ParBFT: Faster Asynchronous BFT Consensus with a Parallel Optimistic Path

Xiaohai Dai
Huazhong Univ. of Science and Tech.
Univ. of Illinois Urbana-Champaign

Bolin Zhang
Zhejiang University
Univ. of Illinois Urbana-Champaign

Hai Jin
Huazhong Univ. of Science and Tech.

Ling Ren
Univ. of Illinois Urbana-Champaign

ABSTRACT

To reduce latency and communication overhead of asynchronous *Byzantine Fault Tolerance* (BFT) consensus, an optimistic path is often added, with Ditto and BDT as state-of-the-art representatives. These protocols first attempt to run an optimistic path that is typically adapted from partially-synchronous BFT and promises good performance in good situations. If the optimistic path fails to make progress, these protocols switch to a pessimistic path after a timeout, to guarantee liveness in an asynchronous network. This design crucially relies on an accurate estimation of the network delay Δ to set the timeout parameter correctly. A wrong estimation of Δ can lead to either premature or delayed switching to the pessimistic path, hurting the protocol's efficiency in both cases.

To address the above issue, we propose ParBFT, which employs a parallel optimistic path. As long as the leader of the optimistic path is non-faulty, ParBFT ensures low latency without requiring an accurate estimation of the network delay. We propose two variants of ParBFT, namely ParBFT1 and ParBFT2, with a trade-off between latency and communication. ParBFT1 simultaneously launches the two paths, achieves lower latency under a faulty leader, but has a quadratic message complexity even in good situations. ParBFT2 reduces the message complexity in good situations by delaying the pessimistic path, at the cost of a higher latency under a faulty leader. Experimental results demonstrate that ParBFT outperforms Ditto or BDT. In particular, when the network condition is bad, ParBFT can reach consensus through the optimistic path, while Ditto and BDT suffer from path switching and have to make progress using the pessimistic path.

CCS CONCEPTS

• **Blockchain and Distributed Systems** → Consensus protocols;

KEYWORDS

Byzantine fault tolerance, Byzantine generals, consensus, blockchain

ACM Reference format:
Xiaohai Dai, Bolin Zhang, Hai Jin, and Ling Ren. 2023. ParBFT: Faster Asynchronous BFT Consensus with a Parallel Optimistic Path. In *Proceedings of Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security, Copenhagen, Denmark, November 26–30, 2023 (CCS '23),* 17 pages.
https://doi.org/10.1145/3576915.3623101

Xiaohai Dai and Hai Jin are with the School of Computer Science and Technology, Huazhong University of Science and Technology, National Engineering Research Center for Big Data Technology and System, Services Computing Technology and System Lab, Cluster and Grid Computing Lab, Wuhan, 430074, China.

1 INTRODUCTION

Over the past decade, the increasing popularity of blockchain [37, 54, 65] has brought considerate attention back to the *Byzantine Fault Tolerance* (BFT) consensus protocols [33, 64, 66]. In general, a BFT consensus protocol ensures multiple replicas reach agreement, even if a fraction of them may behave arbitrarily (called Byzantine replicas) [43]. BFT consensus protocols can be roughly divided into three categories based on their timing assumptions: synchronous ones, partially synchronous ones, and asynchronous ones. Among the three categories, asynchronous protocols offer the strongest robustness to unpredictable network conditions [26, 36, 49]. However, asynchronous BFT protocols are rarely deployed in production for performance reasons [45]. More specifically, compared to their synchronous and partially synchronous counterparts, asynchronous BFT protocols have higher latency (larger number of rounds) and higher communication overheads, even when all replicas are non-faulty and the network condition is good.

To remedy the inferior performance of asynchronous BFT, a number of works introduce an optimistic path [42, 56], with Ditto [32] and BDT [45] as recent representatives. At a high level, these protocols typically have two paths: an optimistic partially synchronous path driven by a leader and a pessimistic path that works in asynchrony. The system first attempts to run the optimistic path, which has low latency and smaller communication overhead. If the optimistic path fails to make progress, the protocol falls back to the pessimistic path after a timeout event. After one or more agreement instances on the pessimistic path, the protocol will switch back to the optimistic path. Since only one path is being executed at any given time, we call this design the **serial-path** paradigm.

The serial-path paradigm has several drawbacks. First, it requires a good estimation of network latency, usually denoted Δ, to set the timer accordingly. It is quite challenging to get the parameter Δ right. When the leader is Byzantine, the optimistic path cannot make any progress, and the fallback to the pessimistic path should ideally be launched as soon as possible. A large value of Δ will delay the fallback and hurt latency. On the contrary, if Δ is mistakenly set too small, the timeout and fallback events will be triggered prematurely, potentially disrupting a non-faulty leader on the optimistic path who is about to make progress.

Moreover, when to switch back to the optimistic path is also a tough decision. If the switch is performed too late since the network has healed, the protocol has unnecessarily stayed on the pessimistic path for too long. Conversely, switching back too hastily while the network condition remains poor is meaningless and wasteful as the optimistic path still cannot make progress. This may even cause

Table 1: Consensus performance comparison. As for the serial-path protocols (i.e., Ditto and BDT), the performance is measured with the protocol starting from the optimistic path, which is the default in these protocols. The number of total replicas is denoted as n, and the actual number of faulty replicas is denoted as t.

	Δ is needed	Latency			Message complexity		
		Non-faulty leader		Faulty leader	Non-faulty leader		Faulty leader
		$\delta \leq \Delta$	$\delta > \Delta$		$\delta \leq \Delta$ ¶	$\delta > \Delta$	
Ditto [32] *	Yes	5δ	$2\Delta + 16\delta$	$2\Delta + 16\delta$	$O(tn)$	$O(n^2)$	$O(n^2)$
BDT [45] †	Yes	5δ	$2\Delta + 25\delta$	$2\Delta + 25\delta$	$O(tn)$	$O(n^2)$	$O(n^2)$
ParBFT1 ‡	No	5δ	5δ	22δ	$O(n^2)$	$O(n^2)$	$O(n^2)$
ParBFT2 §	Yes	5δ	5δ	$2\Delta + 25\delta$	$O(tn)$	$O(n^2)$	$O(n^2)$

† The optimistic path in BDT is implemented by sequential multicast (Bolt-sCAST [45]), while the pessimistic path is implemented by sMVBA [35], which offers the lowest latency in the asynchronous network.
‡ Although Δ is needless in ParBFT1, we list the metrics in both cases (i.e., $\delta \leq \Delta$ and $\delta > \Delta$) to make the table more readable.
* Although Ditto needs lower latency in bad situations, it suffers from a liveness problem [31].
* †§ When the optimistic path is implemented through chain structure, the timeout parameter is set as an estimated *Round Trip Time*, which equals 2Δ.
†§ The ABA protocol adopted by BDT and ParBFT is adapted from [1], whose expected worst-case latency is 9δ.
¶ It is worth noting that Ditto, BDT, and ParBFT2 have a message complexity of $O(tn)$ instead of $O(n)$ due to the need for replicas to respond to data retrieval requests, which can be initiated by t faulty replicas.

frequent back-and-forth switches, making the protocol even slower than simply running the pessimistic path alone. For some contexts, Ditto [32] opts for the hasty approach and performs the switch back whenever a single agreement instance on the pessimistic path is finished. BDT [45] similarly uses a hasty switch in their pseudocode. Although BDT mentions that other heuristics can be used for the switch back, designing these heuristics is also a tricky task.

To address these challenges regarding path switches, we propose an alternative paradigm for adding optimistic paths to asynchronous BFT: **running the two paths in parallel**. At a high level, by running the two paths in parallel, replicas can reach a decision as soon as one of the two paths succeeds. This enables the protocol to gracefully handle both good and bad network conditions and avoid the drawbacks of the serial-path paradigm. To be more concrete, we propose ParBFT that runs a partially synchronous optimistic path and an asynchronous pessimistic path in parallel. The two paths may each produce an output (called candidates). ParBFT then leverages an *Asynchronous Binary Agreement* (ABA) algorithm to reach an agreement between these two candidates. The last key design element of ParBFT is a *shortcut* mechanism: if the leader is non-faulty and the network is good, all replicas will decide at the end of the optimistic path and directly advance to the next instance, without the need to execute the ABA algorithm or even the pessimistic path. This makes ParBFT's performance in the good situation similar to the serial-path paradigm.

We present two variants of ParBFT, which we call ParBFT1 and ParBFT2, that give a trade-off between latency and communication. ParBFT1 launches the two paths simultaneously; this variant offers better latency under a Byzantine leader but suffers from quadratic message complexity even in a good situation. On the contrary, ParBFT2 delays the launch of the pessimistic path, and as a result, reduces the message complexity to linear in a good situation at the cost of higher latency under a Byzantine leader.

As shown in Table 1, prior works Ditto [32] and BDT [45] achieve a low latency of 5δ (δ represents the actual network delay) only when the leader is non-faulty *and* the parameter Δ is estimated correctly (i.e., $\delta \leq \Delta$). In contrast, ParBFT1 and ParBFT2 achieve a good latency of 5δ as long as the leader of the optimistic path is non-faulty, regardless of whether Δ is estimated correctly or not. As mentioned, ParBFT1 makes a sacrifice on the message complexity in the good situation: when the leader is non-faulty and the estimation of Δ is correct, ParBFT1 incurs quadratic communication. ParBFT2 avoids this problem by delaying the launch of the pessimistic path by 5Δ time: this reduces the communication complexity in the good case back to $O(tn)$[1] (t and n represent the number of actual faulty replicas and total replicas, respectively) but increases the latency under a Byzantine leader by that amount.

We also note that while ParBFT1 does not need the parameter Δ at all, ParBFT2 brings back the parameter of Δ. But unlikely prior works, the penalty for an incorrect estimation of Δ is much smaller. Concretely, when Δ is set too small, i.e., $\Delta < \delta$, ParBFT2 only incurs an increase in the communication cost, while prior works incur much longer latency, increased communication cost, and the potential problem of back-and-forth switching.

We implement both variants of ParBFT and conduct extensive experiments to evaluate their performance in comparison with prior works. Our implementations use the chain-based paradigm in which different agreement instances are pipelined to improve the throughput. The experiments are divided into three parts, corresponding to three different scenarios. The first part mimics a good situation where the leader is non-faulty and the network is good. In the second part, we simulate a slow network by intentionally delaying messages, while assuming a non-faulty leader. Finally, in the third part, we introduce a faulty leader by delaying proposals from the leader.

The experimental results demonstrate that, under good situations, ParBFT2 performs comparably well to Ditto and BDT, as all three protocols can commit through the optimistic path. As expected, as the number of replicas increases beyond twenty-two, the performance of ParBFT1 deteriorates due to its quadratic message complexity. In the situation of a slow network, where the delay is

[1] A number of prior works [32, 45, 67] claim $O(n)$ communication in the good case. But upon closer inspection, they ignored the cost of retrieving the committed data. In more detail, a replica that commits on the linear optimistic path has to respond to retrieval requests from other replicas who have not, or claim to have not, received the committed data. This adds a factor of t to the communication overhead, since each faulty replica can send such a retrieval request to all non-faulty replicas. See [58, 63] for a more thorough discussion on this issue.

set larger than Δ, both ParBFT1 and ParBFT2 exhibit significantly lower latency compared to Ditto and BDT. ParBFT achieves lower latency because it can commit through the optimistic path even if the network delay is wrongly estimated, whereas Ditto or BDT must follow the pessimistic path. In the case of a faulty leader, all protocols will commit through the pessimistic path. However, ParBFT1 offers lower latency than either Ditto, BDT, or ParBFT2, as it launches the pessimistic path immediately without waiting for a timeout event.

To sum up, we make the following contributions in this paper. We first identify major limitations of current serial-path asynchronous protocols: they rely on accurate estimates of network latency to appropriately switch between the two paths. We then propose a new paradigm called ParBFT that runs the two paths in parallel to address these limitations. Two variants of ParBFT are presented, offering a trade-off between latency and communication overhead. Finally, we implement our protocols and conduct comprehensive experiments to demonstrate their advantages.

The remainder of this paper is structured as follows. In Section 2, we introduce the model used in our work and present some preliminaries that will serve as building blocks to our protocols. Section 3 outlines the main idea of parallel paths by describing a preliminary version named ParBFT0. In Section 4 and Section 5, we elaborate on the two practical variants of ParBFT that provide a trade-off between latency and communication overhead. To improve throughput, we also devise chain-based versions of ParBFT, which are described in Section 6 along with a detailed evaluation. We discuss related work in Section 7 and conclude the paper in Section 8.

2 MODELS AND PRELIMINARIES

2.1 Models and definitions

We consider a distributed system consisting of $n = 3f + 1$ replicas, among which up to f can misbehave in an arbitrary manner, i.e., they can be Byzantine. Each replica has a unique identity denoted as p_i ($0 \le i < n$). All the Byzantine replicas are under the control of an adversary who can coordinate their actions. Each pair of replicas is connected through a reliable link, which will eventually deliver every message, but the network is asynchronous, meaning that any message can be delayed by the adversary arbitrarily. Leaders of the optimistic path are selected by a predetermined order, e.g., simple round-robin.

