IEEE Conference on Computer Communications*. IEEE, 2021, pp. 1–10.

[46] M. Pease, R. Shostak, and L. Lamport, "Reaching agreement in the presence of faults," *Journal of the ACM*, vol. 27, no. 2, pp. 228–234, 1980.

[47] M. Pilkington, "Blockchain technology: Principles and applications," in *Research Handbook on Digital Transformations*. Edward Elgar Publishing, 2016.

[48] M. O. Rabin, "Randomized byzantine generals," in *Proceedings of the 24th Annual Symposium on Foundations of Computer Science*. IEEE, 1983, pp. 403–409.

[49] M. A. Schett and G. Danezis, "Embedding a deterministic bft protocol in a block dag," in *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*. ACM, 2021, pp. 177–186.

[50] A. Spiegelman, N. Giridharan, A. Sonnino, and L. Kokoris-Kogias, "Bullshark: Dag bft protocols made practical," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2022, pp. 2705–2718.

[51] S. Toueg, "Randomized byzantine agreements," in *Proceedings of the 3rd Annual ACM Symposium on Principles of Distributed Computing*. ACM, 1984, pp. 163–178.

[52] X. Wang, S. Duan, J. Clavin, and H. Zhang, "Bft in blockchains: From protocols to use cases," *ACM Computing Surveys*, vol. 54, no. 10, pp. 1–37, 2022.

[53] Y. Xiao, N. Zhang, W. Lou, and Y. T. Hou, "A survey of distributed consensus protocols for blockchain networks," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 2, pp. 1432–1465, 2020.

[54] M. Yin, D. Malkhi, M. K. Reiter, G. G. Gueta, and I. Abraham, "Hotstuff: Bft consensus with linearity and responsiveness," in *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*. ACM, 2019, pp. 347–356.

[55] H. Zhang and S. Duan, "Pace: Fully parallelizable bft from reproposable byzantine agreement," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, 2022, pp. 3151–3164.

[56] Z. Zheng, S. Xie, H.-N. Dai, X. Chen, and H. Wang, "Blockchain challenges and opportunities: A survey," *International Journal of Web and Grid Services*, vol. 14, no. 4, pp. 352–375, 2018.

APPENDIX

A. Proof of Lemmas in Section V

LEMMA 1. *Every non-faulty replica will receive either $t+1$ values of 0 or $n-t$ values of 1 during the prepare phase.*

Proof: This is established through two cases.

Case 1: At least one non-faulty replica has the binary input of 0. In this case, each non-faulty replica will receive a message of $(\texttt{PREP}, 0, *)$. According to Lines 6-8 in Algorithm 1, each non-faulty replica will also broadcast a message with 0 if it has not yet broadcast this message. Therefore, each non-faulty replica will eventually receive $t+1$ values of 0 during the prepare phase.

Case 2: Every non-faulty replica has a binary input of 1. Here, each non-faulty replica broadcasts a message of $(\texttt{PREP}, 1, *)$, leading to each receiving $n-t$ values of 1. ■

LEMMA 3. *If a non-faulty replica concludes an epoch in the iteration $iter_h$, all non-faulty replicas will also conclude that epoch in $iter_h$.*

Proof: If a non-faulty replica concludes an epoch in $iter_h$, it implies that DBA_h outputs 1. According to DBA's biased validity property, at least $n-2t$ non-faulty replicas must have inputted 1 to DBA_h. Consequently, as per the rules of Case 2.1, these $n-2t$ non-faulty replicas must have ceased voting for the opt-block B_h. This means no valid QC_h can be generated, and no valid B_{h+1} can be constructed, effectively stalling progress on the optimistic path. On the other hand, a replica will input to DBA_{h+1} only after DBA_h outputs. Based on DBA's agreement property, every non-faulty replica will output 1 from DBA_h and consequently conclude the epoch in $iter_h$ directly. ■

LEMMA 4. *If a non-faulty replica p_i does not conclude the epoch in or before the iteration $iter_h$, p_i will receive either an opt-block B_{h+1} or an output from DBA_h.*

Proof: According to Lemma 3, if a non-faulty replica does not conclude the epoch in or before the iteration $iter_h$, no non-faulty replica concludes that epoch in or before $iter_h$ either. In other words, all non-faulty replicas will advance to $iter_{h+1}$. If any non-faulty replica p_j receives an opt-block B_{h+1} in $iter_h$, all non-faulty replicas, including p_i, will receive B_{h+1} since p_j will broadcast this block. Next, we consider the situation where no non-faulty replica receives B_{h+1}.

