Sleepy Consensus in the Known Participation Model

Chenxu Wang
cxwang16117@gmail.com
Shandong University
China

Sisi Duan
duansisi@tsinghua.edu.cn
Tsinghua University
China

Minghui Xu
mhxu@sdu.edu.cn
Shandong University
China

Feng Li
fli@sdu.edu.cn
Shandong University
China

Xiuzhen Cheng
xzcheng@sdu.edu.cn
Shandong University
China

ABSTRACT

We study sleepy consensus in the known participation model, where replicas are aware of the minimum number of awake honest replicas. Compared to prior works that almost all assume the unknown participation model, we provide a fine-grained treatment of sleepy consensus in the known participation model and show some interesting results. First, we present a synchronous atomic broadcast protocol with $5\Delta + 2\delta$ expected latency and $2\Delta + 2\delta$ best-case latency, where Δ is the bound on network delay and δ is the actual network delay. In contrast, the best-known result in the unknown participation model (MMR, CCS 2023) achieves 14Δ latency, more than twice the latency of our protocol. Second, in the partially synchronous network (the value of Δ is unknown), we show that without changing the conventional $n \geq 3f + 1$ assumption, one can only obtain a secure sleepy consensus by making the stable storage assumption (where replicas need to store intermediate consensus parameters in stable storage). Finally, still in the partially synchronous network but not assuming stable storage, we prove the bounds on $n \geq 3f + 2s + 1$ without the global awake time (GAT) assumption (all honest replicas become awake after GAT) and $n \geq 3f + s + 1$ with the GAT assumption, where s is the maximum number of honest replicas that may become asleep simultaneously. Using these bounds, we transform HotStuff (PODC 2019) into a sleepy consensus protocol via a *timeoutQC* mechanism and a low-cost recovery protocol.

1 INTRODUCTION

Byzantine fault-tolerant state machine replication (BFT) is a fundamental tool in fault-tolerant distributed computing, allowing a group of replicas to reach an agreement in the presence of arbitrary failures [12, 18, 19, 33, 43–45]. Classic BFT protocols assume that replicas are aware of the identities of each other and non-faulty replicas are expected to always stay online. Recently, inspired by the emergence of Bitcoin [35], the *sleepy model* of consensus [38] is introduced. In sleepy consensus, certain number of replicas may unpredictably go offline (and become *asleep*) and later come back online (and become *awake*). The model is also known as the *crash-recovery* model in the distributed computing literature [11], characterizing the model where replicas keep infinitely often crashing and recovering. In this model, each replica is sometimes assumed to have a *stable storage* (or a *log*), which can be accessed even if the replica crashes and recovers later.

We provide a categorization of sleepy consensus in Table 1. Based on the timing assumptions, consensus protocols can be classified into three types: synchronous, partially synchronous, and asynchronous. In a synchronous network, there exists a *known* upper bound Δ for message transmission and processing. In a partially synchronous network [21], there exists such an upper bound but the value of Δ is unknown. In an asynchronous network, there does not exist an upper bound. Most sleepy consensus protocols known so far [16, 30, 31, 34, 38] consider a synchronous network and an unknown participation model, where replicas are not aware of the minimum number of awake honest replicas h_a. Pass and Shi [38] showed that sleepy consensus in the unknown participation model cannot be achieved in the partially synchronous model or asynchronous model. Momose and Ren (MR) [34] proposed the first constant-time sleepy consensus protocol. A follow-up work by Malkhi, Momose, and Ren (MMR) [31] further reduced the expected latency. While these works do not explicitly specify the requirement for stable storage, we believe most of them assume stable storage implicitly[1]. However, the recovery protocols introduce additional costs. In particular, the message delivery assumption [31] made by prior works specifies that "if an honest node p is awake at time t, then p has received all messages that were sent to it by honest nodes by time $t - \Delta$." In this case, any messages received right before a replica goes to sleep need to be stored locally for the replica to process after it becomes awake. Meanwhile, in the partially synchronous model, the Ebb-and-Flow protocol [36] and its follow-up work [37] mentioned that one can use conventional BFT protocols such as HotStuff [43] and PBFT [12] to obtain sleepy consensus. In this model, replicas always know the minimum number of awake honest replicas h_a. However, the concrete construction is not provided.

	Storage assumption	Sync. systems	Partially sync. systems
Unknown h_a	Not explicitly specified	[16, 30, 31, 34, 38]	Impossible. [38]
Known h_a	With stable storage	Not known yet	[36, 37]
	Without stable storage		Not known yet

Table 1: Overview of sleepy consensus.

[1]By using recovery protocols (e.g., the recovery mechanisms in MR and MMR), the assumption about stable storage can be removed.

Model	Protocol		Maximum number of asleep honest replicas	Known h_a	Expected latency	Stable storage?
Synchronous sleepy consensus ($n \geq 2f+1$)	MR [34] (without the recovery protocol)	safety	$n-2f-1$	✗	32Δ	✓
		liveness	$n-2f-1$			
	MMR [31]	safety	$n-2f-1^{\ddagger}$	✗	14Δ	✗
		liveness	$n-2f-1^{\ddagger}$			
	Koala-1 (Sec. 3)	safety	$n-f$	✓	$5\Delta+2\delta$	✗
		liveness	$n-2f-1$			
Partially synchronous sleepy consensus ($n \geq 3f+1$)	Ebb-and-flow* [36] **with stable storage (Sec. 4)**	safety	$n-f$	✓	7Δ (based on HotStuff)	✓
		liveness	$n-f$ †			
	Koala-2 (Sec. 5)	safety	$n-f$	✓	7Δ (based on HotStuff)	✗
		liveness	$n-3f-1^{\dagger}$ or $\lfloor \frac{n-3f-1}{2} \rfloor$			

Table 2: Comparison of sleepy consensus protocols. ‡The bound considers the worst case where all f Byzantine replicas remain awake. †Liveness of the protocol is guaranteed under the global awake time (GAT) assumption. *While Ebb-and-flow does not specify whether stable storage is required, we show that the result can only be achieved by assuming stable storage.

While sleepy consensus in the unknown participation model is a *better fit* for systems such as Bitcoin, sleepy consensus in the known participation model (where h_a is known) is of independent interest. For instance, systems using Proof-of-Stake (PoS) protocols [10, 22, 27] (i.e., Ethereum 2.0 [42], Polkadot [41]) or conventional BFT protocols (i.e., permissioned blockchains [7]) all assume that replicas have an agreement on the total number of replicas. Some protocols allow dynamic participation where the number of replicas may change over time [10, 20], but replicas always know the number of replicas of the system. However, sleepy consensus in the known participation model has not been well studied.

Therefore, an interesting open problem of sleepy consensus is:

Can we provide a more fine-grained treatment of sleepy consensus especially in the known participation model?

In this paper, we study sleepy consensus with known h_a in both synchronous and partially synchronous networks, with and without stable storage. As summarized in Table 2, we provide the following results.

Koala-1: faster synchronous sleepy consensus. We show that in the known participation model, the latency of synchronous sleepy consensus can be made closer to conventional consensus protocols. In particular, consider a system with n replicas among which at most f are faulty, our protocol requires that at least $h_a = f+1$ honest replicas are awake at any point of the protocol execution (but the set of honest replicas may differ from time to time). With this assumption, we construct a synchronous sleepy consensus protocol, Koala-1, with an expected latency of $5\Delta + 2\delta$ and a best-case latency of $2\Delta+2\delta$. In contrast, the best result so far in the unknown participation model has a latency of 14Δ (i.e., MMR [31]), more than twice the latency of Koala-1. The result is closer to the $2\Delta + \delta$ latency achieved by conventional synchronous BFT protocols [4] (not in the sleepy model).

We achieve our result via a *double confirmation mechanism* and a new *validated triple-graded proposal election (VT-GPE)* primitive, which might be of independent interest. The major challenge of synchronous sleepy consensus, even in the known participation model, is that one cannot use the *conventional* Byzantine quorum size of $\lceil \frac{n+f+1}{2} \rceil$ anymore. Consider a system in which more than a majority of honest replicas become asleep, using the conventional Byzantine quorum size easily breaks the liveness of the system. Our double confirmation mechanism makes it possible to use a Byzantine quorum with only h_a size. Meanwhile, a certificate with h_a matching votes becomes *transferrable*. We further utilize this nice property to build VT-GPE and the three grades implement a *commit-lock-prepare* process for atomic broadcast, a commonly implemented technique in conventional BFT [12, 43]. Additionally, Koala-1 also enjoys the benefit of pipelining. This allows our approach to further improve the throughput, while it is unclear how to do so for existing sleepy consensus protocols.

Partially synchronous sleepy consensus with stable storage. To date, the only known sleepy consensus in the partially synchronous model is proposed in the Ebb-and-Flow protocol [36] and its follow-up work [37]. It was briefly mentioned that one can use conventional BFT such as HotStuff [43], PBFT [12], and Streamlet [13] to directly obtain a sleepy consensus assuming global awake time (GAT) (after GAT, every honest replica becomes awake). Same as conventional BFT protocols in the partially synchronous model, a system with n replicas tolerates $f = \lfloor \frac{n-1}{3} \rfloor$ failures (i.e., $n \geq 3f+1$). Unfortunately, the concrete construction is not provided.

In this work, we show that by assuming the conventional $n \geq 3f+1$ bound, a partially synchronous sleepy consensus protocol can only be achieved by making the stable storage assumption. Furthermore, one can not directly use a known BFT to obtain a sleepy consensus protocol without explicitly specifying *which* intermediate consensus parameters are stored in stable storage. In particular, we show some corner cases where by storing *no* intermediate parameters of the protocol in the stable storage, the protocol can be easily broken in the sleepy model. While storing all the intermediate parameters in stable storage is an option, it is usually

very expensive to do so as frequent disk I/O is involved [9, 11, 17]. For instance, when implementing BFT-SMaRt [1] as a consistent key-value store where consensus data are all stored in stable storage, the throughput is only 1% of its storage-free counterpart. Even with fast stable storage such as SSDs, the throughput of the system is only 23% of its storage-free counterpart [9].

We show that instead of storing all intermediate parameters, we only need to store the view number and the *lockedQC* to make HotStuff a sleepy consensus in this model.

Koala-2: partially synchronous sleepy consensus without stable storage. In the partially synchronous model without assuming stable storage, we assume $h_a = \lceil \frac{n+f+1}{2} \rceil$, where n is the number of replicas and f is the number of faulty replicas. We show that $s = \lfloor \frac{n-3f-1}{2} \rfloor$ is a tight bound for sleepy consensus without assuming stable storage, where s is the maximum number of honest replicas that may become asleep at the same time. Rephrasing the bound, we have $n \geq 3f + 2s + 1$. Additionally, by assuming the existence of global awake time (GAT), the bound on s can be further improved to $n - 3f - 1$ (i.e. $h_a = 2f + 1$ and $n \geq 3f + s + 1$).