We assume a *public-key infrastructure* (PKI), which allows each replica p_i to be identified by a public key pk_i, and all the public keys are known to all replicas. Corresponding to pk_i, each replica holds its private key sk_i. We also assume a threshold cryptosystem is established among the replicas, possibly via *Distributed Key Generation* (DKG) protocols [3, 23, 40], to enable threshold signatures. We also assume a collision-resistant hash function is available. Finally, we assume that the adversary has limited computational resources and cannot break the PKI, the threshold cryptosystem, or the hash function.

For performance evaluation, we consider two types of situations: good situations and bad situations. A good situation is when the leader of the optimistic path is non-faulty and (if applicable) the actual network delay δ is no greater than the estimated parameter Δ. On the contrary, a bad situation is when the designated leader is faulty or δ is larger than Δ. It is worth noting that since there is no parameter of Δ in ParBFT0 or ParBFT1, the good and bad situations depend solely on whether the designated leader is non-faulty.

A consensus protocol maintains a replicated log among all non-faulty replicas. Each entry in the log corresponds to a request or some submitted data from a client. Henceforth, we use the terms "request", "data", and "log entry" interchangeably. Besides, the client is assumed to send the data to the leader on the optimistic path initially. If, within a predetermined period, the data cannot be successfully committed, the client will then broadcast the data to all replicas. A correct consensus protocol must guarantee safety and liveness, which are defined as follows:

- **Safety:** If two non-faulty replicas commit two data d and d' at the same log position, then d must be equal to d'.
- **Liveness:** If a client proposes a request req, req will eventually be committed.

2.2 Preliminaries

In the design of ParBFT, we make use of *Validated Asynchronous Byzantine Agreement* (VABA) protocols to implement the pessimistic path and *Asynchronous Binary Agreement* (ABA) protocol to decide between the outputs from the two paths. We utilize ABA in a black-box manner and slightly modify VABA to enable it to output a proof for the decided value. We refer to the modified VABA as *Provable VABA* (PVABA). In this section, we present the interfaces of ABA and PVABA and show how to modify a VABA protocol to a PVABA protocol.

2.2.1 ABA interface. An ABA protocol is used to reach consensus on a single bit [55, 62]. In an ABA protocol, each replica inputs a bit value of 0 or 1, and ultimately, each non-faulty replica will decide on the same bit value as the output. To be more precise, an ABA protocol must satisfy the following three properties:

- **Validity:** If a non-faulty replica decides on a value v, v must be input by at least one non-faulty replica.
- **Agreement:** If two non-faulty replicas decide on two values v and v' respectively, then $v = v'$.
- **Termination:** If all non-faulty replicas complete inputting values to the protocol, every non-faulty replica will eventually decide on a value.
- **Integrity:** No non-faulty replica decides twice.

Over the past few decades, various ABA protocols have been proposed [1, 8, 30, 51]. We will use ABA in a black box.

2.2.2 VABA & PVABA interfaces. First, we describe the original VABA interface. In a VABA protocol, each replica is allowed to input an arbitrary value, and the protocol will eventually decide on a value [15]. To prevent the protocol from deciding on an invalid or trivial value, an external validation predicate Q is defined, and the output value must satisfy Q. More formally, a correct VABA protocol must satisfy the properties as follows:

- **External-validity:** If a non-faulty replica decides on a value v, $Q(v)$ must be True.
- **Agreement:** If two non-faulty replicas decide on two values v and v' respectively, then $v = v'$.

- **Termination:** If all non-faulty replicas complete inputting values to the protocol, every non-faulty replica will eventually decide on a value.
- **Quality:** The probability of deciding on a non-faulty replica's input is at least 1/2.
- **Integrity:** No non-faulty replica decides twice.

Decided values of a VABA protocol will be taken as inputs for the final agreement of ParBFT. To prevent Byzantine replicas from forging decided values, we further require the VABA protocol to output a proof for the decided value. Therefore, the output from the VABA protocol has the format of (v, σ), where σ is the proof for the value v. Each replica can verify the legitimacy of the VABA output through an external validity predicate $R(v, \sigma)$. We note the differences between the two predicates: Q is to verify the external validity of an input, while R is to verify that a value is indeed decided by the VABA instance. The adapted VABA interface is named PVABA, which has an additional property than VABA:

- **Provability:** If a non-faulty replica output (v, σ), then $R(v, \sigma) = true$. If a Byzantine replica outputs (v, σ) satisfying $R(v, \sigma) = true$, then some non-faulty replica must have output (v, σ).

Existing VABA protocols [5, 15, 35, 46] can be easily modified to satisfy provability. Taking AMS-VABA [5] or sMVBA [35] as examples, the proof σ can be set as the VIEW-CHANGE message (as shown in Line 22 of Algorithm 3 in [5]) or the HALT message (as shown in Line 16 of Algorithm 5 in [35]), and the predicate $R(v, \sigma)$ can be set as the threshold signature verification function. When there is no ambiguity, we will simply use VABA to mean PVABA in the remaining parts of this paper.

3 PARBFT DESIGN

Before delving into the final designs of ParBFT (i.e., ParBFT1 and ParBFT2), we first introduce a preliminary variant named ParBFT0 in this section. ParBFT0 is meant to illustrate the basic idea of running two parallel paths and is not designed for efficiency. As such, ParBFT0 has higher latency and larger communication overhead even in a good situation. But it demonstrates the feasibility of removing the parameter Δ and the finicky path-switch mechanism.

3.1 Description of ParBFT0

The structure of ParBFT0 is illustrated in Figure 1. The protocol consists of two stages: parallel paths and final agreement. In the first stage, an optimistic path and a pessimistic path are launched simultaneously, and each replica participates in both paths. The optimistic path can be implemented using the normal-case protocol of many partially synchronous BFT works. To be concrete, we adopt the normal-case protocol of SBFT [34], as it offers a low communication overhead of $O(tn)$. The pessimistic path can be constructed using any VABA protocol in a black box.

We borrow the notion of *Provable Broadcast* (PB) from AMS-VABA [5] or sMVBA [35] to describe the process of data broadcast plus vote collection. In a PB instance, a broadcaster p_b first broadcasts its data d along with a proof π in the format of (d, π) to each replica. The proof π is used to verify the validity of d according to a global predicate function. If the validation passes, a replica p_i will output a tuple (d, π) locally and send its vote through a threshold signature share ρ on d to p_b. To aid presentation, we refer to the

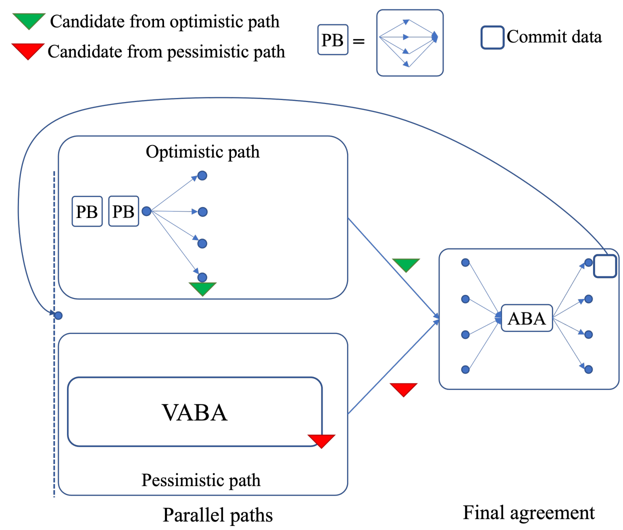

Figure 1: The structure of ParBFT0.

replicas that send votes to the broadcaster in a PB instance as *voters*. After collecting more than two-thirds of the shares, p_b can combine them into a final threshold signature σ and output the tuple (d, σ).

As Figure 1 illustrates, the optimistic path consists of two consecutive PB instances followed by an additional broadcast performed by the leader (p_L). For brevity, we refer to the two consecutive PBs as one *Strong Provable Broadcast* (SPB) as defined in sMVBA [35]. In an SPB instance, the broadcaster p_b uses the output from the first PB (PB1) as input for the second PB (PB2). In other words, $\pi_2 = \sigma_1$ where σ_1 represents p_b's output from PB1 and π_2 denotes the proof for d in PB2. The broadcaster p_b's output from SPB is exactly the output from PB2. Moreover, in the additional broadcast after SPB, p_b broadcasts its output from SPB, namely the tuple (d, σ_2).

A replica returns from the optimistic path after receiving the tuple of (d, σ_2), marked by the green triangle in Figure 1. Recall that in Section 2.2.2, a replica returning from the pessimistic path (i.e., VABA) also possesses a tuple of (d, σ), which is marked by the red triangle in Figure 1. The tuples returned from the two parallel paths are referred to as candidates. We distinguish them as optimistic candidates and pessimistic candidates, denoted by (d_o, σ_o) and (d_p, σ_p), respectively. It is worth noting that (d_o, σ_o) obtained by different replicas are identical, and the same holds true for (d_p, σ_p).

In the second stage of ParBFT0, each replica takes the first candidate it obtains from the parallel paths as input for the final agreement. The final agreement, described in Algorithm 1, is primarily implemented based on a black-box ABA protocol, where 0 represents the optimistic candidate (d_o, σ_o) and 1 represents the pessimistic candidate (d_p, σ_p). A replica will first broadcast its candidate (Line 2) and then invoke the ABA protocol with the mapped bit (Lines 3-8). Once the ABA protocol outputs a decision bit, the replica waits until the candidate corresponding to the decision bit is received (Lines 9-13) and then outputs the candidate (Lines 14-16).

To reduce the number of communication rounds, the round of broadcasting the candidate (Line 2 of Algorithm 1) can be merged with the first round of ABA. Additionally, a replica only accepts the candidate broadcast by others if it passes the check against a global

Algorithm 1 FINAGR0: Final agreement protocol in ParBFT0 (for replica p_i)

Let v_i denote the input (a candidate in the context of ParBFT0) of p_i and $ValFn$ denote a global predicate function.

1: **initialize** $vals[2] \leftarrow [\perp, \perp]$
2: **broadcast** $(\texttt{FA}, v_i)$
3: **if** v_i is an optimistic candidate **then:**
4:     $vals[0] \leftarrow v_i$
5:     **invoke** ABA with 0
6: **else:**
7:     $vals[1] \leftarrow v_i$
8:     **invoke** ABA with 1

9: **upon** receiving $(\texttt{FA}, v_j)$ from p_j that $ValFn(v_j) = true$ **do:**
10:     **if** v_j is an optimistic candidate **and** $vals[0] = \perp$ **then:**
11:         $vals[0] \leftarrow v_j$
12:     **else if** v_j is a pessimistic candidate **and** $vals[1] = \perp$ **then:**
13:         $vals[1] \leftarrow v_j$

14: **upon** receiving the output b from ABA **do:**
15:     **wait** until $vals[b] \neq \perp$
16:     **output** $vals[b]$

predicate function $ValFn$ (Line 9 of Algorithm 1). If the candidate is optimistic, $ValFn$ is simply the verification function of the threshold signature. If the candidate is pessimistic, $ValFn$ is precisely the predicate $R(v, \sigma)$ mentioned in Section 2.2.2.

Note that a replica that returns from either path can immediately stop participating in the other path. Besides, it is possible for a replica to receive valid candidate (d, σ) from the final agreement protocol before it returns from either path in the first stage. In such a case, the replica can treat (d, σ) as its own candidate (as though it has obtained (d, σ) from the first stage on its own), input (d, σ) to the final agreement, and terminate both paths in the first stage.

3.2 Correctness analysis of ParBFT0

The correctness analysis of ParBFT0 includes two parts: safety and liveness. Notably, each instance of the ParBFT0 protocol described above is responsible for committing data at one log position. Therefore, for safety, we only need to show that all non-faulty replicas commit the same data from a given ParBFT0 instance. For liveness, since each leader attempts to propose requests from clients, we only need to show that each non-faulty replica is able to commit from the ParBFT0 instance.

3.2.1 Safety. The safety analysis of ParBFT0 is straightforward and relies on the safety guarantees provided by the SBFT, VABA, and ABA protocols. According to the safety property of SBFT, all optimistic candidates are identical, and according to the agreement property of VABA, all pessimistic candidates are also identical. This means that there can only be two distinct candidates taken as inputs into the final agreement protocol, which are mapped to bits 0 and 1. The ABA protocol ensures that all non-faulty replicas will output the same bit. Thus, all non-faulty replicas will output the same candidate from the final agreement protocol corresponding to the ABA's output bit. This guarantees the safety of ParBFT0.