According to Case 1 described in Section IV-C, no non-faulty replica will stop participating in the DBA_{h-1} instance. Therefore, each non-faulty replica can output from DBA_{h-1}. Since no non-faulty replica concludes this epoch in or before the iteration $iter_h$, every replica will participate in DBA_h by inputting the binary value 1. Since no non-faulty replica receives B_{h+1}, the opt-block B_{h+2} cannot be generated and non-faulty replicas cannot receive B_{h+2}. In a similar way, no non-faulty replica will stop participating in the DBA_h instance and each one can eventually receive an output from DBA_h. ■

LEMMA 5. *A replica will commit at least two blocks in an epoch, as long as all non-faulty replicas enter this epoch.*

Proof: If a replica does not conclude the epoch in or before the iteration $iter_3$, based on Lemma 4, it will either receive an opt-block B_{k+1} or an output O_k from DBA_k for each k ($1 \leq k \leq 3$). Besides, if it receives an output O_k, we must have $O_k.b = 0$. For $k = 3$, if the replica receives B_3, it will commit B_1 based on the two-chain mechanism; otherwise, namely, if the replica receives O_2, it will also commit B_1 based on the rule of Case 2.1. Thus, the replica will commit the opt-block B_1. Similarly, it will commit the opt-block B_2. In other words, at least two opt-blocks are committed in this case, thus establishing the lemma.

Otherwise, if a non-faulty replica concludes the epoch in or before $iter_3$, according to Lemma 3, every non-faulty replica will conclude the epoch in or before $iter_3$. Before a replica concludes the epoch, it will commit two pess-blocks, which also establishes the lemma. ■

LEMMA 7.1. *Within an epoch, if a non-faulty replica commits an opt-block at height h and another non-faulty replica output b from DBA_{h+1}, then b must be 0.*

Proof: We assume these two non-faulty replicas to be p_i and p_j where p_i commits an opt-block B_h and p_j receives b from DBA_{h+1}. B_h must be committed through the rules of either Case 1 or Case 2.1, as outlined in Section IV-C. If B_h is committed through Case 1, at least $n-t$ replicas, among which $n-2t$ are non-faulty, must have voted for B_{h+1}. This implies that at least $n-2t$ non-faulty replicas would use 0 as the binary input to DBA_{h+1}. Given $n \geq 3t+1$, we have $n-2t \geq t+1$. Therefore, by the biased validity property of DBA, DBA_{h+1} should output a binary value of 0. If B_h is committed through Case 2.1, based on DBA's agreement property, p_j would also receive a binary output of 0 from DBA_{h+1}. ■

LEMMA 7.2. *Within an epoch, if two non-faulty replicas commit two blocks at the same height, then either both two blocks are opt-blocks, or both are pess-blocks.*

Proof: We prove this lemma via contradiction. Without loss of generality, assume two non-faulty replicas p_i and p_j commit opt-block B_h and pess-block C_h, respectively. Based on Lemma 7.1, p_j will receive 0 from DBA_{h+1} if it receives an output at all. According to the protocol described in Section IV-C, p_j must commit a pess-block C_{h-1} or C_{h+1}. We consider the following two situations:

Situation 1: p_j commits C_h and C_{h+1}. According to rules of Case 2.2, p_j must receive a binary output of 1 from DBA_{h+1}, which contradicts the earlier conclusion that DBA_{h+1} would output 0.

Situation 2: p_j commits C_{h-1} and C_h. Per the rules of Case 2.2 described in Section IV-C, p_j must receive a binary output of 1 from DBA_h. With DBA's biased validity property, at least $n-2t$ non-faulty replicas must have inputted 1 to DBA_h. Consequently, according to the rules of Case 2.1, these replicas would stop voting for the opt-block B_h, preventing the generation of a valid QC_h and construction of B_{h+1}. This makes it impossible for p_i to commit B_h through the rules of Case 1. Additionally, p_j will conclude the current epoch in the iteration $iter_h$. By Lemma 3, all non-faulty replicas will conclude the epoch in $iter_h$ without inputting to DBA_{h+1}, making it impossible for p_i to commit B_h through rules of Case 2.1. This contradicts the assumption of p_i committing an opt-block B_h.