We transform HotStuff into a sleepy consensus protocol called Koala-2 that retains the 7δ latency. To build Koala-2, we provide a *timeoutQC* mechanism and an efficient recovery protocol for sleeping replicas to restore their states after recovering.

Our contributions. In summary, our work makes the following contributions:

- (Sec. 3) In the synchronous model, we provide Koala-1, a sleepy consensus protocol with an expected latency of $5\Delta + 2\delta$. In contrast, the latency of the best result known so far in the unknown participant model is more than twice our result. Our protocol can be further used in the pipelining mode to improve the throughput, while it is unclear how to do so in prior works.
- (Sec. 4) In the partially synchronous model, we conclude that sleepy consensus with the $n \geq 3f+1$ bound can only be achieved by assuming stable storage. We further show that we only need to store view number and *lockedQC* in stable storage to transform HotStuff into sleepy consensus under the GAT assumption.
- (Sec. 5) In the partially synchronous model, we propose Koala-2, a sleepy consensus protocol without the assumption of stable storage. We show the bound $s = \lfloor \frac{n-3f-1}{2} \rfloor$ without the GAT assumption and $s = n-3f-1$ with the GAT assumption. We transform HotStuff into sleepy consensus in this model that achieves the same expected latency as that for HotStuff.

2 SYSTEM MODEL AND BUILDING BLOCKS

Byzantine fault tolerance (BFT). In a BFT protocol, clients *submit* transactions (requests) and replicas *deliver* them. The client obtains a final response to the submitted transaction from the responses. Within a BFT system of n replicas, a maximum of f replicas may fail arbitrarily under the control of an adversary. These faulty replicas are also known as Byzantine failures and non-Byzantine replicas are called *honest* replicas. The correctness of a BFT protocol (under the sleepy model) is specified as follows:

- **Safety**: If an honest replica *delivers* a transaction tx before *delivering* tx', then no honest replica *delivers* the transaction tx' without first *delivering* tx.
- **Liveness**: If a transaction tx is *submitted* to all honest replicas, then all awake honest replicas eventually *deliver* tx.

An equivalent primitive atomic broadcast (ABC) is often used interchangeably with BFT. Atomic broadcast is only syntactically different from BFT. In atomic broadcast, a replica *a-broadcasts* messages and all replicas *a-deliver* messages.

- **Safety**: If an honest replica *a-delivers* a message m before it *a-delivers* m', then no honest replica *a-delivers* the message m' without first *a-delivering* m.
- **Livenss**: If an honest replica *a-broadcasts* a message m, then all awake honest replicas eventually *a-deliver* m.

While the BFT and atomic broadcast abstractions do not expose the *order* to the API, an implicit order is given in most protocols, e.g., sequence number [12, 20], height [39, 43]. Utilizing this implicit order, many partially synchronous protocols achieve a weaker safety property as follows [12, 39, 43].

- **Consistency**: If an honest replica *delivers* a transaction tx and another honest replica *delivers* a transaction tx', both with the same order, $tx = tx'$.

Our Koala-1 protocol follows the conventional atomic broadcast model. Our Koala-2 protocol achieves the consistency property, following that of HotStuff.

Network models and communication channels. We consider both synchronous and partially synchronous networks. In the synchronous model, there exists an upper bound Δ for message processing and transmission latency. We additionally assume a completely synchronous clock, i.e., replicas have access to a common global clock. In the partially synchronous model [21], there still exists an upper bound but the value of Δ is unknown. An alternative notion of the partially synchronous model is that there exists an unknown global stabilization time (GST) such that after GST, messages sent between two honest replicas arrive within a fixed delay.

We assume authenticated channels for message transmission. We use the symbol $*$ to denote any value. We use δ to denote the actual network latency.

The sleepy model. The notion of the sleepy model was first introduced by Pass and Shi [38]. A sleepy replica can be either *awake* or *asleep*. An awake replica actively participates in the execution, while an asleep replica does not execute any code of the protocol or send/receive any message. In our system, each honest replica can become asleep, whose status can change at any time under the control of an adversary, without any advance notice. In practice, this implies that replicas are allowed to leave and rejoin the protocol's execution at will without notifying other replicas. In the distributed computing literature [11], the sleepy model aligns with the crash-recovery model, where replicas can keep crashing and recovering repeatedly. It is notably highlighted that an honest replica might encounter "amnesia" after crashing, leading to the loss of its internal state stored in its volatile storage.

Our work considers the known participation model, where all replicas have foreknowledge of the minimum number of awake honest replicas h_a. Meanwhile, we use s to denote the maximum number of asleep replicas at any point of the protocol execution. In our synchronous sleepy consensus protocol, h_a is $f + 1$. In our

partially synchronous consensus protocol, h_a is $\lceil \frac{n+f+1}{2} \rceil$. If we consider the global awake time (GAT) assumption, where after GAT every sleeping replica will be awake, our partially synchronous protocol can be achieved with $h_a = 2f + 1$.

Cryptographic assumptions. We make use of digital signatures with a public-key infrastructure (PKI). We use $\langle \mu \rangle_i$ to denote a message μ signed by replica p_i. We assume a cryptographic collision-resistant hash function denoted as $\mathrm{H}(\cdot)$.

We also assume a verifiable random function (VRF) in one of our protocols. A replica p_i evaluates $(\rho_i, \pi_i) \leftarrow \mathrm{VRF}_i(\mu)$ on any input μ and obtain a pseudorandom value ρ_i and a proof π_i. Using π_i and the public key of replica p_i, anyone can verify whether ρ_i is a correct evaluation of VRF_i on input μ.

Blocks. We use *block* B to denote a batch of transactions. Blocks are ordered in a chain where the previous block of B is called its *parent block*. The first block in the chain is called the genesis block B_0. A block B extends block B' if B' is an ancestor of B in the chain. Two blocks B and B' conflict with each other if neither of them extends the other.

Byzantine quorums and quorum certificates. A *byzantine quorum* (or *quorum* in short) denotes a specific number of replicas. Matching *votes* from a quorum is necessary for honest replicas to reach an agreement. A set of signatures signed by a quorum of replicas is called a *quorum certificate* (QC or *certificate* in short). In conventional BFT systems with $n \geq 3f + 1$ replicas, a Byzantine quorum consists of $\lceil \frac{n+f+1}{2} \rceil$ replicas.

By slightly abusing notation, we use $view()$ function to denote the view number of a QC or a block. For example, if qc is a QC for block B, $view(qc)$=$view(B)$.

Graded proposal election (GPE). The notion of GPE is introduced by MMR [31]. In GPE, each replica *gpe-proposes* a block and *gpe-decides* either (B, g) or $\perp$, where B is a block and $g \in \{0, 1\}$ is the grade. GPE achieves the following properties:

- **Consistency.** If an honest replica *gpe-decides* $(B, *)$ and another honest replica *gpe-decides* $(B', *)$, $B = B'$.
- **Graded delivery.** If an honest replica *gpe-decides* $(B, 1)$, all honest replicas *gpe-decide* $(B, *)$.
- $1/2$**-validity.** With a probability of at least $1/2$, all honest replicas *gpe-decide* $(B, 1)$, where B has been *gpe-proposed* by an honest replica.

3 KOALA-1: FAST SYNCHRONOUS SLEEPY CONSENSUS

In this section, we introduce a synchronous sleepy consensus atomic broadcast protocol called Koala-1. We consider a system with $n \geq 2f + s + 1$ replicas and $h_a = f + 1$. We assume all f faulty replicas are always awake, following the assumption made by prior works [23, 34, 38]. Without loss of generality, we assume stable storage and message delivery, i.e., once a replica becomes awake at time t, it will immediately receive all messages sent to it by any honest replica before time $t - \Delta$. Later in Appendix C, we provide a practical recovery protocol to remove both assumptions.

We build a practical sleepy consensus in the known participation model with latency close to conventional synchronous BFT protocols (e.g., Sync HotStuff [3] has $2\Delta + \delta$ latency). In particular, Koala-1 has a fast path that achieves $2\Delta + 2\delta$ latency, which occurs when all awake replicas are honest. Meanwhile, the expected latency of Koala-1 is $5\Delta + 2\delta$. In contrast, the state-of-the-art sleepy consensus protocol MMR [31] has a latency of 14Δ, more than twice the latency of Koala-1. Besides, Koala-1 enjoys the pipelining mode, where replicas can start to process a new block before an agreement is reached for the current block. A comparison of Koala-1 with current sleepy consensus protocols is provided in Table 3.

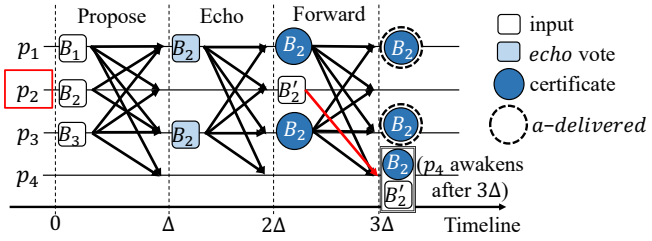
Protocol	Model	Known h_a	Expected latency	Best-case latency
Sync HotStuff [3]	Static participation model	✓	$2\Delta + \delta$	$2\Delta + \delta$
MR [34]	Sleepy model	✗	32Δ	16Δ
MMR [31]	Sleepy model	✗	14Δ	4Δ
Koala-1	Sleepy model	✓	$5\Delta + 2\delta$	$2\Delta + 2\delta$

Table 3: Comparison of synchronous atomic broadcast protocols. Δ is the upper bound on message processing and transmission latency and δ is the actual network latency.

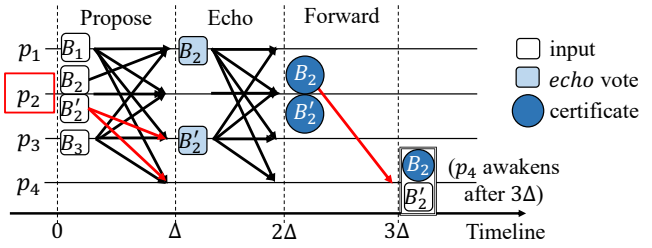
3.1 Overview of Koala-1

In the classic static participation model, one can obtain a synchronous atomic broadcast protocol assuming $n = 2f + 1$ and a quorum size of $f + 1$. Indeed, just as mentioned by MMR, $f + 1$ matching votes form a quorum certificate and the certificate is *transferrable*, i.e., it can be verified by any replicas. Together with an *equivocation detection* mechanism that detects whether a replica sends inconsistent messages to different replicas, one can obtain a secure synchronous GPE protocol and use the GPE protocol to build an atomic broadcast (ABC) protocol.