3.2.2 Liveness. We refer to the execution of ParBFT to commit a single decision as one instance. Within each instance, a client can initially send the request to the leader of the optimistic path. If the request does not get committed through the optimistic path for some time, the client broadcasts the request to all replicas. Recall that the leader of the optimistic path is predetermined in a round-robin fashion. If the optimistic path under some non-faulty leader succeeds, the client's request will be committed. On the flip side, if all instances with non-faulty leaders commit in the pessimistic path, the quality property of VABA ensures with at least 1/2 probability that a non-faulty replica's input will be committed, which will include the client's request. It remains to show that each consensus instance will successfully commit. We will first establish a lemma.

LEMMA 1. *Every non-faulty replica in ParBFT0 will eventually invoke the ABA protocol.*

PROOF. We establish this lemma through two cases.

Case 1: Some non-faulty replica p_i outputs from the optimistic path. According to Algorithm 1, p_i will broadcast its optimistic candidate during the stage of final agreement. Therefore, non-faulty replicas that have not yet output from either the optimistic or the pessimistic path can receive an optimistic candidate from p_i. This ensures that every non-faulty replica will acquire a candidate and invoke the ABA protocol.

Case 2: No non-faulty replica outputs from the optimistic path. In this case, every non-faulty replica will keep running the pessimistic path. The termination property of VABA guarantees that each non-faulty replica will eventually output from the pessimistic path and acquire a pessimistic candidate. Thus, each non-faulty replica invokes the ABA protocol. □

THEOREM 2. *Every non-faulty replica in ParBFT0 can successfully commit in each consensus instance.*

PROOF. Due to Lemma 1, every non-faulty replica will invoke the ABA protocol. Subsequently, by the termination property of ABA, every non-faulty replica will eventually output from the ABA protocol. Based on the validity property of ABA, at least one non-faulty replica must have inputted the same bit as the output bit. That replica must have also broadcast the corresponding candidate. Therefore, each non-faulty replica will receive a candidate corresponding to the output bit and commit that candidate value. This concludes the proof of Theorem 2. □

3.3 Performance analysis of ParBFT0

We analyze the performance of ParBFT0 in terms of consensus latency and communication overhead. To this end, we assume that ABA and VABA are implemented based on the state-of-the-art ABY-ABA [1] and the sMVBA [35], respectively. The expected latency of ABY-ABA is 4δ in a good situation and 9δ in a bad situation. The expected latency of sMVBA is 6δ in a good situation and 12δ in a bad situation.

If the leader is non-faulty, each replica will return from the optimistic path first, which takes 5δ. In addition, the ABA protocol has an expected latency of 4δ. Therefore, in the case of a non-faulty leader, the expected latency of ParBFT0 is 9δ. When the leader is

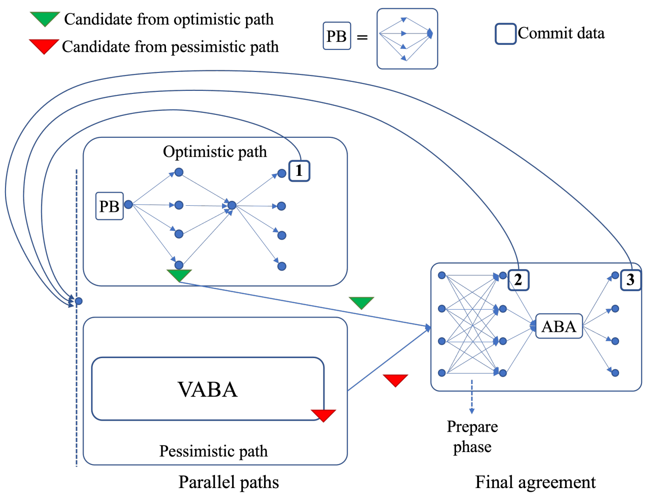

Figure 2: The structure of ParBFT1.

faulty, each replica will return from the pessimistic path first. Consequently, the expected consensus latency of ParBFT0 is 21δ: 12δ from sMVBA and 9δ from ABA. Regarding communication overhead, since each replica broadcasts data on the pessimistic path, ParBFT0 always has a message complexity of $O(n^2)$.

4 PARBFT1 WITH LOWER LATENCY

To reduce latency under a non-faulty leader, we propose ParBFT1, which allows a replica to commit directly on the optimistic path without going through the final agreement. This is achieved by adding a shortcut on the optimistic path and a *prepare* phase to exchange candidates before running ABA. We also modify the rule of returning candidates from the optimistic path.

4.1 Description of ParBFT1

Figure 2 illustrates the structure of ParBFT1, where we open the box of PB2 to show how a replica outputs a candidate in PB2. Comparing it with ParBFT0 in Figure 1 highlights the difference of ParBFT1 from ParBFT0: a replica outputs a candidate from the optimistic path after receiving (d_o, σ_1) in PB2, without waiting for (d_o, σ_2) as in ParBFT0. Instead, upon receiving (d_o, σ_2), a replica can immediately commit and exit the current ParBFT1 instance, marked by 1 in Figure 2. This serves as a shortcut on the optimistic path, eliminating the need to execute the final agreement and resulting in an optimal latency of 5δ, which is the same as Ditto or BDT. Algorithm 2 outlines the pseudocode of the optimistic path in ParBFT1. For brevity, we omit the validity check of data in the pseudocode. As shown in Lines 10-11, a replica outputs the optimistic candidate after receiving data from PB2. To ensure liveness, a replica will broadcast a Halt message before exiting. Any replica that receives a valid Halt message can take a shortcut to commit and exit the current ParBFT1 instance as well. Pseudocode related to the decision and broadcast of Halt messages is shown in Lines 12-15 of Algorithm 2.

The use of a shortcut rule may pose safety risks to the algorithm, as some replicas may commit through the shortcut while others may commit different data through the final agreement. To mitigate this safety risk, we introduce a *prepare* phase to exchange candidates

Algorithm 2 OPTPATH1: Optimistic path protocol in ParBFT1 (for replica p_i, with p_L as the leader)

Let v_i represent the data proposed by p_i.

1: **if** $p_i = p_L$ **then:**
2: $d_o \leftarrow v_i$
3: **activate** PB1 as the broadcaster with $(d_o, \perp)$ as data
4: **upon** receiving (d_o, σ_1) from PB1 **do:**
5: **activate** PB2 as the broadcaster with (d_o, σ_1) as data
6: **upon** receiving (d_o, σ_2) from PB2 **do:**
7: **broadcast** $(\texttt{OPTH}, d_o, \sigma_2)$
8: **else:**
9: **activate** PB1 and PB2 as a voter

10: **upon** receiving (d_o, σ_1) from PB2 **do:**
11: **output** the candidate (d_o, σ_1)

12: **upon** receiving $(\texttt{OPTH}, d_o, \sigma_2)$ from p_L **do:**
13: **commit** d_o
14: **broadcast** $(\texttt{HALT}, d_o, \sigma_2)$ **if** has not
15: **exit**

before activating the ABA protocol. The *prepare* phase also provides an additional shortcut for committing data without running an ABA protocol. The final agreement after adding the *prepare* phase is described by Algorithm 3. Each replica will begin by broadcasting a PREP message, which contains the candidate and a partial threshold signature on the data (Lines 3-4 of Algorithm 3). The threshold is set to $n - f$. Once a replica has received $n - f$ valid PREP messages, it checks whether it can commit using another shortcut, marked by 2 in Figure 2. If it cannot, the replica will prepare the input value to the ABA protocol. In more detail, there are three cases:

Case 1: If all the $n - f$ PREP messages contain optimistic candidates (Lines 6-10 of Algorithm 3), the replica can construct a complete threshold signature σ for d_o based on the partial signatures in the PREP messages. With a valid σ, the replica can commit d_o directly without activating the ABA protocol. Also, the replica will broadcast a Halt message containing (d_o, σ) to help other replicas commit d_o.

Case 2: If all the $n - f$ PREP messages contain pessimistic candidates (Lines 16-20 of Algorithm 3), the replica will broadcast the pessimistic candidate (d_p, σ_p) and invoke the ABA protocol with 1.

Case 3: If both optimistic and pessimistic candidates are present in these $n - f$ PREP messages (Lines 11-15 of Algorithm 3), the replica will broadcast the optimistic candidate and invoke the ABA protocol with 0.

Pseudocode of ParBFT1 is given in Algorithm 4. Note that even if a replica has obtained a candidate from the optimistic path, it will continue the remaining parts of the optimistic path. However, like in ParBFT0, a replica that obtains a candidate from either path will terminate its participation in the other path (Lines 8-11 of Algorithm 4). To speed up the progress, a replica can use the candidate from the received PREP message as if it is obtained from the first stage. In other words, the replica can construct and broadcast its PREP message using the candidate received from others. Besides, in Lines 3-6 of Algorithm 4, once a replica receives a valid Halt message, it can commit immediately and exit the current ParBFT1 instance. If

Algorithm 3 FINAGR: Final agreement protocol in ParBFT1 and ParBFT2 (for replica p_i)

Let v_i represent an input value (a candidate in the context of ParBFT1 or ParBFT2) of p_i. *SignShare* and *Combine* denote the threshold signature functions.

initialize $vals[2] \leftarrow [\perp, \perp]$
parse v_i as (tag, d, σ)
$\rho \leftarrow SignShare_{n-f}(d, tag)$
broadcast (PREP, tag, d, σ, ρ)

upon receiving $n - f$ PREP messages **do:**
  if all the $n - f$ messages with tag OPT **then:**
    $S_\rho \leftarrow$ all the ρ from $n - f$ messages
    extract d_o from one message
    broadcast $\left(\text{HALT}, d_o, Combine_{n-f}(S_\rho, d_o, \text{OPT})\right)$
    commit d_o; **exit**
  else if at least one message with tag OPT **then:**
    extract d_o and σ_o from the message with tag OPT
    broadcast (FA, d_o, σ_o)
    invoke ABA with 0
    $vals[0] \leftarrow (d_o, \sigma_o)$
  else:
    extract d_p and σ_p from one message
    broadcast $(\text{FA}, d_p, \sigma_p)$
    invoke ABA with 1
    $vals[1] \leftarrow (d_p, \sigma_p)$

// Same as Lines 9-16 of Algorithm 1 (FINAGR0)

data is committed at the end of the final agreement (Lines 13-15 of Algorithm 4), a replica is not necessary to broadcast a Halt message. This is because the ABA protocol in the final agreement already includes a broadcast step that assists others in obtaining the output from ABA and committing the data [1].

4.2 Correctness analysis

4.2.1 Safety. There are three points at which data can be committed in ParBFT1: the end of the optimistic path, the end of the *prepare* phase, and the end of the final agreement. For brevity, we refer to these three points as t_1, t_2, and t_3, respectively. Next, we will analyze the safety of ParBFT1 in three situations.

Situation 1: A non-faulty replica commits d at t_1. In this situation, at least $f + 1$ non-faulty replicas have returned from the optimistic path, each of which will broadcast the optimistic candidate in the *prepare* phase. Therefore, every replica will receive at least one optimistic candidate among the $n - f$ PREP messages, and only Case 1 or Case 3 in Section 4.1 are possible. If a non-faulty replica is in Case 1, it will commit d directly. If it is in Case 3, it will broadcast the optimistic candidate (i.e., d) and invoke the ABA protocol with 0. In other words, each non-faulty replica will invoke the ABA protocol with 0, provided that it has not exited at t_1 or t_2. According to the validity property of ABA, the data output from ABA must be 0, and the data to be committed at t_3 must be d. Therefore, safety is guaranteed in this situation.

Algorithm 4 ParBFT1 protocol (for replica p_i)

Let v_i represent the data proposed by p_i.

1: **activate** OPTPATH1(v_i)
2: **activate** VABA(v_i)

3: **upon** receiving (HALT, d, σ) from p_j **do:**
4:   **commit** d
5:   **broadcast** (HALT, d, σ) **if** has not
6:   **exit**

7: **wait** for the output (d, σ) from OPTPATH1 or VABA
8: **if** the output is an optimistic candidate **then:**
9:   **terminate** the pessimistic path; $tag \leftarrow$ OPT
10: **else:**
11:   **terminate** the optimistic path; $tag \leftarrow$ PES
12: **activate** FINAGR with (tag, d, σ) **if** has not

13: **wait** for the output d from FINAGR
14: **commit** d
15: **exit**

Situation 2: A non-faulty replica commits d at t_2. According to Case 1 in Section 4.1, at least $f + 1$ non-faulty replicas must have broadcast the optimistic candidate in the *prepare* phase. The remaining analysis is identical to Situation 1.

Situation 3: A non-faulty replica commits d at t_3. If there are other non-faulty replicas that commit at t_1 or t_2, safety is guaranteed based on the analysis of Situation 1 and Situation 2. Therefore, we only need to consider the remaining situation where all the non-faulty replicas commit at t_3. According to the agreement property of ABA, non-faulty replicas will get the same output bit from ABA and thus commit the corresponding candidate. Since all the optimistic (respectively, pessimistic) candidates are identical, safety is guaranteed in this situation.