Therefore, it is impossible for one non-faulty replica to commit an opt-block and the other to commit a pess-block at the same height, establishing the lemma. ■

LEMMA 7. *Within an epoch, if two non-faulty replicas commit two blocks at the same height, these two blocks must be identical.*

Proof: Per Lemma 7.2, either both two blocks are opt-blocks or both are pess-blocks. If they are pess-blocks, then according to DBA's agreement property, these two blocks must be identical. Now, we consider the situation where both are opt-blocks.

As described in Section IV-C, an opt-block B_h is committed through the rules of Case 1 or Case 2.1. If B_h is committed through the rules of Case 1, a QC for B_h must be generated. If B_h is committed through rules of Case 2.1, DBA_{h+1} must produce an output O_{h+1} where $O_{h+1}.b = 0$ and $O_{h+1}.d = B_h$. By DBA's proof validity property, a replica must have input a valid tuple $\langle 0, \sigma, * \rangle$ to DBA_{h+1}, where σ represents the QC for B_h. In other words, the QC for B_h is also generated. To sum up, when an opt-block is committed, it must be certified by a QC.

Consider the situation where two committed blocks are opt-blocks, denoted as B_h and B'_h, which are certified by QC_h and QC'_h, respectively. By a standard quorum intersection argument, the blocks certified by QC and QC' must also be identical, namely $B_h = B'_h$. ■

LEMMA 8. *If two non-faulty replicas conclude the same epoch, they must commit the same number of blocks within that epoch.*

Proof: Based on Lemma 3, these two replicas must conclude the epoch in the same iteration, which we denote as $iter_l$. According to Algorithm 3, both replicas must receive 1 from DBA_l. Since every non-faulty replica inputs 0 to DBA_1 (as stated in Line 6 of Algorithm 3), the biased-validity property ensures that DBA_1 will output 0. Consequently, the value of l must be equal to or greater than 2. Additionally, both replicas must have received 0 from the preceding DBA instance, which is $iter_{l-1}$.

At any height k, where $1 \leq k \leq l-2$, both replicas commit an opt-block B_k. At heights $l-1$ and l, they commit a pess-block, C_{l-1} or C_l, respectively. Therefore, these two replicas commit the same number of blocks within that epoch. ■

B. Additional Related Work

In this section, we summarize the additional related works except in Section VII, specifically including the synchronous BFT consensus and partially-synchronous BFT consensus.

1) Synchronous BFT consensus: Synchronous BFT consensus protocols are designed under the network assumption that each message can be delivered within a predefined period, denoted as Δ, after its transmission. Representatives in this category encompass many early works [46], [37], [18], [22] as well as some recent studies [33], [3]. However, protocols designed for synchronous networks encounter a challenge in setting the right value for Δ. If Δ is set too small, the synchronous assumption becomes fragile. Conversely, if Δ is set too large, the resulting protocol will be slow, as its performance must directly depend on Δ [3].

2) Partially-synchronous BFT consensus: Given the FLP impossibility [23], which states that deterministic fault-tolerant asynchronous consensus is impossible, Dwork et al. propose an intermediate network assumption called partial synchrony [21]. The partial synchrony model assumes the network to be synchronous after an unknown *Global Stabilization Time* (GST), which has been the mainstream model for practical systems for a long time.

One of the most notable works adopting the partially-synchronous assumption is PBFT [14]. Building on PBFT, subsequent works aim to reduce consensus latency by introducing a fast committing path [36], [39], [30], [15]. Drawing inspiration from the flourishing blockchain technology [44], structures like blocks and chains are incorporated into BFT consensus to pipeline consecutive consensus instances, thereby enhancing throughput. Example chained BFT consensus include Tendermint [11], Casper [12], and HotStuff [54]. Some works [45], [28] address liveness issues in chained-BFT where faulty leaders can prevent progress. Motorway constructs a data dissemination layer to improve throughput during periods of bad networks [27].

Despite its popularity, partially-synchronous protocols have raised concerns about their robustness [42], [19]. An adversary with network manipulation capacities can compromise the liveness of a partially-synchronous protocol. Consequently, a recent line of work revisits the asynchronous network [32], [55], [20].

Ling: It would best if we can squeeze Appendix A and C back to the main body.**Xiaohai:** Have done this. Once the colored comments are removed, the main body will approximately meet the page limit of 13 pages.