Specifically, the GPE protocol works as follows, considering each replica *gpe-proposes* the block it *a-broadcasts*. The protocol begins with a round where each replica broadcasts the block it *gpe-proposes* and a *leader election* mechanism is often embedded in this round. Assuming that all honest replicas agree on the identity of a common leader, each replica then *echoes* the block it receives from the leader to all replicas. Once receiving $f + 1$ echoed block B, each replica forwards the certificate to all replicas in the third round. If a replica detects an equivocation of the leader, it forwards the equivocating messages to all replicas. If a valid certificate for B is received in the second round and no equivocation has been detected by the end of the third round, a replica *gpe-decides* B with grade 1. If no equivocation is detected in the second round and a valid certificate for B has been received by the end of the third round, any replica that has not *gpe-decided* yet *gpe-decides* B with grade 0. When a replica *gpe-decides* a block B with grade 1, it *a-delivers* B in the ABC protocol. Prior synchronous Byzantine agreement protocols and atomic broadcast protocols roughly follow this paradigm as well [2–4]. Here, the grades are very close to the *commit-lock* relation in conventional BFT. Namely, grade 0 for block B can be viewed as a *lock* for B and honest replicas will never vote for another block *conflicting* with B. Meanwhile, if a replica *gpe-decides* B with grade 1, the block is *committed*. In some cases, each replica



(a) Scenario 1: p_2 sends B_2 to p_1 and p_3, and sends B_2' to p_4. p_1 and p_3 echo B_2, collect a certificate, and *a-deliver* B_2.



(b) Scenario 2: p_2 sends B_2 to p_1 and B_2' to p_3 and p_4. p_1 echoes B_2 and p_3 echoes B_2'. None of p_1 or p_3 *a-deliver* any block.

Figure 1: Two situations of synchronous atomic broadcast protocols in the sleepy model. In both scenarios, p_2 is Byzantine, p_4 receives B_2' from p_2 and a valid certificate for B_2. In scenario 1, p_1 and p_3 *a-deliver* B_2. In contrast, in scenario 2, none of p_1 or p_3 *a-deliver* any block.

may also need to broadcast its locked block (possibly in another round) to ensure the correctness of the protocol.

In the sleepy model, an tempting solution is to simply change the size of the Byzantine quorum and *transform* the protocol mentioned above into a sleepy consensus protocol. Unfortunately, even under the known participation model, building a secure sleepy consensus protocol is far from trivial. This is mainly because we cannot use the *conventional* quorum size anymore as (possibly more than the majority of) honest replicas may become asleep. W.l.o.g., we assume $n = 2f+s+1$. If we use the *conventional* way to decide Byzantine quorum, the quorum size becomes $\lceil \frac{n+f+1}{2} \rceil = \lceil \frac{s+f}{2} \rceil + f + 1$. However, only at least $h_a = f+1$ honest replicas are awake at any point of the protocol execution. Using a quorum size of h_a may easily make the protocol suffer from safety issues and the certificate with h_a matching votes thus fails to be transferrable.

Note that even assuming that a certificate with h_a matching votes is transferrable (for example, under the help of a powerful equivocation detection mechanism), there might still be safety and liveness issues. We show two scenarios in Figure 1 that are indistinguishable for an honest replica p_4, but the status of other honest replicas are different. In both scenarios, p_2 is Byzantine and is the leader. In the first scenario shown in Figure 1a, the leader p_2 broadcasts its block B_2 to all replicas, and p_1 and p_3 echo B_2. By $t = 2\Delta$, p_1 and p_3 both collect a certificate for B_2 and forward the certificate to all replicas. By $t = 3\Delta$, both p_1 and p_3 *a-deliver* B_2, as no equivocation is detected. After $t = 3\Delta$, the sleeping replica

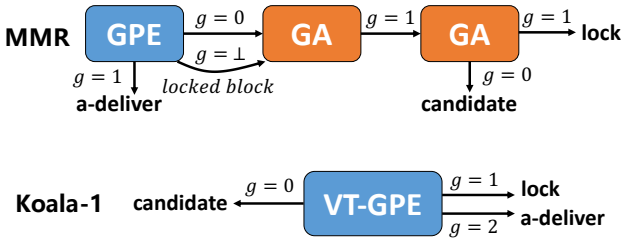


Figure 2: MMR [31] vs. Koala-1. GA denotes graded agreement.

p_4 wakes up and receives the certificate for B_2. Meanwhile, it also receives block B_2' from p_2, a conflicting block with B_2.

Figure 1b illustrates another scenario. Different from the first scenario, p_2 sends B_2 to p_1 and B_2' to p_3 and p_4. As p_2 receives echo messages for both B_2 and B_2', it holds two certificates, each with $f+1$ matching echo messages. It sends the certificate for B_2 to p_4. The two scenarios are indistinguishable for p_4. However, none of the honest replicas *a-deliver* any block in scenario 2. In contrast, in scenario 1, p_1 and p_3 *a-deliver* B_2, in which case p_4 is supposed to be locked on B_2.

h_a-enabled quorum. Our contribution is to use h_a as the quorum size and make the certificate with h_a matching votes *transferrable* and meanwhile build a secure protocol (to address the indistinguishability issue). This is achieved by a carefully designed *double confirmation mechanism* and an equivocation detection mechanism. The double confirmation mechanism requires every honest replica to vote for blocks that extend a block B only if there are *two* certificates for B, each with h_a matching votes. The two certificates are used as a *proof* for block B, making the status of honest replicas verifiable. Meanwhile, the equivocation detection mechanism ensures that no honest replicas will decide inconsistent values.

We slightly extend the GPE notion by MMR to validated triple-graded proposal election (VT-GPE). As the name suggests, VT-GPE has three grades instead of two in GPE. The three grades mimic the *commit-lock-prepare* relation of conventional BFT [12, 43]. Thanks to the transferability of the h_a-enabled quorum, our VT-GPE is *validated*: the input of each replica consists of both a block and the certificates that can be verified by all honest replicas. Such a design enjoys three immediate benefits. First, the protocol is much simplified compared to existing sleepy consensus protocols. As illustrated in Figure 2, we only need the VT-GPE primitive to build an atomic broadcast protocol and the latency of our protocol is close to conventional synchronous BFT. Second, our protocol enjoys the benefit of the pipelining mode to achieve higher throughput compared to the existing sleepy atomic broadcast protocol, while it is unclear how to do so in existing protocols. Finally, the recovery protocol (which can be used to remove the assumption about message delivery and stable storage) can be much simplified as well, as replicas only need to collect and verify the certificates instead of processing *all* missing messages.

3.2 Validated Triple-graded Proposal Election

Validated Triple-graded Proposal Election (VT-GPE). We define *validated triple-graded proposal election* (VT-GPE) as follows. Each honest replica *tgpe-proposes* a block (together with a valid proof) and *tgpe-decides* either (B, g, σ) (where $B \neq \perp$) or $\perp$. Here, g is a grade where $g \in \{0, 1, 2\}$. We also need an *external validity* property for VT-GPE to be validated. In particular, we define a global predicate that is determined by the particular application and known to all parties. In this work, we define σ as the proof for the validity of block B. Let the predicate be Q and we say B is validated by σ if $Q(B, \sigma)$ holds. Each honest replica only *tgpe-decides* one block in a VT-GPE instance, but it may *tgpe-decides* the same block multiple times with different grades. A validated VT-GPE protocol achieves the following properties:

- **External validity.** If an honest replica *tgpe-decides* $(B, *, *)$ such that $B \neq \perp$, $Q(B, \sigma)$ holds for at least one honest replica.
- **Consistency.** If an honest replica *tgpe-decides* $(B, *, *)$ and another honest replica *tgpe-decides* $(B', *, *)$, $B = B'$.
- **Graded delivery.** If an honest replica *tgpe-decides* $(B, g, *)$ such that $g \in \{1, 2\}$, any honest replica *tgpe-decides* $(B, g - 1, *)$.
- **Validity.** With a probability of $\alpha > 1/2$, all honest replicas *tgpe-decide* $(B, 2, *)$ where block B is *tgpe-proposed* by an honest replica.

The VT-GPE protocol. We use VT-GPE$_v$ to denote a VT-GPE instance. VT-GPE$_v$ consists of two phases: a VRF-based leader election phase and a graded consensus phase. The leader election phase selects a leader and honest replicas may select different leaders. The graded consensus phase allows replicas to converge on the result of the leader election. Our protocol is described in Algorithm 1.

The protocol begins with a VRF-based leader election. Each replica p_i broadcasts a $\langle \text{INPUT}, B_i, \sigma_i, \rho_i, \pi_i \rangle_i$ message, where B_i is the block p_i *tgpe-proposes*, σ_i is the proof for B_i, ρ_i is a VRF evaluation on the current view number, and π_i is a proof of the VRF evaluation. As defined above, every replica only considers B_i valid if $Q(B_i, \sigma_i)$ holds. For now we do not care about the instantiation of σ_i and later we will define it in our atomic broadcast protocol. The VRF evaluations are used for leader election. In particular, according to the VRF evaluations each replica receives, the producer of the highest VRF is considered the leader, and the corresponding $\langle \text{INPUT} \rangle$ message is defined as the *winning input*. If equivocating $\langle \text{INPUT} \rangle$ messages are received from the leader, the winning input is set as $\perp$. Additionally, the block B associated with the winning input is called the *winning block*.

As each replica may receive different sets of $\langle \text{INPUT} \rangle$ messages and the winning inputs might be different, we define it for each replica p_i. In particular, $\langle \text{INPUT}, B, \sigma, \rho, \pi \rangle_j$ from p_j is a winning input for p_i if the following conditions are met:

(1) $Q(B, \sigma)$ holds;
(2) π is a valid proof of ρ on the current view number;
(3) ρ is the highest among all the VRF evaluations in the $\langle \text{INPUT} \rangle$ messages;
(4) p_i has not received another valid $\langle \text{INPUT}, B', \sigma', \rho, \pi \rangle_j$ such that $B' \neq B$.

After the leader election, from time $t = \Delta$ to $t = 4\Delta$, the graded consensus phase is executed. The workflow is as follows.