4.2.2 Liveness. Similar to the liveness analysis in ParBFT0, the liveness property of ParBFT1 is stated in Theorem 4, with its proof relying on Lemma 3.

LEMMA 3. *In ParBFT1, if no non-faulty replica commits at t_1 or t_2, every non-faulty replica will eventually invoke the ABA protocol.*

PROOF. This lemma is established through two cases.

Case 1: Some non-faulty replica p_i outputs from the optimistic path. According to Algorithm 3, p_i will broadcast its optimistic candidate during the *prepare* phase. Each non-faulty replica will receive this optimistic candidate. This ensures that each non-faulty replica can broadcast a PREP message and expect to receive at least $n - f$ PREP messages during the *prepare* phase. Then, every non-faulty replica will invoke the ABA protocol.

Case 2: No non-faulty replica outputs from the optimistic path. In this case, all non-faulty replicas will keep participating in the pessimistic path, eventually obtaining a pessimistic candidate according to the termination property of VABA. Every non-faulty replica can then broadcast a PREP message and invoke the ABA protocol after receiving $n - f$ PREP messages. □

THEOREM 4. *Every non-faulty replica in ParBFT1 can successfully commit in each consensus instance.*

PROOF. First, if some non-faulty replica p_i commits at t_1 or t_2, it will broadcast a `Halt` message. Every non-faulty replica will eventually receive this `Halt` message from p_i, leading them to commit if it has not yet. Next, if no non-faulty replica commits at t_1 or t_2, then due to Lemma 3, each non-faulty replica will invoke the ABA protocol. The termination property of ABA ensures that each non-faulty replica will eventually output from the ABA protocol. Based on the validity property of ABA, at least one non-faulty replica must have inputted the same bit as the output bit. According to Algorithm 3, that replica must have also broadcast the corresponding candidate. Therefore, each non-faulty replica will receive a candidate corresponding to the output bit and commit that candidate value. This concludes the proof of Theorem 4. □

4.3 Performance analysis

In a good situation with a non-faulty leader, a replica in ParBFT1 can commit at the end of the optimistic path, which has a latency of 5δ. In a bad situation characterized by a faulty leader, ParBFT1 takes 22δ to reach consensus, slightly larger than 21δ in ParBFT0, due to the additional *prepare* phase. Furthermore, since the pessimistic path always results in quadratic communication overhead, the optimistic path in ParBFT1 could be implemented using the normal-case protocol of PBFT [18], where each replica sends the vote to all replicas instead of only to the leader. This will give ParBFT1 a latency of 3δ under a non-faulty leader.

It is worth noting that if the adversary manipulates the network only slightly, ParBFT1 can still commit in the optimistic path. To be more specific, if $f + 1$ or more non-faulty replicas obtain the optimistic candidates earlier than pessimistic candidates, each non-faulty replica will receive at least one `PREP` message containing the optimistic candidate by the end of the *prepare* phase. Consequently, each non-faulty replica will invoke the ABA protocol with input 0. As indicated by the validity property, the ABA protocol will output 0 and each non-faulty replica will commit the optimistic candidate Furthermore, if all non-faulty replicas obtain optimistic candidates earlier, they can even take a shortcut to commit the optimistic candidate at the end of the *prepare* phase, bypassing the need to run the ABA protocol altogether.

However, since the pessimistic path is launched at the beginning, ParBFT1 has a message complexity of $O(n^2)$, even when the leader is non-faulty and the network is good, which is larger than the $O(tn)$ complexity of Ditto or BDT where t is the actual number of Byzantine replicas.

5 PARBFT2 WITH LOWER COMMUNICATION

To reduce the message complexity in good situations, we propose ParBFT2, whose key idea is to delay the launch of the pessimistic path by 5Δ. When it is in a good situation, the consensus can be reached through the optimistic path in 5Δ, without running the pessimistic path and avoiding the quadratic message complexity. Although ParBFT2 reintroduces the parameter Δ, its negative effects are not as severe as those in prior works. To be more specific, an incorrect estimation of Δ in Ditto or BDT can lead to premature switching from the optimistic path to the pessimistic path, resulting in both high latency and large communication overhead. In ParBFT2, incorrect estimation of Δ will only increase communication overhead. Furthermore, if the optimistic path is implemented using the chain structure, as detailed in Section 6.1, the timer for delaying the pessimistic path can be configured to 2Δ, same as in Ditto or BDT.

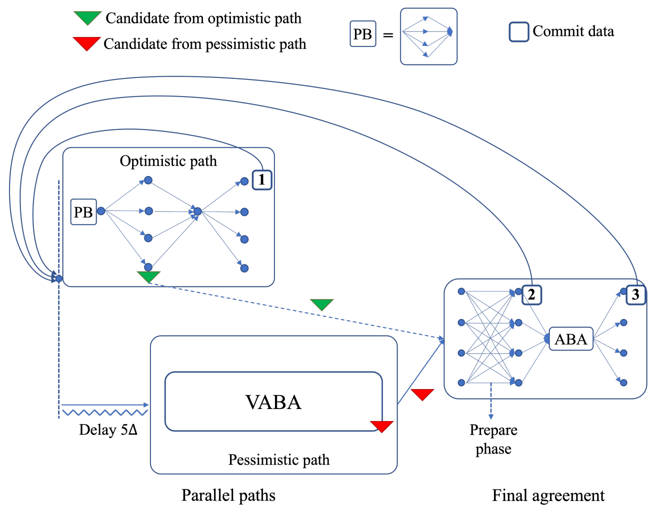

Figure 3: The structure of ParBFT2.

Algorithm 5 ParBFT2 protocol (for replica p_i)

Let v_i represent the data proposed by p_i.

$bd \leftarrow$ false // bd indicates whether p_i has broadcast data before
activate OPTPATH2(v_i)

upon receiving ($\texttt{HALT}, d, \sigma$) from p_j **do:**
  commit d
  if bd **then:**
    broadcast ($\texttt{HALT}, d, \sigma$) **if** has not
  exit

wait until the timer of 5Δ expires
$op1 \leftarrow$ OPTPATH2; $bd \leftarrow$ true
if $op1 \neq \perp$ **then:**
  parse $op1$ as (d, σ)
  activate FINAGR with ($\texttt{OPT}, d, \sigma$) **if** has not
else:
  activate VABA(v_i)
  wait for the output $op2$ from OPTPATH2 or VABA
  parse $op2$ as (d, σ)
  if $op2$ is an optimistic candidate **then:**
    terminate the pessimistic path; $tag \leftarrow \texttt{OPT}$
  else:
    terminate the optimistic path; $tag \leftarrow \texttt{PES}$
  activate FINAGR with (tag, d, σ) **if** has not

wait for the output d from FINAGR
commit d
exit

Algorithm 6 OPTPATH2: Optimistic path protocol in ParBFT2 (for replica p_i, with p_L as the leader)

Let v_i represent the data proposed by p_i. bd is a variable shared with Algorithm 5.

```
// Same as lines 1-11 of Algorithm 2 (OPTPATH1)

upon receiving (OPTH, d_o, σ_2) from p_L
    commit d_o
    if bd then:
        broadcast (HALT, d, σ) if has not
    exit
```

5.1 Description of ParBFT2

Figure 3 illustrates the structure of ParBFT2, which delays launching the pessimistic path by 5Δ. The rationale behind this delay is that, in a good situation, a replica is expected to commit on the optimistic path within 5Δ. To be more specific, a replica that cannot commit within this time period will check whether it has obtained the optimistic candidate. If it has, the replica will activate the final agreement with the optimistic candidate, avoiding the need to launch the pessimistic path. Otherwise, the replica will launch the pessimistic path.

Algorithm 5 describes the ParBFT2 protocol. It differs from ParBFT1 in that replicas do not activate the final agreement immediately after obtaining an optimistic candidate. Instead, the final agreement is activated only after the timer of 5Δ expires (Lines 8-12 of Algorithm 5), similar to the launch of the pessimistic path. Additionally, a replica that commits on the optimistic path or receives a Halt message will not always broadcast a Halt message to avoid introducing quadratic communication overhead. Instead, the replica will check if it has already activated FINAGR or VABA before. Only if this is true will it broadcast Halt messages. Furthermore, to ensure that each non-faulty replica can commit, a replica that has committed must send a Halt message to another replica p_j if it receives a FINAGR or VABA message from p_j, even though it has exited from the current ParBFT2 instance. It is worth noting that the partially synchronous BFT protocols such as HotStuff also use a similar design to help each non-faulty replica commit, where a non-faulty replica p_i responds to another replica p_j with the blocks lacked by p_j.

In fact, ParBFT2 can be viewed as an intermediate protocol between the serial-path protocols (i.e., Ditto/BDT) and ParBFT1. At one end of the spectrum, the serial-path protocols execute the optimistic and pessimistic paths in a serial manner. At the other end of the spectrum, ParBFT1 launches these two paths simultaneously in parallel. As an intermediate design point, ParBFT2 launches the two paths in a partially parallel fashion, with the pessimistic path being activated slightly later than the optimistic path.

5.2 Correctness analysis

It is evident that ParBFT2's safety proof is identical to that of ParBFT1, so we focus on liveness.

THEOREM 5. *Every non-faulty replica in ParBFT2 can successfully commit in each consensus instance.*

PROOF. We refer to the three points to commit in ParBFT2 as t_1, t_2, and t_3. We prove liveness by analyzing three cases.

Case 1: Some non-faulty replica p_i commits at t_1. If another non-faulty replica p_j cannot commit at t_1, it will trigger the execution of VABA and FINAGR. Then, p_i will receive a VABA/FINAGR message from p_j and will send a Halt message to help p_j commit as well. Thus, every non-faulty replica can commit in this case.

Case 2: No non-faulty replica commits at t_1, but some non-faulty replica p_i outputs from the optimistic path. In this case, p_i will broadcast its optimistic candidate during the *prepare* phase after the timer expires. Any non-faulty replica that has not output from the stage of parallel paths can obtain an optimistic candidate from p_i. Therefore, every non-faulty replica broadcasts a PREP message. If some non-faulty replica p_j manages to commit at the end of the *prepare* phase (i.e., t_2), it will broadcast a HALT message to help others commit as well. If no non-faulty replica commits at t_2, every non-faulty replica will advance to the ABA protocol. The termination and validity properties of ABA ensure that every non-faulty replica eventually commits, similar to the proof of Theorem 4.

Case 3: No non-faulty replica commits at t_1 or outputs from the optimistic path. In this case, each non-faulty replica will launch the pessimistic path after the timer expires. VABA's termination property guarantees that each non-faulty replica can obtain a pessimistic candidate. Subsequently, every non-faulty replica will broadcast a PREP message and invoke the ABA protocol. The remaining analysis is similar to Case 2.

To summarize, all non-faulty replicas in ParBFT2 commit. □

5.3 Performance analysis

In a good situation where the leader on the optimistic path is non-faulty, ParBFT2 can achieve the same latency of 5δ as ParBFT1. In a bad situation involving a faulty leader, ParBFT2's latency is 5Δ larger than ParBFT1, at an expected latency of $5\Delta + 22\delta$ due to the delay to the pessimistic path. However, by adopting the chain structure and the pipelining technique described in Section 6.1 and Appendix A, ParBFT2 can achieve a latency of $2\Delta + 25\delta$ under a faulty leader, which is the same as that of BDT.

Regarding the communication overhead, if it is in a good situation where the leader is non-faulty and $\delta \leq \Delta$, ParBFT2 can commit without launching the pessimistic path or activating the final agreement protocol. As a result, ParBFT2 has a message complexity of $O(tn)$, which is better than ParBFT1 and comparable to Ditto or BDT. On the contrary, if it is in a bad situation, the message complexity of ParBFT2 is $O(n^2)$, the same as ParBFT1, Ditto, and BDT.

As can be seen from Table 1, a wrong estimation of Δ in ParBFT2 will only increase the message complexity without affecting the consensus latency. We can think of ParBFT2 as making a trade-off between latency and communication over ParBFT1. To be more specific, ParBFT2 trades the larger latency under a Byzantine leader for a smaller message complexity in a good situation.

6 IMPLEMENTATION AND EVALUATION

In this section, we first introduce the chain-based version of ParBFT, which organizes data on the optimistic path into blocks that are chained one by one and processed in a pipelined manner to improve throughput. We then implement the chain-based system prototypes of both variants (i.e., ParBFT1 and ParBFT2) and conduct extensive experiments to evaluate their performance.

6.1 Chain-based ParBFT

In the previous description of ParBFT, we focused on a single instance of consensus to illustrate our main ideas more clearly. We can easily organize the data on the optimistic path across consecutive ParBFT instances into blocks and chain them together. This allows us to pipeline the processing of these blocks to improve throughput, as is commonly done in many partially-synchronous protocols [13, 67].

In general, the chain-based ParBFT proceeds in epochs, with blocks in an epoch indexed by increasing and successive height numbers. On the optimistic path of an epoch, the leader L_h of height h will create a *Quorum Certificate* (QC_{h-1}) by combing the partial threshold signatures on the block (B_{h-1}) of height $h-1$. After embedding QC_{h-1} in its newly created block B_h, L_h will broadcast B_h to other replicas. When a replica receives B_h, it will commit the block B_{h-2} and vote for B_h by sending its partial threshold signature on B_h to the leader L_{h+1} of height $h+1$. This optimistic path is similar to Tendermint [13] or two-chain HotStuff [67], where block processing is pipelined. The difference is that the chain-based ParBFT also attempts to launch a pessimistic path and then the final agreement protocol for each height, either immediately in ParBFT1 or delayed in ParBFT2. An epoch ends if any candidate from the pessimistic path gets committed, at which point the protocol moves on to the next epoch.

For chain-based ParBFT2, the timing parameter for delaying the pessimistic path can be set to 2Δ, resulting in a latency of $2\Delta + 25\delta$ under a faulty leader, as is shown in Table 1. Due to space constraints, we defer a detailed description of chain-based ParBFT to Appendix A. From now on, we refer to the chain-based ParBFT simply as ParBFT in the remainder of the paper when there is no ambiguity.

6.2 Implementation and experimental details

We implement the chain-based version of ParBFT in Golang (v1.17). Our implementation leverages several open-source libraries, including kyber[2] for threshold signatures, go-msgpack[3] for network communication, and gorpc[4] for synchronizing data payloads. We choose the MMR version of the ABA protocol [51] for implementation due to its simplicity. We are aware that the MMR protocol is vulnerable to liveness attacks if the adversary can arbitrarily manipulate message deliveries. This problem has known solutions [1, 47, 52], but it is not central to our paper.

Although there is an open-source implementation of BDT, it is written in Python, which generally has worse performance than Golang implementations. In addition, its pessimistic path uses Dumbo-MVBA [36], which is no longer the state-of-the-art. To ensure fairness, we implement our own version of BDT in Golang and give it a more efficient MVBA subroutine (i.e., sMVBA [35]) as its pessimistic path. For Ditto, we directly adopt its open-source Rust implementation[5]. For a lack of better heuristics, we follow the default configuration of BDT and Ditto that switch back to the optimistic path once a single agreement decision is reached on the pessimistic path.

We implement clients to send transactions to replicas at a rate controlled by a tunable configuration parameter. Additionally, we implement a mempool [31] to facilitate replicas to synchronize the data blocks in the background without embedding them into consensus messages. The payload size in the mempool is set to 512 KB. Each block proposal can contain hash digests of up to 32 payloads. Each hash digest is 32 bytes, making the maximum size of a block proposal 1 KB.

In Ditto's open-source implementation, a non-leader replica will create and broadcast a payload only after receiving enough transactions to fill a payload. This will lead to very large end-to-end latency when the input rate is low. To address this problem, we add an improvement on Ditto's mempool implementation: in addition to broadcasting a payload whenever it is full, a replica also broadcasts a payload one second after broadcasting the previous payload, even if the new payload is not full.

Our experiments are conducted in three different settings that attempt to capture the three situations in Table 1: (1) a good situation where the leader of the optimistic path is non-faulty and the network is good; (2) a situation with a non-faulty leader but a slow network; and (3) a situation where the leader is faulty.

We focus on the performance metrics of throughput and end-to-end latency. Throughput is calculated as the average number of committed transactions per second, while end-to-end latency is measured as the time it takes for a transaction to be committed since the client sends that transaction.

Each experiment lasts for five minutes and is repeated three times. Each data point in the rest of this section reports the average and is accompanied by error bars. The experiments are conducted on Amazon Web Service (AWS). Each replica is implemented on an m5d.2xlarge EC2 instance with 8 vCPUs, 32 GB memory, and a network bandwidth of up to 10 Gbps. The replicas are distributed across five AWS regions in a geo-distributed manner: US-East (N. Virginia), US-West (N. California), Asia-Pacific (Sydney), EU (Stockholm), and Asia-Pacific (Tokyo).

6.3 Performance in a good situation

In this section, we compare the performance of different protocols in a good situation. Specifically, we set the parameter Δ in Ditto, BDT, and ParBFT2 to 500 ms (*milliseconds*), leading to a timer setting of 1,000 ms (2Δ), which is significantly larger than the actual network delay. Our evaluation consists of two parts. Firstly, we analyze the relationship between latency and throughput for three system scales. Next, we conduct a more detailed comparison of latency as the number of replicas increases when the input rate does not saturate the system.

In the first part of our evaluation, we set the number of replicas to 10, 19, and 40, respectively. The results are shown in Figure 4. As anticipated, as the system scales up, all protocols exhibit a reduction in their peak throughput. For each replica count, ParBFT2 demonstrates a peak throughput comparable to BDT and Ditto. ParBFT1 also delivers a similar peak throughput when there are only 10 replicas, but as the system scales up, ParBFT1 shows worse performance

[2] https://github.com/dedis/kyber
[3] https://github.com/hashicorp/go-msgpack
[4] https://github.com/valyala/gorpc
[5] https://github.com/danielxiangzl/Ditto

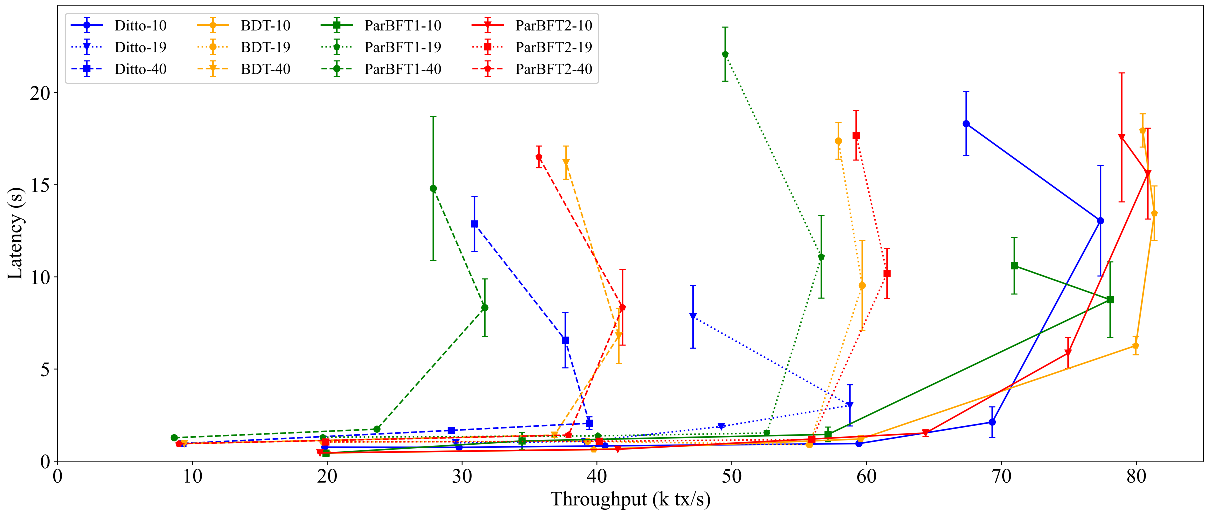

Figure 4: Latency vs throughput in a good network.

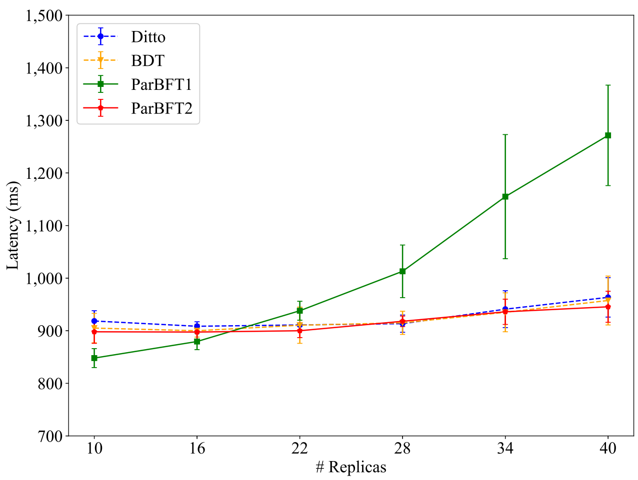

Figure 5: Latency comparison as the number of replicas increases in a good network.

than others due to its quadratic communication in the pessimistic path.

In the second part, we fix the input rate to 10,000 transactions per second and vary the number of replicas from ten to forty. The latency comparison is illustrated in Figure 5. Notably, Ditto, BDT, and ParBFT2 exhibit excellent scalability as the replica count increases, sustaining a 900~1,000 ms latency with up to 40 replicas. This is because in the good case, Ditto or BDT do not switch to the pessimistic path, and ParBFT2 need not launch the pessimistic path. On the other hand, ParBFT1 demonstrates poor scalability as the replica count increases, again due to its quadratic message complexity. It is worth noting that ParBFT1 has an advantage in latency over other protocols when the number of replicas is small. The reason is that replicas in ParBFT1 can promptly activate the *prepare* phase within the final agreement protocol upon receiving a subsequent block (or receiving output from `PB1` in Figure 2). The *prepare* phase empowers replicas to commit a block within one round of communication, in contrast to the two rounds mandated by the optimistic path.

6.4 Performance in a slow network

In this situation, we simulate a slow network by adding delays to all messages. We introduce a new delay parameter ζ. We note that ζ represents an artificial delay added to all messages, so the final message delay would be ζ plus the original network delay. We fix the number of replicas at sixteen and retain the same 500 ms value of Δ as in Section 6.3. Our experiments include two parts: the first part depicts the relationship between latency and throughput, while the second part explores how the latency changes as the artificial delay ζ increases.

For the first part, we try two ζ values: 200 ms and 600 ms. The results are given in Figure 6. When ζ is 200 ms, all the protocols exhibit similar performance. When ζ is set to 600 ms, Ditto and BDT suffer considerably worse performance compared to ParBFT1 or ParBFT2. This is the case where Ditto and BDT fail to commit in their optimistic paths and switch to the pessimistic path after the timeout is triggered. Although the timer also expires and the pessimistic path is launched in ParBFT2, the optimistic path will still finish faster than the pessimistic path, enabling ParBFT2 to commit through the optimistic path, without having to finish the entire pessimistic path.

In the second part, we fix the input rate to 10,000 transactions per second and vary the value of ζ from 0 ms to 700 ms in increments of 100 ms. The experimental results are presented in Figure 7. As shown, the performance of Ditto and BDT deteriorates significantly when ζ exceeds 500 ms. By contrast, the performance of ParBFT1 and ParBFT2 degrades in a gradual manner.

An interesting phenomenon captured by Figure 7 is the initial lower latency of ParBFT1 compared to ParBFT2. As the value of ζ increases, this latency difference becomes larger. However, eventually, the latency of ParBFT2 converges to a level similar to ParBFT1. The reason for this trend is that at the start of small ζ, ParBFT1

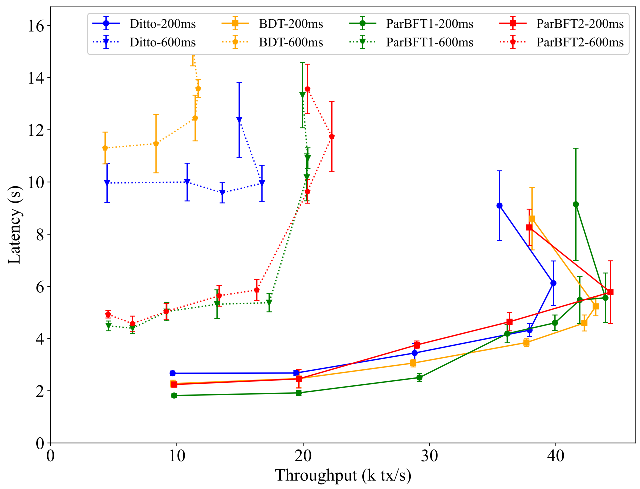

Figure 6: Latency vs throughput in a slow network.

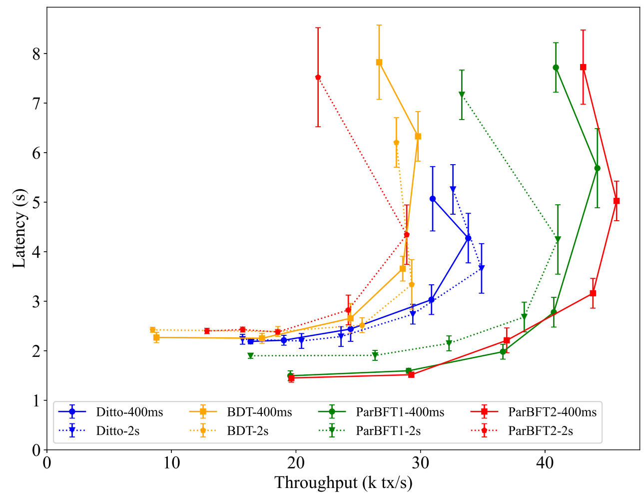

Figure 8: Latency vs throughput under a faulty leader.