- At $t = \Delta$, if replica p_i is awake, it forwards its winning input and broadcasts an $\langle \text{ECHO} \rangle$ message for the winning block.
- At $t = 2\Delta$, p_i broadcasts a $\langle \text{WINNER1} \rangle$ message containing its winning input. If p_i receives at least $f + 1$ matching $\langle \text{ECHO} \rangle$ messages for its winning block, the replica forwards these $\langle \text{ECHO} \rangle$ messages and broadcasts a $\langle \text{READY} \rangle$ message.
- At $t = 3\Delta$, p_i broadcasts a $\langle \text{WINNER2} \rangle$ message containing its winning input. Similar to the previous round, if p_i receives at least $f+1$ $\langle \text{READY} \rangle$ messages for its winning block, the replica forwards these $\langle \text{READY} \rangle$ messages and broadcasts a $\langle \text{LOCK} \rangle$ message.
- When $t \geq 4\Delta$, there are four conditions. First, if p_i receives $f + 1$ matching $\langle \text{LOCK} \rangle$ messages for its winning block B_j, it *tgpe-decides* B_j with grade 1 and uses $f + 1$ $\langle \text{LOCK} \rangle$ messages as the proof for B_j. Second, if p_i receives $f+1$ $\langle \text{READY} \rangle$ and $f+1$ $\langle \text{WINNER2} \rangle$ messages for any block B, it *tgpe-decides* B with grade 1. Here, both $f + 1$ $\langle \text{READY} \rangle$ messages and $f + 1$ $\langle \text{WINNER2} \rangle$ messages are used as proofs for B. Finally, if p_i receives $f + 1$ $\langle \text{ECHO} \rangle$ and $f + 1$ $\langle \text{WINNER1} \rangle$ messages for any block B, it *tgpe-decides* B with grade 0. Here, the $\langle \text{ECHO} \rangle$ messages and $\langle \text{WINNER1} \rangle$ messages are used as proofs for B. Otherwise, p_i *tgpe-decides* a special symbol $\perp$.

3.3 Atomic Broadcast (ABC)

Our ABC protocol follows the view-by-view construction of many classic BFT protocols [12, 39, 43] and also prior sleepy consensus protocols. In each view, each honest replica *a-broadcasts* a block and *a-delivers* at most one block.

The protocol starts from view 1 and the pseudocode for view v is shown in Algorithm 2. In each view v, there is one VT-GPE instance denoted as VT-GPE$_v$. In each VT-GPE$_v$, each replica p_i *tgpe-proposes* block B that extends its candidate, where B is the block p_i *a-broadcasts*. Recall that the idea is to use the grade $g \in \{2, 1, 0\}$ of VT-GPE to mimic the *commit-lock-prepare* relation in conventional BFT. To maintain the status, every replica maintains several local parameters, including the candidate and lock, which are initially set as the genesis block B_0. If a block B is *tgpe-decided* with grade 0 (resp. 1), the candidate (resp. lock) is set as B.

We define the global predicate Q for VT-GPE as follows. Given the value (B, qc) *tgpe-proposed* by any replica p_j, $Q(B, qc)$ holds at p_i if and only if:

- $view(B)$ equals the current view number of p_i, qc is a valid *prepareQC* for B, and the parent block of B is the block of qc;
- the view number of qc is at least the same as p_i's lock.

In our protocol, *prepareQC* is the *proof* each replica p_i holds after it *tgpe-decides* a block B with grade 0. According to our VT-GPE instantiation, the proof consists of two certificates, i.e., $f+1$ $\langle \text{ECHO} \rangle$ messages and $f + 1$ $\langle \text{WINNER1} \rangle$ messages for B. The certificates are crucial for B to be validated and we call them the double confirmation mechanism for B. Meanwhile, ensuring the view number of qc is at least the same as p_i's locked block further prevents forks from happening and is crucial for both safety and liveness.

Every replica p_i waits for the output of VT-GPE$_v$ and there are three possible outputs.

(1) If p_i *tgpe-decides* $(B, 0, (E(B), W_1(B)))$, p_i sets its candidate as B and *prepareQC* as $(E(B), W_1(B))$.

Algorithm 1 Validated Triple-graded Proposal Election of view v - VT-GPE$_v$.

Replica p_i executes the following algorithm at every time $t \geq 0$ after starting VT-GPE$_v$ in view v, and *tgpe-proposes* (B_i, σ_i) such that a global predicate $Q(B_i, \sigma_i)$ holds.
p_i maintains the following parameters for each received block B:
  $E(B) \leftarrow$ all received $\langle \text{ECHO}, B\rangle_*$ messages
  $R(B) \leftarrow$ all received $\langle \text{READY}, B\rangle_*$ messages
  $L(B) \leftarrow$ all received $\langle \text{LOCK}, B\rangle_*$ messages
  $W_1(B) \leftarrow$ all received $\langle \text{WINNER1}, \langle \text{INPUT}, B\rangle_*\rangle_*$ messages
  $W_2(B) \leftarrow$ all received $\langle \text{WINNER2}, \langle \text{INPUT}, B\rangle_*\rangle_*$ messages
if $t = 0$ **then**
  $(\rho_i, \pi_i) \leftarrow \text{VRF}_i(v)$
  broadcast $\langle \text{INPUT}, B_i, \sigma_i, \rho_i, \pi_i\rangle_i$
if $t = \Delta$ **then**
  if there exists a winning input $\langle \text{INPUT}, B_j, \sigma_j, \rho_j, \pi_j\rangle_j$ **then**
    forward the winning input (if not yet)
    if $Q(B_j, \sigma_j)$ holds **then**
      broadcast $\langle \text{ECHO}, B_j\rangle_i$
  else
    forward the equivocating INPUT messages by any replica
if $t = 2\Delta$ **then**
  update local winning input based on received $\langle \text{INPUT}\rangle$ messages
  if $\langle \text{INPUT}\rangle_j \neq \perp$ **then** // Let $\langle \text{INPUT}\rangle_j$ be the winning input
    broadcast $\langle \text{WINNER1}, \langle \text{INPUT}\rangle_j\rangle_i$
    if $|E(B_j)| \geq f + 1$ **then**
      broadcast $E(B_j)$ and $\langle \text{READY}, B_j\rangle_i$
  else
    forward the equivocating INPUT messages by any replica
if $t = 3\Delta$ **then**
  update local winning input based on received $\langle \text{INPUT}\rangle$ messages
  if $\langle \text{INPUT}\rangle_j \neq \perp$ **then** // Let $\langle \text{INPUT}\rangle_j$ be the winning input
    broadcast $\langle \text{WINNER2}, \langle \text{INPUT}\rangle_j\rangle_i$
    if $|R(B_j)| \geq f + 1$ **then**
      broadcast $R(B_j)$ and $\langle \text{LOCK}, B_j\rangle_i$
  else
    forward the equivocating INPUT messages by any replica
if $t \geq 4\Delta$ **then**
  update local winning input based on received $\langle \text{INPUT}\rangle$ messages
  if $\langle \text{INPUT}\rangle_j \neq \perp$ **and** $|L(B_j)| \geq f + 1$ **then** // Let $\langle \text{INPUT}\rangle_j$ be the winning input
    tgpe-decide $(B_j, 2, L(B_j))$
  if $|R(B)| \geq f + 1$ **and** $|W_2(B)| \geq f + 1$ for any block B **then**
    tgpe-decide $(B, 1, (R(B), W_2(B)))$
  if $|E(B)| \geq f + 1$ **and** $|W_1(B)| \geq f + 1$ for any block B **then**
    tgpe-decide $(B, 0, (E(B), W_1(B)))$
  if no block is *tgpe-decided* **then**
    tgpe-decide $\perp$

(2) If p_i *tgpe-decides* $(B, 1, (R(B), W_2(B)))$, it sets its lock as B and *lockedQC* as $(R(B), W_2(B))$. A valid *lockedQC* for block B consists of $f + 1$ $\langle \text{READY}\rangle$ and $f + 1$ $\langle \text{WINNER2}\rangle$ messages for B. The lock parameter is useful for defining the predicate Q and the *lockedQC* parameter is only useful in the recovery protocol (to be described in Appendix C).
(3) If p_i *tgpe-decides* $(B, 2, L(B))$, it *a-delivers* B and all the ancestors of B.

Pipelining mode. Our protocol enjoys the benefit of pipelining where replicas can enter the next view $v + 1$ at $t = 3\Delta$ of the current view v. While a new instance VT-GPE$_{v+1}$ is started, the current instance VT-GPE$_v$ still runs until each replica *tgpe-decides*. To see why replicas can enter the next view at $t = 3\Delta$, consider that an honest replica is locked on a block B in VT-GPE$_v$. All replicas awake at $t = 3\Delta$ must receive the *prepareQC* (including $f+1$ $\langle \text{ECHO}\rangle$ and $f + 1$ $\langle \text{WINNER1}\rangle$ messages) for B. Any honest replica that proposes new blocks must be able to extend B in newer views. Besides, as lock can be updated at $t = 4\Delta$ of view v, replicas can use their updated lock to verify the new blocks at $t = \Delta$ of view $v + 1$.

Fast path. Our protocol has a fast path that *a-delivers* a block in $2\Delta + 2\delta$ time. We achieve this by slightly modifying our VT-GPE primitive into a weaker version called wT-GPE. wT-GPE does not achieve the consistency property anymore and has a *weak consistency* property instead, defined as follows.

- *Weak consistency.* If an honest replica *tgpe-decides* $(B, g, *)$ with grade $g \geq 1$ and another honest replica *tgpe-decides* $(B', *, *)$, $B = B'$.

Compared to the consistency property achieved by VT-GPE, the weak consistency property achieves consistency only if an honest replica *tgpe-decides* a block with a grade of at least 1. Via this change of definition, we do not need the $\langle \text{WINNER1}\rangle$ and $\langle \text{WINNER2}\rangle$ messages in our wT-GPE construction anymore. As a result, each replica *tgpe-decides* a block B with grade 0 after it receives valid $E(B)$ at $t \geq 3\Delta$ and the sender of the $\langle \text{INPUT}\rangle$ for B has the highest VRF evaluation. Meanwhile, each replica *tgpe-decides* a block B with grade 1 or 2 after it receives valid $R(B)$ or $L(B)$ at time $t > 2\Delta$.

Although we do not need the $\langle \text{WINNER1}\rangle$ and $\langle \text{WINNER2}\rangle$ messages, our wT-GPE protocol still employs the double confirmation mechanism to make *prepareQC* consistent with *lockedQC* in each view. This is achieved by additionally modifying the predicate Q. In particular, upon receiving a valid *prepareQC* qc with $view(qc) = view(\text{lock})$, each replica additionally checks whether the block of the *prepareQC* is the same as its lock. In this way, only the *prepareQC* that matches the *lockedQC* will be verified by each honest replica.

As the workflow of the protocols is similar to those presented in this section, we show the pseudocodes of our wT-GPE protocol and our pipelined ABC protocol (with the fast path) in Appendix B.

3.4 Analysis

Why h_a-enabled quorum? The double confirmation mechanism we use ensures that a certificate with h_a matching messages is transferrable. In our VT-GPE construction, we use the double confirmation scheme for both grade 0 and grade 1. To *tgpe-decide* block B with grade 0, a replica needs to collect $f+1$ matching $\langle \text{ECHO}\rangle$ messages and $f+1$ matching $\langle \text{WINNER1}\rangle$ messages for B. Meanwhile, to *tgpe-decide* B with grade 1, a replica needs to collect $f+1$ matching $\langle \text{READY}\rangle$ messages and $f + 1$ matching $\langle \text{WINNER2}\rangle$ messages for B.