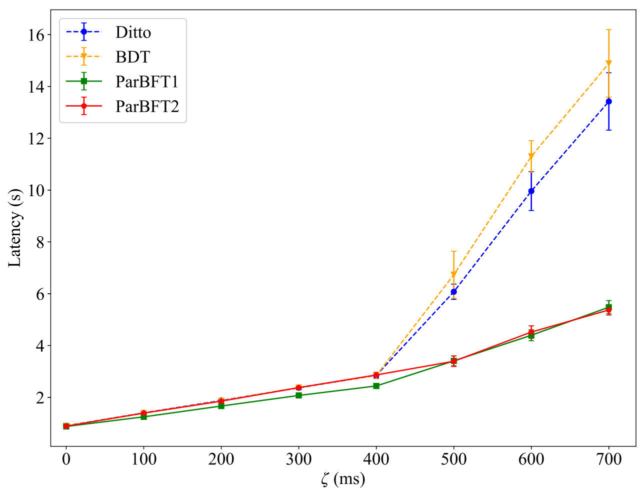

Figure 7: Latency comparison as the added delay increases in a slow network.

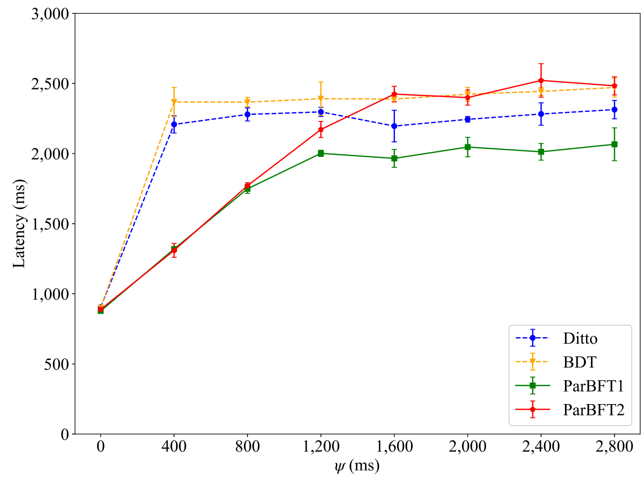

Figure 9: Latency comparison as the leader becomes slower.

can benefit from early decision in the *prepare* phase in contrast to ParBFT2, as we have discussed in Section 6.3. As ζ increases from 0 ms to 400 ms, the benefits of one less communication round in ParBFT1 become more and more significant, leading to an increasing latency difference. However, when ζ reaches 500 ms, the timer in ParBFT2 expires and the *prepare* phase is activated. In this case, ParBFT2 also benefits from the *prepare* phase, similar to ParBFT1, and hence achieves comparable performance.

6.5 Performance under a faulty leader

In this section, we examine the situation where the leader is faulty. Although a Byzantine faulty leader can behave arbitrarily, it is reasonable to focus on a crashed or slow leader. This is because the leader's power in ParBFT is limited to the optimistic path. The worst disruption a faulty leader can cause is to spoil the optimistic path, which can be achieved by simply crashing or being slow. Thus, we delay the block proposals from the leader by a parameter of ψ, through which we can observe the performance change under different ψ values. For this group of experiments, we fix the number of replicas at sixteen. The parameter Δ is configured at 250 ms, resulting in a timer of 500 ms. Our experiments again include two parts: the first part shows the relationship between latency and throughput, and the second part analyzes the latency as a function of ψ.

In the first part, we try two values of ψ: 400 ms and 2 seconds. Experimental results are shown in Figure 8. From the figure, we see that when ψ is set to 400 ms, both ParBFT1 and ParBFT2 demonstrate superior performance compared to Ditto or BDT. In this case, Ditto and BDT will switch to run the pessimistic path. Despite the timer also expiring in ParBFT2, ParBFT2 can still commit in the optimistic path similar to the previous situation. When ψ is set to 2 seconds, all protocols resort to the pessimistic path to commit. In this case, ParBFT1 outperforms the other protocols due to the simultaneous launch of both two paths. In contrast, Ditto, BDT, and ParBFT2 activate the pessimistic path only after a timer expires.

For the second part, we set the input rate to 10,000 transactions per second while varying the value of ψ from 0 ms to 2,800 ms in increments of 400 ms. The results of these experiments are shown in Figure 9. We can immediately notice that the latency of all protocols grows when the block proposals are delayed. Upon a more careful comparison, we see that Ditto and BDT experience a sharp increase in latency when ψ reaches 400 ms, due to the expiration of the timer and consequent path switch. In contrast, the latency of ParBFT1 and ParBFT2 increases gradually, thanks to the early decision in the *prepare* phase. Specifically, in the case of ParBFT2, a block can still be committed at the end of the *prepare* phase, even after the timer expires and the pessimistic path is launched when ψ exceeds 400 ms. In terms of the final steady performance, all protocols demonstrate a high latency, as a result of running the pessimistic path. However, BDT and ParBFT2 exhibit slightly larger latency than Ditto, possibly due to the additional usage of an ABA protocol.

7 RELATED WORK

Based on different timing assumptions, BFT protocols can be classified into three categories: synchronous, partially-synchronous, and asynchronous.

7.1 Synchronous BFT protocols

The pioneering works of Pease et al. [43, 53] introduce the problem of Byzantine agreement, originally in a synchronous network where messages between non-faulty replicas are delivered in a timely manner. Assuming a network delay upper bound (i.e., Δ), early synchronous protocols coordinate all the replicas to proceed in a lock-step manner [2, 9, 25, 28, 38]. However, this approach is caught in a delicate dilemma between security and efficiency. If Δ is set too small, the synchrony will be violated, and the protocol will lose safety. On the other hand, if Δ is set too large, each lock-step round will take a long time, causing unnecessary delays and poor performance. For this reason, synchronous BFT consensus protocols have long been considered impractical. Recent works such as Sync HotStuff [4] alleviated this problem by embracing a non-lockstep model of synchrony, enabling replicas to advance more quickly to the next steps and minimizing the protocol's performance dependency on Δ. Despite the improvement, synchronous protocols, including Sync HotStuff, still have their performance fundamentally dependent on Δ and thus still face the dilemma of incorrect estimation of Δ.

7.2 Partially-synchronous BFT protocols

The partial synchrony model proposed by Dwork et al. [27] opens up a new avenue for BFT consensus protocol design. PBFT [18], based on a partially synchrony model and using the view-based design, becomes the de facto standard for practical BFT consensus for over a decade. To reduce the (already low) latency of PBFT from three rounds to two rounds, a range of works propose adding a fast path. These include Zyzzyva [41], FastBFT [44], SBFT [34], and Trebiz [21]. More recently, the emergence of blockchains inspires further simplification of the view-based partially synchronous BFT paradigm protocol with the new chain-based structures of blocks, as seen in Tendermint [13], Casper FFG [14], HotStuff [67], and Streamlet [19]. Although partially synchronous protocols exhibit decent performance in the good case, they have recently been criticized for being vulnerable to liveness attack [49]. To be more specific, even with a non-faulty leader, the adversary may construct an elaborate network scheduler that blocks messages to and from the leader until the leader is demoted. This results in a loss of liveness.

Aublin et al. propose a black-box framework to switch between multiple protocols [7] to get their respective benefits. Their framework adopts the serial-path paradigm. The two baselines considered in our work, Ditto and BDT, can be viewed as concrete instantiations of this framework.

Some recent works explore an orthogonal direction of employing multiple leaders to concurrently drive multiple consensus instances [60, 61] to improve throughput. In contrast, ParBFT runs two parallel paths within each single consensus instance to accelerate the instance.

7.3 Asynchronous BFT protocols

Research on the asynchronous BFT protocols dates back to the 1980s [8, 12, 17, 20]. Asynchronous BFT broadcast protocols enable replicas to deliver the same message from a designated broadcaster, with Bracha's reliable broadcast [11] and Dolev's consistent broadcast [24] being notable examples. These protocols are typically used as subroutines in the Byzantine consensus or state machine replication protocols. The famous FLP impossibility states that asynchronous BFT consensus protocols must make use of randomness [29]. Early works in this area include Ben-Or [8], Canetti-Rabin [17], CKPS [15] and SINTRA [16]. Many works focus on the simpler problem of agreeing on a single bit (0 or 1), also known as *Asynchronous Binary Agreement* (ABA) [1, 8, 30, 51]. Recent practical advances in synchronous BFT include HoneybadgerBFT [49], the Dumbo family of protocols [35, 36, 46], and *Directed Acyclic Graph* (DAG)-based protocols [22, 39, 57, 59].

Although asynchronous consensus protocols are more robust than partially synchronous ones, they generally have inferior performance. To match the performance of partially synchronous protocols, a number of works propose adding an optimistic path, which is often adapted from a partially synchronous protocol, and use the original asynchronous protocol as a pessimistic fallback [32, 45]. We have discussed the drawbacks of this design extensively, and it is also the motivation of our work.

Some recent works combine synchronous and asynchronous protocols to improve fault tolerance [6, 10, 48, 50]. It is well known that asynchronous (and partially-synchronous) protocols tolerate at most $n/3$ Byzantine faults while synchronous protocols tolerate up to $n/2$ Byzantine faults. These works aim to tolerate more than $n/3$ Byzantine faults in the good case when the network happens to be synchronous. In contrast, ParBFT focuses on improving performance in the good case.

8 CONCLUSION

The existing serial-path BFT consensus protocols can result in significant latency if the network delay is incorrectly estimated. To deal with this problem, we propose ParBFT, whose intuitive idea is to parallelize the optimistic and pessimistic paths. In general, ParBFT can achieve a low latency of 5δ as long as the leader on the optimistic path is non-faulty, without requiring estimation of the network delay. We present two variants of ParBFT (i.e., ParBFT1 and ParBFT2) that offer a trade-off between latency and communication overhead. To enhance system throughput, we also introduce the chain-based version of ParBFT, which incorporates chain structure and pipeline technology into the optimistic path. We prove that ParBFT can guarantee both liveness and safety, and our experimental results demonstrate its feasibility and efficiency.

ACKNOWLEDGMENTS

We thank Atuski Momose for helpful suggestions, as well as Zhuolun Xiang and Guanxiong Wang for their assistance with experiments. This work is funded in part by NSF award 2143058 and National Natural Science Foundation of China (Grant No. 62202187).

REFERENCES

[1] Ittai Abraham, Naama Ben-David, and Sravya Yandamuri. 2022. Efficient and Adaptively Secure Asynchronous Binary Agreement via Binding Crusader Agreement. In *Proceedings of the 2022 ACM Symposium on Principles of Distributed Computing*. ACM, 381–391.

[2] Ittai Abraham, Srinivas Devadas, Danny Dolev, Kartik Nayak, and Ling Ren. 2019. 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, 320–334.

[3] Ittai Abraham, Philipp Jovanovic, Mary Maller, Sarah Meiklejohn, Gilad Stern, and Alin Tomescu. 2021. Reaching Consensus for Asynchronous Distributed Key Generation. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*. ACM, 363–373.

[4] Ittai Abraham, Dahlia Malkhi, Kartik Nayak, Ling Ren, and Maofan Yin. 2020. Sync Hotstuff: Simple and Practical Synchronous State Machine Replication. In *Proceedings of the 41st IEEE Symposium on Security and Privacy*. IEEE, 106–118.

[5] Ittai Abraham, Dahlia Malkhi, and Alexander Spiegelman. 2019. Asymptotically Optimal Validated Asynchronous Byzantine Agreement. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*. ACM, 337–346.

[6] Andreea B Alexandru, Erica Blum, Jonathan Katz, and Julian Loss. 2022. State Machine Replication under Changing Network Conditions. In *Proceedings of the 2022 International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 681–710.

[7] Pierre-Louis Aublin, Rachid Guerraoui, Nikola Knežević, Vivien Quéma, and Marko Vukolić. 2015. The Next 700 BFT Protocols. *ACM Transactions on Computer Systems* 32, 4 (2015), 1–45.

[8] Michael Ben-Or. 1983. Another Advantage of Free Choice: Completely Asynchronous Agreement Protocols. In *Proceedings of the 2nd Annual ACM Symposium on Principles of Distributed Computing*. ACM, 27–30.

[9] Piotr Berman, Juan A Garay, and Kenneth J Perry. 1992. Bit Optimal Distributed Consensus. *Computer Science Research* (1992), 313–322.

[10] Erica Blum, Jonathan Katz, and Julian Loss. 2021. 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, 547–572.

[11] Gabriel Bracha. 1987. Asynchronous Byzantine Agreement Protocols. *Information and Computation* 75, 2 (1987), 130–143.

[12] Gabriel Bracha and Sam Toueg. 1985. Asynchronous Consensus and Broadcast Protocols. *Journal of the ACM* 32, 4 (1985), 824–840.

[13] Ethan Buchman. 2016. *Tendermint: Byzantine Fault Tolerance in the Age of Blockchains*. Ph.D. Dissertation. University of Guelph.

[14] Vitalik Buterin and Virgil Griffith. 2017. Casper the Friendly Finality Gadget. *arXiv preprint arXiv:1710.09437* (2017).

[15] Christian Cachin, Klaus Kursawe, Frank Petzold, and Victor Shoup. 2001. Secure and Efficient Asynchronous Broadcast Protocols. In *Proceedings of the 2001 Annual International Cryptology Conference*. Springer, 524–541.

[16] Christian Cachin and Jonathan A Poritz. 2002. Secure Intrusion-tolerant Replication on the Internet. In *Proceedings of the 2002 International Conference on Dependable Systems and Networks*. IEEE, 167–176.

[17] Ran Canetti and Tal Rabin. 1993. Fast Asynchronous Byzantine Agreement with Optimal Resilience. In *Proceedings of the 25th Annual ACM Symposium on Theory of Computing*. ACM, 42–51.

[18] Miguel Castro and Barbara Liskov. 1999. Practical Byzantine Fault Tolerance. In *Proceedings of the 1999 USENIX Symposium on Operating Systems Design and Implementation*. USENIX, 173–186.

[19] Benjamin Y Chan and Elaine Shi. 2020. Streamlet: Textbook Streamlined Blockchains. In *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies*. ACM, 1–11.

[20] Miguel Correia, Nuno Ferreira Neves, and Paulo Veríssimo. 2006. From Consensus to Atomic Broadcast: Time-free Byzantine-resistant Protocols without Signatures. *The Computer Journal* 49, 1 (2006), 82–96.

[21] Xiaohai Dai, Liping Huang, Jiang Xiao, Zhaonan Zhang, Xia Xie, and Hai Jin. 2022. Trebiz: Byzantine Fault Tolerance with Byzantine Merchants. In *Proceedings of the 38th Annual Computer Security Applications Conference*. ACM, 923–935.

[22] George Danezis, Lefteris Kokoris-Kogias, Alberto Sonnino, and Alexander Spiegelman. 2022. Narwhal and Tusk: A DAG-based Mempool and Efficient BFT Consensus. In *Proceedings of the 17th European Conference on Computer Systems*. ACM, 34–50.

[23] Sourav Das, Thomas Yurek, Zhuolun Xiang, Andrew Miller, Lefteris Kokoris-Kogias, and Ling Ren. 2022. Practical Asynchronous Distributed Key Generation. In *Proceedings of the 2022 IEEE Symposium on Security and Privacy*. IEEE, 2518–2534.

[24] Danny Dolev. 1982. The Byzantine Generals Strike Again. *Journal of Algorithms* 3, 1 (1982), 14–30.

[25] Danny Dolev and H. Raymond Strong. 1983. Authenticated Algorithms for Byzantine Agreement. *SIAM Journal on Computing* 12, 4 (1983), 656–666.

[26] Sisi Duan, Michael K Reiter, and Haibin Zhang. 2018. BEAT: Asynchronous BFT Made Practical. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2028–2041.

[27] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. 1988. Consensus in the Presence of Partial Synchrony. *Journal of the ACM* 35, 2 (1988), 288–323.

[28] Paul Feldman and Silvio Micali. 1988. Optimal Algorithms for Byzantine Agreement. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing*. ACM, 148–161.

[29] Michael J Fischer, Nancy A Lynch, and Michael S Paterson. 1985. Impossibility of Distributed Consensus with One Faulty Process. *Journal of the ACM* 32, 2 (1985), 374–382.

[30] Roy Friedman, Achour Mostefaoui, and Michel Raynal. 2005. Simple and Efficient Oracle-based Consensus Protocols for Asynchronous Byzantine Systems. *IEEE Transactions on Dependable and Secure Computing* 2, 1 (2005), 46–56.

[31] Yingzi Gao, Yuan Lu, Zhenliang Lu, Qiang Tang, Jing Xu, and Zhenfeng Zhang. 2022. Dumbo-ng: Fast Asynchronous BFT Consensus with Throughput-oblivious Latency. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 1187–1201.

[32] Rati Gelashvili, Lefteris Kokoris-Kogias, Alberto Sonnino, Alexander Spiegelman, and Zhuolun Xiang. 2022. Jolteon and Ditto: Network-adaptive Efficient Consensus with Asynchronous Fallback. In *Proceedings of the 2022 International Conference on Financial Cryptography and Data Security*. Springer, 296–315.

[33] Vincent Gramoli. 2020. From Blockchain Consensus back to Byzantine Consensus. *Future Generation Computer Systems* 107 (2020), 760–769.

[34] Guy Golan Gueta, Ittai Abraham, Shelly Grossman, Dahlia Malkhi, Benny Pinkas, Michael Reiter, Dragos-Adrian Seredinschi, Orr Tamir, and Alin Tomescu. 2019. Sbft: A Scalable and Decentralized Trust Infrastructure. In *Proceedings of the 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. IEEE, 568–580.

[35] Bingyong Guo, Yuan Lu, Zhenliang Lu, Qiang Tang, Jing Xu, and Zhenfeng Zhang. 2022. Speeding Dumbo: Pushing Asynchronous BFT Closer to Practice. *Cryptology ePrint Archive* (2022).

[36] Bingyong Guo, Zhenliang Lu, Qiang Tang, Jing Xu, and Zhenfeng Zhang. 2020. Dumbo: Faster Asynchronous BFT Protocols. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 803–818.

[37] Hai Jin and Jiang Xiao. 2022. Towards Trustworthy Blockchain Systems in the Era of "Internet of Value": Development, Challenges, and Future Trends. *Science China Information Sciences* 65 (2022), 1–11.

[38] Jonathan Katz and Chiu-Yuen Koo. 2006. On Expected Constant-round Protocols for Byzantine Agreement. In *Proceedings of the 2006 Annual International Cryptology Conference*. Springer, 445–462.

[39] Idit Keidar, Eleftherios Kokoris-Kogias, Oded Naor, and Alexander Spiegelman. 2021. All You Need is DAG. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*. ACM, 165–175.

[40] Eleftherios Kokoris Kogias, Dahlia Malkhi, and Alexander Spiegelman. 2020. Asynchronous Distributed Key Generation for Computationally-Secure Randomness, Consensus, and Threshold Signatures. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 1751–1767.

[41] Ramakrishna Kotla, Lorenzo Alvisi, Mike Dahlin, Allen Clement, and Edmund Wong. 2007. Zyzzyva: Speculative Byzantine Fault Tolerance. In *Proceedings of 21st ACM SIGOPS Symposium on Operating Systems Principles*. ACM, 45–58.

[42] Klaus Kursawe and Victor Shoup. 2005. Optimistic Asynchronous Atomic Broadcast. In *Proceedings of the 32nd International Colloquium on Automata, Languages and Programming*. Springer, 204–215.

[43] Leslie Lamport, Robert Shostak, and Marshall Pease. 1982. The Byzantine Generals Problem. *ACM Transactions on Programming Languages and Systems* 4, 3 (1982), 382–401.

[44] Jian Liu, Wenting Li, Ghassan O Karame, and N Asokan. 2018. Scalable Byzantine Consensus via Hardware-assisted Secret Sharing. *IEEE Transactions on Computers* 68, 1 (2018), 139–151.
[45] Yuan Lu, Zhenliang Lu, and Qiang Tang. 2022. 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, 2159–2173.
[46] Yuan Lu, Zhenliang Lu, Qiang Tang, and Guiling Wang. 2020. Dumbo-mvba: Optimal Multi-valued Validated Asynchronous Byzantine Agreement, Revisited. In *Proceedings of the 39th ACM Symposium on Principles of Distributed Computing*. ACM, 129–138.
[47] Ethan MacBrough. 2018. Cobalt: BFT Governance in Open Networks. *arXiv preprint arXiv:1802.07240* (2018).
[48] Dahlia Malkhi, Kartik Nayak, and Ling Ren. 2019. Flexible Byzantine Fault Tolerance. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 1041–1053.
[49] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. 2016. The Honey Badger of BFT Protocols. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 31–42.
[50] Atsuki Momose and Ling Ren. 2021. Multi-threshold Byzantine Fault Tolerance. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 1686–1699.
[51] Achour Mostéfaoui, Hamouma Moumen, and Michel Raynal. 2014. 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, 2–9.
[52] Achour Mostéfaoui, Hamouma Moumen, and Michel Raynal. 2015. Signature-free Asynchronous Binary Byzantine Consensus with t< n/3, O (n2) Messages, and O (1) Expected Time. *Journal of the ACM* 62, 4 (2015), 1–21.
[53] Marshall Pease, Robert Shostak, and Leslie Lamport. 1980. Reaching Agreement in the Presence of Faults. *Journal of the ACM* 27, 2 (1980), 228–234.
[54] Marc Pilkington. 2016. Blockchain Technology: Principles and Applications. In *Research handbook on digital transformations*. Edward Elgar Publishing.
[55] Michael O Rabin. 1983. Randomized Byzantine Generals. In *Proceedings of the 24th Annual Symposium on Foundations of Computer Science*. IEEE, 403–409.
[56] HariGovind V Ramasamy and Christian Cachin. 2005. Parsimonious Asynchronous Byzantine-Fault-Tolerant Atomic Broadcast. In *Proceedings of the 2005 International Conference On Principles Of Distributed Systems*. Springer, 88–102.
[57] Maria A Schett and George Danezis. 2021. Embedding A Deterministic BFT Protocol in A Block DAG. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*. ACM, 177–186.
[58] Alexander Spiegelman. 2020. In Search for An Optimal Authenticated Byzantine Agreement. *arXiv preprint arXiv:2002.06993* (2020).
[59] Alexander Spiegelman, Neil Giridharan, Alberto Sonnino, and Lefteris Kokoris-Kogias. 2022. Bullshark: DAG BFT Protocols Made Practical. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2705–2718.
[60] Chrysoula Stathakopoulou, Matej Pavlovic, and Marko Vukolić. 2022. State Machine Replication Scalability Made Simple. In *Proceedings of the 17th European Conference on Computer Systems*. ACM, 17–33.
[61] Chrysoula Stathakopoulou, David Tudor, Matej Pavlovic, and M Vukolić. 2022. Mir-BFT: Scalable and Robust BFT for Decentralized Networks. *Journal of Systems Research* 2, 1 (2022).
[62] Sam Toueg. 1984. Randomized Byzantine Agreements. In *Proceedings of the 3rd Annual ACM Symposium on Principles of Distributed Computing*. ACM, 163–178.
[63] Jun Wan, Atsuki Momose, Ling Ren, Elaine Shi, and Zhuolun Xiang. 2023. On the Amortized Communication Complexity of Byzantine Broadcast. In *Proceedings of the 2023 ACM Symposium on Principles of Distributed Computing*. ACM, 253–261.
[64] Xin Wang, Sisi Duan, James Clavin, and Haibin Zhang. 2022. BFT in Blockchains: From Protocols to Use Cases. *ACM Computing Surveys* 54, 10 (2022), 1–37.
[65] Karl Wüst and Arthur Gervais. 2018. Do You Need A Blockchain. In *Proceedings of the 2018 Crypto Valley Conference on Blockchain Technology*. IEEE, 45–54.
[66] Yang Xiao, Ning Zhang, Wenjing Lou, and Y Thomas Hou. 2020. A Survey of Distributed Consensus Protocols for Blockchain Networks. *IEEE Communications Surveys & Tutorials* 22, 2 (2020), 1432–1465.
[67] Maofan Yin, Dahlia Malkhi, Michael K Reiter, Guy Golan Gueta, and Ittai Abraham. 2019. Hotstuff: BFT Consensus with Linearity and Responsiveness. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*. ACM, 347–356.

A CHAIN-BASED PARBFT

To improve the system throughput, we introduce the chain structures to the optimistic path of ParBFT, which enables the blocks to be processed in a pipelining manner. Corresponding to different variants of ParBFT, we devise the chain-based ParBFT1 and ParBFT2, which are described in Algorithm 7 and Algorithm 8, respectively.

Like other chain-based protocols such as HotStuff [67], all the data in chain-based ParBFT are organized into blocks, with each one indexed by a height number and containing a *Quorum Certificate* (QC) of the previous one. The QC is a complete threshold signature, which is created by combining partial threshold signatures from $n-f$ replicas. Each block B_h also contains a data bulk b_h, consisting of transactions to be processed. ParBFT operates in successive epochs, and different epochs are independent, which means there is no complex epoch-change mechanism to switch from one epoch to another. When a replica exits the current epoch, it can directly enter the next epoch. Either Algorithm 7 or Algorithm 8 describes the protocol in a single epoch.