Using the two scenarios mentioned in Figure 1, we show that we can distinguish the two scenarios for p_4. Namely, based on the toy construction mentioned at the beginning of this section, we introduce one change: each replica additionally broadcasts a $\langle \text{WINNER1}\rangle$ message at $t = 2\Delta$ for the block from the leader. In scenario 1 (Figure 1a), p_1 and p_3 do not detect any equivocation, so they send $\langle \text{WINNER1}\rangle$ messages for block B_2 at $t = 2\Delta$. When p_4 wakes up

Algorithm 2 The Koala-1 atomic broadcast protocol. Code for p_i.

Initialize the following parameters
  $v \leftarrow 1$; candidate, lock $\leftarrow B_0$; *prepareQC*, *lockedQC* $\leftarrow \perp$.
  // *lockedQC* is used in the recovery protocol
Let Q be the following predicate for VT-GPE:
  Given (B, qc) *tgpe-proposed* by p_j, $Q(B, qc) \equiv (view(B) = v)$ **and**
  (qc is a valid *prepareQC*) **and** ($B.parent = qc.block$) **and**
  $view(qc) \geq view(\text{lock})$
In each view v, replica p_i executes the following algorithm at every time $0 \leq t \leq 4\Delta$ w.r.t. view v, and then enter the next view $v + 1$.
if $t = 0$ **then**
  $B \leftarrow \langle \text{vals}, \text{H}(\text{candidate}), v \rangle_i$
  tgpe-propose $(B, prepareQC)$ in VT-GPE$_v$ with predicate Q
// The following events may be triggered after view v
upon p_i *tgpe-decides* $(B, 0, (E(B), W_1(B)))$ in VT-GPE$_v$ **do**
  if $view(B) > view(\text{candidate})$ **then**
    candidate $\leftarrow B$, $prepareQC \leftarrow (E(B), W_1(B))$
upon p_i *tgpe-decides* $(B, 1, (R(B), W_2(B)))$ in VT-GPE$_v$ **do**
  if $view(B) > view(\text{lock})$ **then**
    lock $\leftarrow B$, $lockedQC \leftarrow (R(B), W_2(B))$
upon p_i *tgpe-decides* $(B, 2, L(B))$ in VT-GPE$_v$ **do**
  if B has not been *a-delivered* **then**
    a-deliver B and all the ancestors of B

after 3Δ, it receives the $\langle$WINNER1$\rangle$ messages due to the message delivery assumption. Therefore, p_4 can now *gpe-decide* B_2 with grade 0 according to the double confirmation mechanism. Now consider scenario 2 (Figure 1b), p_1 and p_3 detect the equivocation after receiving both B_2 and B_2', so none of them sends a $\langle$WINNER1$\rangle$ message. As p_4 has not received the $\langle$WINNER1$\rangle$ certificate, it does not *gpe-decide* B_2.

Latency. In the fast path, the best-case latency of our ABC protocol is $2\Delta + 2\delta$. Specifically, in the first two communication rounds of a wT-GPE instance, every replica needs to wait until the end of each Δ. This is mainly because each replica needs to wait for Δ time for the VRF evaluations from *all* honest replicas and another Δ to detect any equivocation between their winning inputs. In the last two rounds, each replica can enter the next phase as long as it receives a sufficiently large number of matching messages, so the latency is 2δ in total.

An *a-broadcast* block is expected to be *a-delivered* every *two* views, so the expected latency of our ABC protocol (pipelining mode) is $5\Delta + 2\delta$. This is because each replica enters the next view at as early as $t = 3\Delta$ of the current view and the block *a-broadcast* in the current view is *a-delivered* in the next view (after another $2\Delta + 2\delta$ time).

Communication complexity. Koala-1 achieves $O(\kappa n^3 + Ln^2)$ communication, where κ is the security parameter and L is the size of a block. The Ln^2 term is due to the VRF leader election phase where each replica broadcasts a block. The κn^3 term is because replicas forward the $\langle$ECHO$\rangle$ and $\langle$READY$\rangle$ messages. The communication can be reduced to $O(Ln^2 + \kappa n^2)$ using threshold signatures as replicas are aware of h_a in the known participation model.

We show the correctness of Koala-1 in Appendix A.

4 PARTIALLY SYNCHRONOUS SLEEPY CONSENSUS WITH STABLE STORAGE

In this section, we study partially synchronous sleepy consensus assuming the existence of stable storage. As mentioned in the introduction, Ebb-and-Flow briefly mentions that by assuming GAT, one can directly obtain a sleepy consensus using a conventional BFT [12, 13, 43]. Combined with the partially synchronous assumption, conventional BFT protocols can be always safe in the sleepy model and live after both GAT and GST.

We show that the above statement can be achieved *only* if stable storage is assumed and *intermediate* consensus parameters are stored in stable storage. To date, most BFT protocols known so far do not explicitly discuss what should be stored in stable storage as it is usually out of the scope of the consensus problem. We show that without explicitly storing the intermediate parameters, conventional BFT may not be safe and live in the sleepy model while retaining the $n \geq 3f + 1$ assumption, even assuming both GST and GAT. Intuitively, this is because if an honest replica does not persist its intermediate status during the protocol, its *status* might not be resumed after it sleeps and later becomes awake. Even if the replica synchronizes with all honest replicas after it becomes awake, the protocol may still not be correct.

In this section, we use HotStuff as an example and show an attack on safety without assuming stable storage. We then show that while one can simply transform conventional BFT to sleepy consensus by asking each replica to store *every* intermediate parameter in stable storage, we can provide a cheaper approach by storing only two parameters in stable storage. Indeed, as studied in many prior works [9, 11, 17], if all intermediate parameters are stored in stable storage, even assuming fast storage such as SSDs, the performance of the protocol is significantly degraded.

4.1 Overview of HotStuff

HotStuff operates in a view-by-view manner. We use the syntax of BFT to describe HotStuff and the protocol achieves consistency and liveness properties as defined in Sec. 2. In HotStuff, all replicas agree on a unique leader in each view. To reach an agreement on a block, each view consists of the following phases:

- **Prepare.** The leader p_k proposes a block B by extending the block of the highest received *prepareQC*, where a *prepareQC* is a set of $n - f$ $\langle$PREPARE$\rangle$ messages received in the "prepare" step of a previous view. Once receiving a valid proposal B from p_k, a replica casts a $\langle$PREPARE$\rangle$ vote for B and sends the vote to p_k. A collection of $n - f$ $\langle$PREPARE$\rangle$ votes forms a *qc* denoted as *prepareQC*.
- **Pre-commit.** After collecting a *prepareQC* for B, p_k broadcasts the *prepareQC* to all replicas. Upon receiving a valid *prepareQC* for B, a replica casts a $\langle$PRE-COMMIT$\rangle$ vote for B and sends it to p_k. Similarly, a collection of $n - f$ $\langle$PRE-COMMIT$\rangle$ votes forms a *precommitQC*.
- **Commit.** The leader p_k broadcasts the *precommitQC* to all replicas once it is available. After receiving a *precommitQC* for B, a replica becomes *locked* on B (the replica sets *lockedQC* as the *precommitQC*). Each replica then casts a $\langle$COMMIT$\rangle$ vote for B and sends the vote to p_k. A collection of $n - f$ $\langle$COMMIT$\rangle$ votes forms a certificate *commitQC*.

- **Decide.** Once collecting a *commitQC* for B, p_k sends the *commitQC* to all replicas, after which each replica delivers block B.
- **Advance to the next view.** Before entering the next view, a replica sends its *prepareQC* via a ⟨NEW-VIEW⟩ message to the next leader (which does not necessarily change in every view).

If a replica is locked on a block B in a view v, the replica only votes for blocks that extend B in subsequent views. A replica may become unlocked on B after it learns that $n - f$ replicas are not locked on B. In particular, a *prepareQC* for a conflicting block with a higher view number than B serves as proof for the replica to become unlocked on B.

HotStuff can utilize the pipelining feature to enhance its performance, which is also known as *chained* HotStuff. In particular, the view is changed in every PREPARE phase, so there is only one generic phase. The ⟨PREPARE⟩ vote on every proposed block B is simultaneously a ⟨PRE-COMMIT⟩ vote for the parent block of B and a ⟨COMMIT⟩ vote for the grandparent of B.

4.2 An Attack to HotStuff in the Sleepy Model without the Stable Storage Assumption

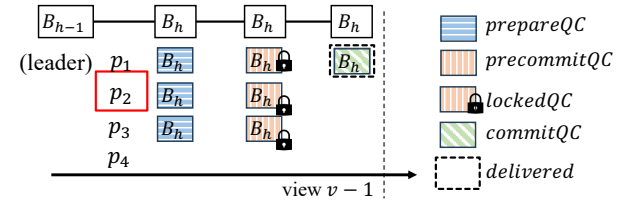
We show our attack in Figure 3 with four replicas among which p_2 is faulty. In the period of asynchrony, we consider that an adversary (i.e., a network scheduler) *manipulates* the network, the same as the assumption made by asynchronous protocols [18, 19, 33]. Note that in a partially synchronous network, we can assume the existence of a network scheduler during the asynchronous period. However, the network becomes synchronous after GST. Additionally, the adversary *controls* the replicas that may become asleep. In this case, the asleep replicas are still honest but just cannot process any messages when they sleep. Under these assumptions, the attack proceeds as follows.

In view $v - 1$, as shown in Figure 3a, p_1 is the leader and it proposes block B_h. After p_1 collects a *commitQC*, it delivers block B_h and replicas p_1, p_2, and p_3 become locked on B_h. Here, the network scheduler delays the messages received by p_4. Therefore, although p_4 is honest, it has not received any messages for B_h. After that, p_3 becomes asleep.

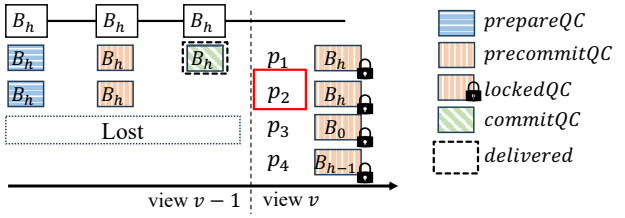
As shown in Figure 3b, replicas then enter view v and p_2 becomes the leader. Then replica p_3 becomes awake in view v. As p_3 does not have stable storage, it *loses* its *lockedQC*. As a result, *lockedQC* is set as the genesis block B_0. In view v, the leader p_2 is faulty and proposes a new block B'_h that extends B_{h-1} (the parent block of B_{h-1} is B_h). As B'_h is conflicting with B_h, replica p_1 considers the proposal B'_h invalid and will not vote for B'_h. However, p_2, p_3, and p_4 can vote for B'_h, as p_2 is faulty and the *lockedQC* of p_3 and p_4 is not conflicting with B'_h.