A.1 Chain-based ParBFT1

Blocks within each epoch are numbered starting from height 0, and each epoch is initialized with a blank block denoted as B_0, which contains an empty data bulk b_h and an empty QC $\perp$. In the chain-based ParBFT1, every replica activates a new epoch by broadcasting its vote for B_0, which includes a partial threshold signature on B_0. It is important to note that broadcasting votes on the optimistic path ensures that the pessimistic path can be launched simultaneously with the optimistic path, as shown in Lines 14-16 of Algorithm 7. Although the broadcast of votes results in quadratic message complexity, it does not significantly affect ParBFT1, since the pessimistic path already brings quadratic message complexity to the protocol.

Blocks on the optimistic path are committed through the two-chain structure. Upon receiving a block B_h, the block B_{h-2} can be committed, and the pessimistic path and the final agreement protocol for the height $h-2$ can be terminated, as shown in Lines 20-21 of Algorithm 7. Additionally, the final agreement protocol for the height $h-1$ can be activated with the optimistic candidate, as shown in Lines 22-23. On the contrary, if the replica obtains the pessimistic candidate for height h, it will activate the final agreement protocol FinAgr_h with the pessimistic candidate and stop voting for the block B_{h+1} on the optimistic path (Lines 24-26).

Once a block is committed through the final agreement protocol FinAgr_h and the block is a pessimistic candidate, the replica will exit the current epoch. To ensure that all the replicas will exit at the same height and commit identical blocks, a replica will wait to find the final agreement protocol with the smallest height number that commits the pessimistic candidate, denoted by FA_s. The replica then commits the block B_s, discards all blocks with heights larger than s, and terminates all pessimistic paths and final agreement protocols with heights greater than s. After this, the replica exits the current epoch and enters the next.

Moreover, the chain-based ParBFT1 includes a block retrieval mechanism similar to other chain-based protocols such as HotStuff. If a replica p_i receives a new block B from leader p_L, but it lacks some ancestor blocks of B, p_i will send a request to p_L to retrieve the missing blocks. Only if all ancestor blocks of B are received can p_i accept B as valid. As this block retrieval mechanism is common in many chain-based protocols, detailed descriptions of it are omitted here.

Algorithm 7 CHAINPARBFT1: Chain-based ParBFT1 protocol (for replica p_i)

Let b_h represent a data bulk extracted from the mempool. *SignShare* and *Combine* denote the threshold signature functions, and *create_block* denotes the function that creates a block based on the data bulk and QC.

1: **initialize** $S \leftarrow [\bot, \bot, ..., \bot]$
2: **broadcast** $(\texttt{VOTE}, B_0, SignShare_{n-f}(B_0))$

3: **upon** receiving $(\texttt{VOTE}, B_h, \rho_h)$ from replica p_j **do:**
4:   **if** has not voted for B_h **then:**
5:     **broadcast** $(\texttt{VOTE}, B_h, SignShare_{n-f}(B_h))$
6:   $S[h] \leftarrow S[h] \cup \rho_h$
7:   **if** VABA_h has not been activated **then:**
8:     **extract** QC_{h-1} from B_h
9:     $B'_h \leftarrow create\_block(h, b_h, QC_{h-1})$
10:     **activate** $\text{VABA}_h(B'_h)$
11:   **if** $|S[h]| = n - f$ **then:**
12:     $QC_h \leftarrow Combine_{n-f}(B_h, S[h])$
13:     $B_{h+1} \leftarrow create\_block(h+1, b_{h+1}, QC_h)$
14:     **if** pi is leader of $h+1$ **then:**
15:       **broadcast** B_{h+1}
16:     **activate** $\text{VABA}_{h+1}(B_{h+1})$

17: **upon** receiving B_h from the leader of h **do:**
18:   **if** has not voted for B_h **then:**
19:     **broadcast** $(\texttt{VOTE}, B_h, SignShare_{n-f}(B_h))$
20:   **terminate** VABA_{h-2} and FINAGR_{h-2}
21:   **commit** B_{h-2}
22:   **extract** QC_{h-1} from B_h
23:   **activate** FINAGR_{h-1} with $(\texttt{OPT}, B_{h-1}, QC_{h-1})$ **if** has not

24: **upon** receiving (B_h, σ_p) from VABA_h **do:**
25:   **activate** FINAGR_h with $(\texttt{PES}, B_h, \sigma_p)$ **if** has not
26:   **stop** voting for the optimistic block B_{h+1}

27: **upon** receiving B_h from FINAGR_h **do:**
28:   **if** B_h is a pessimistic candidate, **then:**
29:     **wait** for each FINAGR_k (k<h) are terminated or finished
30:     $\text{FA}_s \leftarrow$ lowest FINAGR outputting pessimistic candidate
31:     **extract** B_s from FA_s
32:     **commit** B_s **if** has not
33:     **discard** all the blocks $B_k (k > s)$
34:     **terminate** all the VABA_k and $\text{FINAGR}_k (k > s)$
35:     **exit**

A.2 Chain-based ParBFT2

Unlike chain-based ParBFT1, in chain-based ParBFT2, votes are only sent to the leader, avoiding quadratic message complexity. Additionally, the timeout parameter in chain-based ParBFT2 is set as 2Δ instead of 5Δ in non-chain ParBFT2. At the beginning of each epoch, each replica sets a timer and waits for B_1 (Line 3 of Algorithm 8).

When a block B_h is received, the timer for the next height $h+1$ starts. If B_{h+1} is not received within 2Δ, the timeout event is triggered, and the replica activates the pessimistic path at h (Lines 4-6 and 21-23). Moreover, the final agreement protocol at $h-1$ will be activated with an optimistic candidate (Lines 18-19). The remaining parts of chain-based ParBFT2 are the same as chain-based ParBFT1. Since the pessimistic path in chain-based ParBFT2 is delayed by 2Δ, it can achieve an expected latency of $2\Delta + 25\delta$ when the leader on the optimistic path is faulty.

Algorithm 8 CHAINPARBFT2: Chain-based ParBFT2 protocol (for replica p_i)

Let b_h represent a data bulk extracted from the mempool. *SignShare* and *Combine* denote the threshold signature functions, and *create_block* denotes the function that creates a block based on the data bulk and QC.

1: **initialize** $S \leftarrow [\bot, \bot, ..., \bot]$
2: **send** $(\texttt{VOTE}, B_0, SignShare_{n-f}(B_0))$ to leader of height 1

3: **wait** until timer of 2Δ expires **or** B_1 is received
4: **if** B_1 is not received **then:**
5:   $B'_0 \leftarrow create\_block(0, b_0, \bot)$
6:   **activate** $\text{VABA}_0(B'_0)$

7: **upon** receiving $(\texttt{VOTE}, B_h, \rho_h)$ from replica p_j **do:**
8:   $S[h] \leftarrow S[h] \cup \rho_h$
9:   **if** $|S[h]| = n - f$ **and** p_i is leader of $h+1$ **then:**
10:     $QC_h \leftarrow Combine_{n-f}(B_h, S[h])$
11:     $B_{h+1} \leftarrow create\_block(h+1, b_{h+1}, QC_h)$
12:     **broadcast** B_{h+1}

13: **upon** receiving B_h from the leader of h **do:**
14:   **send** $(\texttt{VOTE}, B_h, SignShare_{n-f}(B_h))$ to leader of $h+1$
15:   **commit** B_{h-2}
16:   **terminate** VABA_{h-2} and FINAGR_{h-2}
17:   **extract** QC_{h-1} from B_h
18:   **if** VABA_{h-1} has been activated **then:**
19:     **activate** FINAGR_{h-1} with $(\texttt{OPT}, B_{h-1}, QC_{h-1})$ **if** has not
20:   **wait** until timer of 2Δ expires **or** B_{h+1} is received
21:   **if** B_{h+1} is not received **then:**
22:     $B'_h \leftarrow create\_block(h, b_h, QC_{h-1})$
23:     **activate** $\text{VABA}_h(B'_h)$

24: // Same as lines 24-35 of Algorithm 7

A.3 Safety analysis

For brevity, we use the term "pessimistic decision" to refer to the scenario where the block is committed at the end of the final agreement protocol and the block is a pessimistic candidate. Otherwise, we refer to it as an "optimistic decision". Concerning the safety analysis, there is no difference between chain-based ParBFT1 and chain-based ParBFT2. Therefore, the following analysis applies to both two variants. Furthermore, as the protocol instances for different epochs are independent, we can limit the safety analysis to a single epoch. The safety property is stated as Theorem 8, whose proof relies on two lemmas: Lemma 6 and Lemma 7.

LEMMA 6. *If a non-faulty replica commits B_h through an optimistic decision, then every non-faulty replica will also commit $B_k (k \leq h)$ through an optimistic decision.*

PROOF. We prove this by contradiction. Let us assume that there is a pessimistic decision of block B_k in replica p_i. This implies that at least $f + 1$ non-faulty replicas activate FINAGR_k with pessimistic candidates. These replicas must receive data from VABA_k before receiving B_{k+1}. We consider the following three situations:

- According to Lines 24-26 of Algorithm 7, these replicas will not vote for B_{k+1}. As a result, the leader of height $k + 2$ can neither collect $n - f$ votes for B_{k+1} nor create a valid B_{k+2}. Furthermore, any block with a height m ($m \geq k + 2$) cannot be created either, and any block with a height m ($m \geq k$) cannot be committed on the optimistic path.
- Since any block with a height m ($m \geq k + 2$) cannot be created, any $\text{FINAGR}_m(m \geq k+1)$ instance cannot be activated by an optimistic candidate. Therefore, any optimistic block with a height m ($m \geq k + 1$) cannot be committed at the end of the *prepare* phase or the end of the final agreement protocol.
- Regarding a height $m = k$, if it is committed at the end of the *prepare* phase, the block must be an optimistic candidate, and the final agreement protocol will also commit an optimistic candidate, which leads to a contradiction. If it is also committed at the end of the final agreement protocol, according to the agreement property of ABA, the block will also be a pessimistic candidate.

To sum up, if a non-faulty replica commits a pessimistic block at height k, no non-faulty replica can commit an optimistic block at height $m \geq k$. Therefore, Lemma 6 is proven.

□

LEMMA 7. *If a non-faulty replica exits the current epoch at height h, then every other non-faulty replica will also exit the current epoch at height h.*

PROOF. Assuming a non-faulty replica p_i exits the current epoch at height h, as per Lines 29-35 of Algorithm 7, p_i must commit an optimistic block for each height $m(m < h)$ and commit a pessimistic block at height h. On one hand, according to Lemma 6, each non-faulty replica will also commit an optimistic block for each height $m(m < h)$. Therefore, each non-faulty replica will exit at a height $k(k \geq h)$. On the other hand, according to the agreement property of ABA, each non-faulty replica will definitely commit a pessimistic block at height h. In other words, h is the lowest height where a pessimistic decision occurs, and each non-faulty replica will exit at height h.

□

THEOREM 8 (SAFETY). *If two non-faulty replicas commit blocks B and B' at height h respectively, then $B = B'$.*

PROOF. Without loss of generality, we assume that a non-faulty replica exits the current epoch at height h. As shown in Lemma 7, each non-faulty replica must also exit the current epoch at height h. Thus, for each height $k(k < h)$, every non-faulty replica will commit a block through the optimistic decision, and for height h, everyone will commit a block through the pessimistic decision. According to the safety property of the quorum mechanisms and the agreement property of ABA, all the blocks committed at the same height must be identical, which concludes the proof.

□

A.4 Liveness analysis

In contrast to the original definition in Section 2.1, we interpret the liveness property in chain-based ParBFT as Theorem 9. This theorem guarantees that if every non-faulty replica continues to enter the next epoch after exiting the previous epoch, blocks will be continuously committed, and the liveness is promised. The following analysis also applies to both two chain-based variants.

THEOREM 9 (LIVENESS). *At least one block can be committed in an epoch. Furthermore, for every height h in an epoch, either an optimistic block is committed at h or all non-faulty replicas exit at $k(k \leq h)$.*

PROOF. Lemma 7 implies that all non-faulty replicas will exit an epoch at the same height, denoted as h_e without loss of generality. At every height $h(h \leq h_e)$ in an epoch, either an optimistic block or a pessimistic block is committed, ensuring that at least one block can be committed in an epoch.

For a height h, if $h \geq h_e$, all non-faulty replicas exit at $k = h_e(k \leq h)$. On the other hand, if $h < h_e$, according to Lines 27-35 of Algorithm 7, an optimistic candidate block is committed at h. Thus, for every height h in an epoch, either an optimistic block is committed at h or all non-faulty replicas exit at $k(k \leq h)$. This concludes the proof. □