Finally, as illustrated in Figure 3c, replica p_1 delivers block B_h and replicas p_3 and p_4 deliver block B'_h where B'_h and B_h are conflicting, violating the safety property of the protocol.

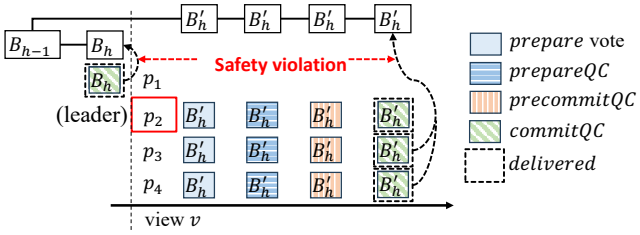
Remark 1. We assume that the adversary manipulates the network and the replicas that go to sleep. In fact, even if the adversary does not manipulate the network and the replicas that go to sleep, the scenarios may still happen, e.g., during network asynchrony or server crash.



(a) The status of replicas in view $v - 1$.



(b) p_3 goes to sleep near the end of view $v-1$ and becomes awake at the beginning of view v. It *loses* its *lockedQC* and sets its *lockedQC* as the genesis block B_0.



(c) The faulty leader p_2 creates a fork that extends B_{h-1} in view v and is able to collect a QC with votes from p_2, p_3, and p_4. Safety is violated as p_1 delivers block B_h and p_3 and p_4 deliver block B'_h.

Figure 3: An attack to HotStuff in the sleepy model assuming the intermediate consensus parameters are not stored in stable storage.

Remark 2. Although we present a concrete example using HotStuff, almost *all* partially synchronous BFT protocols utilize a variant of *commit-lock-prepare* paradigm [12, 13, 39]. Our attack is thus generic to almost all (if not all) partially synchronous BFT. We omit the generalization of the attack in our paper.

4.3 A Fully-fledged Sleepy Consensus Protocol under the Stable Storage Assumption

According to the attack mentioned above, conventional BFT protocols under the standard $n \geq 3f + 1$ assumption can be made correct only under the assumption of stable storage. However, storing *all* intermediate consensus parameters in stable storage significantly degrades the system performance. Therefore, an interesting research question to answer is:

Under the $n \geq 3f + 1$ and stable storage assumption, can we transform a conventional BFT protocol to a sleepy consensus protocol and meanwhile store minimum intermediate consensus parameters in stable storage?

We use HotStuff as an example and show that the minimum requirement for stable storage is the view number and the *lockedQC*.

Namely, if the current view number is lost when an honest replica falls asleep, the replica can only *catch up* with other replicas to learn the latest view number after waking up. It is possible that the replica re-enters the same view before it fell asleep. In this case, the replica might vote for a conflicting block with the one it has voted for (before falling asleep). Thus, two conflicting *qc* could be generated in the same view, violating the safety property. To ensure safety within a view, the highest view v when a replica has cast a vote should be stored in stable storage.

Meanwhile, the attack described in Figure 3 shows that the safety across views might be violated if a replica loses its *lockedQC*. We show that storing *lockedQC* in stable storage is sufficient to ensure safety across views. In particular, if a block B is delivered, a quorum of replicas becomes locked on B. To deliver a block B' that is conflicting with B, at least one honest replica of the quorum must have voted for the B'. Since an honest replica already sets its *lockedQC* as B, it will never vote for a block conflicting with B. Consider the example mentioned in Figure 3. p_3 stores its *lockedQC* for block B_h in stable storage before it goes asleep. When p_3 becomes awake at the beginning of view v, it restores the *lockedQC* for B_h from its stable storage. In view v, the leader p_2 proposes a new block B'_h that extends B_{h-1} (the parent block of B_h). As B'_h is conflicting with B_h, replica p_1 and p_3 do not consider the proposal B'_h valid and will not vote for B'_h. Thus only p_2 and p_4 can vote for B'_h. *prepareQC*, *precommitQC* or *commitQC* cannot be formed for B'_h, so no honest replicas will deliver block B'_h.

BFT in production systems. We surveyed Diem BFT[2] (which implements HotStuff) and Tendermint[3] (a variant of PBFT). We found that both implementations store some consensus parameters in stable storage but in different ways. Diem stores almost all consensus parameters in stable storage, including proposed blocks, the last vote, quorum certificates, and the highest timeout certificate (persistent_liveness_storage.rs:24-62[4]). Interestingly enough, it was clearly mentioned that this is used to ensure liveness even if all replicas crash and later recover. As a result, Diem BFT achieves a throughput of around 1 ktx/s [40] while HotStuff claims to achieve a throughput of over 300 ktx/s. Meanwhile, Tendermint uses a *write-ahead log* mechanism to implement stable storage for consensus parameters. In particular, all consensus messages (timeouts, proposals, block part, and vote) are first pushed to a log and the log pushes the data to stable storage every two seconds (wal.go:28,138[5]). In this way, if some honest replicas sleep within the 2-second duration, the protocol might not be safe.

The survey results validate our analysis. Our conclusion is that one does not have to store *all* consensus parameters in stable storage to build a both safe and live protocol in the sleepy model.

5 KOALA-2: PARTIALLY SYNCHRONOUS SLEEPY CONSENSUS WITHOUT STABLE STORAGE

In this section, we study partially synchronous sleep consensus without the stable storage assumption. We show that $n \geq 3f+2s+1$ is required to guarantee safety and liveness, where s is the maximum number of honest replicas that may become asleep simultaneously. We show that the quorum size of $n-f-s$ is a lower bound for sleepy consensus without assuming stable storage.

Still using HotStuff as an example, we transform the protocol into a sleepy consensus protocol. The main workflow remains almost the same as in HotStuff. We only need to adjust the quorum size of the main protocol and modify the view change protocol (i.e., leader election) to incorporate a *timeoutQC* mechanism. Besides, we introduce a new recovery protocol for asleep replicas to catch up after they recover. Our transformation is generic and can be extended to other partially synchronous BFT. We show the transformation of PBFT to sleepy consensus in Appendix F.

5.1 Technical Overview

The lower bound of $n \geq 3f + 2s + 1$. Consider f failures and s sleeping replicas, a Byzantine quorum tolerating f failures is set as $\lceil \frac{n+f+1}{2} \rceil$, so two Byzantine quorums always overlap in at least one honest replica. To see why this is the case, every Byzantine quorum contains $\lceil \frac{n+f+1}{2} \rceil - f = \lceil \frac{n-f+1}{2} \rceil$ honest replicas. Two disjoint Byzantine quorums thus would have at least $n-f+1$ honest replicas. As there are $n-f$ honest replicas in total, there is at least one overlapped honest replica in both Byzantine quorums.

Now we consider liveness and responsiveness (the protocol makes progress after collecting messages from a quorum of replicas). As $f+s$ replicas may not respond, the number of awake honest replicas in the system must be equal to or greater than the quorum size. This condition is satisfied only when $n-f-s \geq \lceil \frac{n+f+1}{2} \rceil$. Accordingly, $n \geq 3f + 2s + 1$.

If we consider the GAT assumption, the bound can be lowered to $n \geq 3f+s+1$. In particular, let β_1 be the quorum size in the main protocol. We can modify the liveness requirement to $\beta_1 \leq n-f$, as eventually every honest replica can receive messages from all honest replicas (after GAT). However, without stable storage, a *recovering* replica needs to collect information from other awake replicas to restore local parameters. There is no guarantee that all honest replicas will respond even after GAT, as some of them may also be in the recovery status. As at most s honest replicas may fall asleep simultaneously, up to $f+s$ replicas might be unavailable in the system. To ensure that the system is live, we need $\beta_2 \leq n-f-s$, where β_2 denotes the number of messages a recovering replica needs to collect. Besides, to ensure safety, a recovering replica should receive messages from at least one honest replica of each quorum. Therefore, β_1 and β_2 should overlap in at least $f+1$ replicas, i.e., $\beta_1 + \beta_2 - n \geq f + 1$. Summarizing the results, the lower bound is then $n \geq 3f + s + 1$.

Overview of Koala-2. We now describe the Koala-2 protocol without the GAT assumption. According to the discussion above, we can simply change the quorum size of HotStuff from $n-f$ to $n-f-s$ to ensure the correctness of the protocol. However, we still need to ensure that any honest replica that just recovered will not vote for the *wrong* block so the security properties will not be violated. Therefore, a correct recovery protocol is all we need to show in Koala-2. However, achieving safety within a view and safety across views are still not trivial and we also modify the

[2] https://github.com/diem/diem
[3] https://github.com/tendermint/tendermint
[4] https://github.com/diem/diem/blob/main/consensus/src/persistent_liveness_storage.rs
[5] https://github.com/tendermint/tendermint/blob/main/consensus/wal.go

view change protocol to complete our transformation. Below, we describe the challenges and how we addressed them.

To ensure safety within a view, we only need to ensure that QCs for two conflicting blocks will not be formed. At a glance, this can already be achieved via the quorum intersection rules of Byzantine quorums. However, we still need to ensure that a recovering replica will not vote twice for conflicting blocks in the same view (one before it fell asleep and one after it recovers). Indeed, without the stable storage assumption, such a requirement can only be achieved by the recovery protocol. Accordingly, in our recovery protocol, each recovering replica needs to synchronize the latest view number with other replicas. We use a special type of QC called *timeoutQC* to realize this. Each *timeoutQC* is generated during the view changes. Using *timeoutQC*, a replica can confirm that the view number it obtains during the recovery process is at least the view number before it fell asleep.

To ensure safety across views, similar to conventional BFT protocol, we need to ensure that the delivered block remains delivered across views. In HotStuff, this is achieved by the *commit-lock-prepare* paradigm. Namely, if a block is delivered, at least a quorum of replicas become locked on this block and will not vote for any conflicting blocks. However, we also need to ensure that any honest replica of the quorum will not lose its status after becoming locked on the delivered block, for similar reasons as discussed in Sec. 4.2. To achieve this goal without the stable storage assumption, a recovering replica needs to synchronize the latest *lockedQC* with other awake replicas. By quorum intersection, at least one honest replica will send the *correct lockedQC* of the delivered block to the recovering replica. However, there is no guarantee in a partially synchronous environment that a recovering replica always obtains the latest *lockedQC*. This is mainly because the recovering replica might complete the recovery before other replicas of the quorum become locked on the delivered block.

We employ an *atomic QC acquiring* mechanism to address this issue. In our recovery protocol, a recovering replica can utilize the information obtained from the *timeoutQC* mechanism to confirm the *time* (view number to be concrete) the lost *lockedQC* has been stored by a sufficiently large number of honest replicas. Utilizing this information from the recovering replica, each awake replica thus refrains from sending its *lockedQC* to the recovering replica until it confirms its *lockedQC* is indeed the one needed by the recovering replica. In this way, the recovering replica always obtains a *correct lockedQC* so safety across views is achieved.

5.2 The Modified View Change Protocol and the Recovery Protocol

In this section, we present the modified view change protocol (Algorithm 3) and our new recovery protocol (Algorithm 4).

The modified view change protocol. The modified view change protocol is triggered when a timeout occurs during the normal case operation. When a replica p_i experiences a timeout within a view v, it stops the normal case operation and broadcasts a $\langle \text{TIMEOUT}, v \rangle_i$ message. A collection of $n - f - s$ matching $\langle \text{TIMEOUT} \rangle$ messages from different replicas forms a *timeoutQC*. After receiving a valid *timeoutQC* of view v, p_i proceeds to view v+1. To expedite the view change process, p_i can broadcast the $\langle \text{TIMEOUT}, v \rangle_i$ message once receiving $f + 1$ $\langle \text{TIMEOUT} \rangle$ messages of view v. When p_i receives the *timeoutQC* of view v, it forwards the *timeoutQC* to all replicas.

Algorithm 3 Modified view change protocol (for replica p_i).

```
1: Let curView be the current view number.
2: upon the timer of curView expires do
3:     broadcast ⟨TIMEOUT, curView⟩_i
4: upon receiving f + 1 ⟨TIMEOUT, curView⟩_* do
5:     stop the timer of curView and broadcast ⟨TIMEOUT, curView⟩_i
6: upon receiving n − f − s ⟨TIMEOUT, v′⟩_* such that v′ ≥ curView do
7:     timeoutQC ← the set of n − f − s ⟨TIMEOUT, v′⟩_*
8:     broadcast ⟨ADVANCE-VIEW, v′, timeoutQC⟩_i
9:     send ⟨NEW-VIEW, v′ + 1, prepareQC⟩_i to the leader of view v′ + 1
10:    curView ← v′ + 1
11: upon receiving a timeoutQC tc of a view v′ ≥ curView do
12:    timeoutQC ← tc
13:    broadcast ⟨ADVANCE-VIEW, v′, timeoutQC⟩_i
14:    send ⟨NEW-VIEW, v′ + 1, prepareQC⟩_i to the leader of view v′ + 1
15:    curView ← v′ + 1
```

The recovery protocol. The protocol proceeds as follows:

- **Obtaining** *timeoutQC*. A recovering replica p_i first broadcasts a $\langle \text{RECOVERY-1} \rangle$ message. Upon receiving the $\langle \text{RECOVERY-1} \rangle$ message, any awake replica will respond to p_i the latest *timeoutQC* (via a $\langle \text{ECHO-1} \rangle$ message). Once receiving $n - f - s$ valid *timeoutQC*, p_i selects the one with the highest view number v_h. Then p_i waits for a *timeoutQC* with a view number $v \geq v_h + 2$ before entering the next step.
- **Atomic QC acquiring mechanism.** After receiving a *timeoutQC* tc for a view $v \geq v_h + 2$, p_i sets its local *timeoutQC* as tc, and broadcasts a $\langle \text{RECOVERY-2}, tc \rangle_i$ message. Any awake replica that receives this message will first start the view change protocol and proceed to view $view(tc) + 1$ (if not yet). Then the replica sends to p_i a $\langle \text{ECHO-2}, curView, (prepareQC, lockedQC) \rangle$ message, where *curView* is the current view number. When p_i receives $n-f-s$ $\langle \text{ECHO-2} \rangle$ messages with view numbers higher than v_h+2, it sets its own *lockedQC* as the highest *lockedQC* among the messages, and sets its *prepareQC* as the highest *prepareQC*. After that, p_i sets the current view number as $view(timeoutQC) + 1$ and becomes awake.

5.3 Analysis

Sketch of correctness. While we prove the correctness in Appendix D, we sketch the correctness here. Our *timeoutQC* mechanism and the recovery protocol together achieve safety within a view and across views. The *timeoutQC* mechanism ensures that a recovering replica will not vote twice in the same view. In particular, a recovering replica p_i first obtains the highest *timeoutQC* from a quorum of awake replicas. Suppose the view number of the highest *timeoutQC* is v_h and p_i fell asleep in view v, our *timeoutQC* mechanism guarantees that $v_h + 2 \geq v$. This is because p_i must have received a *timeoutQC* for view $v - 1$ before entering view v. At that time, a quorum of replicas must have entered view $v - 1$. Due to the quorum intersection rules, at least one honest replica must have a *timeoutQC* of at least view $v - 2$ and send this *timeoutQC* to p_i. Thus, $v_h \geq v - 2$. In our recovery protocol, replicas start a view change so p_i always enters view $v' = v_h + 3$ after it recovers.

Algorithm 4 Recovery protocol for HotStuff (for replica p_i).

Let *curView* be the current view number.
as a recovering replica
  broadcast a $\langle \text{recovery-1} \rangle_i$ message
  wait for $n - f - s$ $\langle \text{ECHO-1}, timeoutQC \rangle_*$
  $v_h \leftarrow$ the view number of the highest *timeoutQC* among all received $\langle \text{ECHO-1} \rangle$ messages
  wait for a *timeoutQC* tc such that $view(tc) \geq v_h + 2$
  $timeoutQC \leftarrow tc$
  broadcast $\langle \text{RECOVERY-2}, timeoutQC \rangle_i$
  wait for $n - f - s$ $\langle \text{ECHO-2}, v', (prepareQC, lockedQC) \rangle_*$ such that $v' > v_h + 2$
  $lockedQC \leftarrow$ the *lockedQC* with the highest view number among received $\langle \text{ECHO-2} \rangle$ messages
  $prepareQC \leftarrow$ the *prepareQC* with the highest view number among received $\langle \text{ECHO-2} \rangle$ messages
  $curView \leftarrow view(timeoutQC) + 1$
  send $\langle \text{NEW-VIEW}, curView, prepareQC \rangle_i$ to the leader of *curView*
  set the state as *awake* and rejoin the main protocol's execution
as an awake replica
  upon receiving $\langle \text{RECOVERY-1} \rangle_j$ **do**
    send $\langle \text{ECHO-1}, timeoutQC \rangle_i$ to replica p_j
  upon receiving $\langle \text{RECOVERY-2}, timeoutQC \rangle_j$ **do**
    if $view(timeoutQC) \geq curView$ **then**
      start view change and proceed to view $view(timeoutQC)+1$
    send $\langle \text{ECHO-2}, curView, (prepareQC, lockedQC) \rangle_i$ to replica p_j

As $v_h \geq v - 2$, $v' > v$. Therefore, replicas will not vote twice in the same view and safety within a view can be achieved. Additionally, in our recovery protocol, every honest awake replica refrains from sending its latest *lockedQC* to p_i until it enters view $v_h + 3 \geq v + 1$. Thus, p_i can always obtain a *lockedQC* for a block that extends any already delivered block in view $v' \leq v$. This ensures that p_i restores a correct *lockedQC*, achieving safety across views.

Using the example in Figure 3, we show that the locked block p_i obtains during its recovery must extend any delivered block before p_i fell asleep. To meet the lower bound of $n \geq 3f+2s+1$, we assume there are two additional honest replicas p_5 and p_6 in the system. p_5 and p_6 remain awake in view $v - 1$ and v. p_5 becomes locked on B_h when B_h is delivered in view $v - 1$, while p_6 is still locked on B_{h-1}. When p_3 wakes up at the beginning of view v, according to the protocol, it collects the *lockedQC* from at least four replicas and selects the highest one as its *lockedQC*. Since at most three replicas (i.e., p_2, p_4 and p_6) might send a *lockedQC* for a block not extending B_h, p_3 must receive a *lockedQC* for B_h and then become locked on B_h after recovery.

Liveness of the protocol roughly follows that of HotStuff, as we only modify the quorum size for the normal case protocol. Since we set the quorum size as $n - f - s$, every replica is able to receive a quorum of votes in every step of the protocol. Meanwhile, our newly designed recovery protocol is non-blocking. In particular, a recovering replica first obtains the *timeoutQC* from a quorum of replicas, where the highest *timeoutQC* is for view v_h. After observing a *timeoutQC* for view $v_h + 2$, the recovering replica sends the *timeoutQC* to awake replicas. Awake replicas can immediately enter view $v_h + 3$ and respond with their *lockedQC*s and *prepareQC*s. Any honest awake replica will eventually complete the process and the recovering replica can always complete the recovery protocol. Additionally, as analyzed in our sketch for safety, the recovering replica can obtain a *prepareQC* no lower than its *lockedQC* before it fell asleep. Therefore, liveness can be achieved.

Communication complexity. The normal case protocol and the view change protocol of Koala-2 achieves $O(\kappa n^2)$ communication, where κ is the security parameter. The bottleneck is the modified view change protocol. Specifically, each replica broadcasts a $\langle \text{TIMEOUT} \rangle$ message that contains the current view number and a digital signature (each has κ length). The recovery protocol of Koala-2 achieves $O(\kappa n)$ communication for the recovering process of one replica. In particular, the recovering replica broadcasts its $\langle \text{RECOVERY-1} \rangle$ and $\langle \text{RECOVERY-2} \rangle$ messages (both of size $O(\kappa)$). Each awake replica will respond with its *timeoutQC*, *prepareQC*, and *lockedQC*. As our protocol is built in the known participation model, we can use threshold signatures to instantiate the QCs so each QC has κ length.

Koala-2 can be adapted to the GAT assumption with some minor modifications, which is presented in Appendix E.

6 ADDITIONAL RELATED WORK

Synchronous BFT. Many classic synchronous Byzantine agreement and Byzantine broadcast protocols aim to lower the expected latency or best-case latency [3–5]. In the sleepy model, a concurrent work by D'Amato and Zanolini [16] aims to achieve lower latency than MMR. This is achieved by introducing a new stable participation assumption where for every time period of $[t, t+2\Delta]$, the number of honest replicas remaining awake during the period exceeds the number of Byzantine replicas.

Other aspects of synchronous sleepy consensus have been studied. For example, Gafni and Losa [30] studied Byzantine agreement in the sleepy model that achieves constant latency. Meanwhile, a recent work by D'Amato, Losa, and Zanolini [15] studies asynchrony resilience for synchronous sleepy consensus. The idea is to make a synchronous protocol safe and live under intermittent asynchronous periods.

Diskless crash recovery. The sleepy model is also known as the *crash-recovery* model in the distributed computing literature [11]. In fact, consensus in the crash-recovery model for crash fault-tolerant protocols has been studied extensively [8, 25, 26]. Most protocols rely on the stable storage assumption. Protocols without the stable storage assumption are also known as protocols in the diskless crash recovery (DCR) model [32]. Aguilera, Chen, and Toueg [6] discuss under what conditions stable storage is necessary. Using failure detectors, they present two consensus protocols, one with stable storage and one without. Michael, Ports, Sharma, and Szekeres [32] provide a generic approach that transforms protocols in the crash-recovery model (with stable storage) to the DCR model. Kończak, et al. [28] propose two recovery algorithms for Paxos [29] to make Paxos correct in the DCR model. All these works consider benign crash failures. In contrast, our Koala-2 protocol can be considered the first BFT protocol in the DCR model.

Consensus with multiple failure types. Some protocols tolerate both Byzantine failures and crash failures [8, 14, 24]. Backes and Cachin [8] propose an asynchronous reliable broadcast protocol in a system with $n \geq 3f + 2s + 1$ replicas, where s is the

maximum number of crashed replicas. UpRight [14] implements a partially synchronous BFT-SMR protocol with the same bound. SBFT [24] provides a dual-mode partially synchronous BFT-SMR protocol for $n = 3f + 2c + 1$ replicas where c is the number of crashed or slow replicas.

7 CONCLUSION

We propose three results for sleepy consensus in the known participation model, where all awake replicas are aware of the minimum number of awake honest replicas. In the synchronous network, we provide an atomic broadcast protocol with a latency close to the state-of-the-art conventional synchronous protocols. Compared to existing sleepy consensus protocols in the unknown participation model, the latency of our approach is over 50% lower. In the partially synchronous network, we show that sleepy consensus retaining the conventional $n \geq 3f + 1$ bound can only be achieved by assuming stable storage. Without assuming stable storage, we prove the tight bounds of $n \geq 3f + 2s + 1$ without the global awake time (GAT) assumption and $n \geq 3f + s + 1$ with the GAT assumption, where s is the maximum number of honest replicas that may go to sleep simultaneously. We then provide a low-cost transformation of HotStuff in the sleepy model.

REFERENCES

[1] 2012. *BFT-SMaRt Project Page.* http://code.google.com/p/bftsmart

[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 *International Conference on Financial Cryptography and Data Security*. Springer, 320–334.

[3] Ittai Abraham, Dahlia Malkhi, Kartik Nayak, Ling Ren, and Maofan Yin. 2020. Sync HotStuff: Simple and Practical Synchronous State Machine Replication. In *IEEE Symposium on Security and Privacy (S&P)*. IEEE, 106–118.

[4] Ittai Abraham, Kartik Nayak, Ling Ren, and Zhuolun Xiang. 2020. Byzantine Agreement, Broadcast and State Machine Replication with Near-optimal Good-Case Latency. *arXiv preprint arXiv:2003.13155* (2020).

[5] Ittai Abraham, Kartik Nayak, Ling Ren, and Zhuolun Xiang. 2021. Good-case Latency of Byzantine Broadcast: A Complete Categorization. In *ACM Symposium on Principles of Distributed Computing (PODC)*. 331–341. https://doi.org/10.1145/3465084.3467899

[6] Marcos Kawazoe Aguilera, Wei Chen, and Sam Toueg. 2000. Failure detection and consensus in the crash-recovery model. *Distributed computing* 13 (2000), 99–125.

[7] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, Srinivasan Muralidharan, Chet Murthy, Binh Nguyen, Manish Sethi, Gari Singh, Keith Smith, Alessandro Sorniotti, Chrysoula Stathakopoulou, Marko Vukolić, Sharon Weed Cocco, and Jason Yellick. 2018. Hyperledger fabric: A distributed operating system for permissioned blockchains. *EuroSys*.

[8] Michael Backes and Christian Cachin. 2003. Reliable Broadcast in a Computational Hybrid Model with Byzantine Faults, Crashes, and Recoveries. In *DSN*, Vol. 3. 37–46.

[9] Alysson Bessani, Marcel Santos, João Felix, Nuno Neves, and Miguel Correia. 2013. On the Efficiency of Durable State Machine Replication. In *2013 USENIX Annual Technical Conference*. 169–180.

[10] Vitalik Buterin, Diego Hernandez, Thor Kamphefner, Khiem Pham, Zhi Qiao, Danny Ryan, Juhyeok Sin, Ying Wang, and Yan X Zhang. 2020. Combining GHOST and casper. *arXiv preprint arXiv:2003.03052* (2020).

[11] Christian Cachin, Rachid Guerraoui, and Luís Rodrigues. 2011. *Introduction to Reliable and Secure Distributed Programming*. Springer Science & Business Media.

[12] Miguel Castro, Barbara Liskov, and et al. 1999. Practical Byzantine Fault Tolerance. In *3rd Symposium on Operating Systems Design and Implementation (OSDI)*. USENIX, 173–186.

[13] Benjamin Y. Chan and Elaine Shi. 2020. Streamlet: Textbook Streamlined Blockchains. In *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies* (New York, NY, USA) *(AFT '20)*. Association for Computing Machinery, New York, NY, USA, 1–11. https://doi.org/10.1145/3419614.3423256

[14] Allen Clement, Manos Kapritsos, Sangmin Lee, Yang Wang, Lorenzo Alvisi, Mike Dahlin, and Taylor Riche. 2009. Upright cluster services. In *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles*. 277–290.

[15] Francesco D'Amato, Giuliano Losa, and Luca Zanolini. 2023. Improving Asynchrony Resilience in Dynamically Available Total-Order Broadcast Protocols. arXiv:2309.05347 [cs.DC]

[16] Francesco D'Amato and Luca Zanolini. 2023. Streamlining Sleepy Consensus: Total-Order Broadcast with Single-Vote Decisions in the Sleepy Model. arXiv:2310.11331 [cs.DC]

[17] Michael Davis and Hans Vandierendonck. 2021. Achieving Scalable Consensus by Being Less Writey. In *Proceedings of the 30th International Symposium on High-Performance Parallel and Distributed Computing*. 257–258.

[18] Sisi Duan, Michael K Reiter, and Haibin Zhang. 2018. BEAT: Asynchronous BFT made practical. In *CCS*. ACM, 2028–2041.

[19] Sisi Duan, Xin Wang, and Haibin Zhang. 2023. Practical Signature-Free Asynchronous Common Subset in Constant Time. *ACM CCS* (2023).

[20] Sisi Duan and Haibin Zhang. 2022. Foundations of Dynamic BFT. In *SP*. 1317–1334.

[21] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. 1988. Consensus in the presence of partial synchrony. *JACM* 35, 2 (1988), 288–323.

[22] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. 2017. Algorand: Scaling byzantine agreements for cryptocurrencies. In *SOSP*. 51–68.

[23] Vipul Goyal, Hanjun Li, and Justin Raizes. 2021. Instant Block Confirmation in the Sleepy Model. In *Financial Cryptography and Data Security (FC)*.

[24] 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 *2019 49th Annual IEEE/IFIP international conference on dependable systems and networks*. IEEE, 568–580.

[25] Michel Hurfin, Achour Mostefaoui, and Michel Raynal. 1998. Consensus in asynchronous systems where processes can crash and recover. In *Proceedings Seventeenth IEEE Symposium on Reliable Distributed Systems (Cat. No. 98CB36281)*. IEEE, 280–286.

[26] Ernesto Jiménez, José Luis López-Presa, and Marta Patiño-Martínez. 2021. Consensus in anonymous asynchronous systems with crash-recovery and omission failures. *Computing* 103, 12 (2021), 2811–2837.

[27] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. 2017. Ouroboros: A Provably Secure Proof-of-Stake Blockchain Protocol. In *CRYPTO*. Springer, 357–388.

[28] Jan Kończak, Paweł T Wojciechowski, Nuno Santos, Tomasz Żurkowski, and André Schiper. 2019. Recovery algorithms for paxos-based state machine replication. *IEEE Transactions on Dependable and Secure Computing* 18, 2 (2019), 623–640.

[29] Leslie Lamport. 1998. The Part-Time Parliament. *ACM Trans. Comput. Syst.* 16, 2 (1998), 133–169.

[30] Giuliano Losa and Eli Gafni. 2023. Consensus in the Unknown-Participation Message-Adversary Model. arXiv:2301.04817 [cs.DC]

[31] Dahlia Malkhi, Atsuki Momose, and Ling Ren. 2023. Towards Practical Sleepy BFT. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*. 490–503.

[32] Ellis Michael, Dan RK Ports, Naveen Kr Sharma, and Adriana Szekeres. 2017. Recovering shared objects without stable storage. In *DISC*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

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

[34] Atsuki Momose and Ling Ren. 2022. Constant Latency in Sleepy Consensus. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 2295–2308.

[35] Satoshi Nakamoto. 2008. Bitcoin: A Peer-to-Peer Electronic Cash System. (2008).

[36] Joachim Neu, Ertem Nusret Tas, and David Tse. 2021. Ebb-and-flow protocols: A resolution of the availability-finality dilemma. In *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 446–465.

[37] Joachim Neu, Ertem Nusret Tas, and David Tse. 2022. The availability-accountability dilemma and its resolution via accountability gadgets. In *International Conference on Financial Cryptography and Data Security*. Springer, 541–559.

[38] Rafael Pass and Elaine Shi. 2017. The Sleepy Model of Consensus. In *Annual International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT)*. Springer, 380–409.

[39] Xiao Sui, Sisi Duan, and Haibin Zhang. 2022. Marlin: Two-Phase BFT with Linearity. In *2022 52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. 54–66. https://doi.org/10.1109/DSN53405.2022.00018

[40] The DiemBFT Team. 2020. The Diem Blockchain. (2020). https://developers.diem.com/docs/technical-papers/the-diem-blockchain-paper/

[41] Parity Technologies. 2023. Polkadot Whitepaper. https://polkadot.network/PolkaDotPaper.pdf. Accessed on 10-2023.

[42] Gavin Wood et al. 2014. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper* (2014).
[43] Maofan Yin, Dahlia Malkhi, Michael K. Reiter, Guy Golan Gueta, and Ittai Abraham. 2019. HotStuff: BFT Consensus with Linearity and Responsiveness. In *ACM Symposium on Principles of Distributed Computing (PODC)*. ACM, 347–356.
[44] Haibin Zhang, Sisi Duan, Boxin Zhao, and Liehuang Zhu. 2023. WaterBear: Practical Asynchronous BFT Matching Security Guarantees of Partially Synchronous BFT. In *USENIX Security*. 5341–5357.
[45] Haibin Zhang, Chao Liu, and Sisi Duan. 2022. How to achieve adaptive security for asynchronous BFT? *J. Parallel and Distrib. Comput.* 169 (2022), 252–268.

A PROOF OF KOALA-1

Proof of our VT-GPE. We begin with the correctness of our VT-GPE protocol shown in Algorithm 1. We split the graded delivery property into graded delivery-1 and graded delivery-2 for our proof.

Lemma A.1 (External Validity). *If an honest replica tgpe-decides* $(B, *, *)$ *such that* $B \neq \perp$*, at least one honest replica has verified* B *and* $Q(B, \sigma)$ *holds at it, where* σ *is the proof of* B*.*

Proof. If an honest replica p_i *tgpe-decides* $(B, *, *)$, p_i holds valid $E(B)$, i.e., p_i receives $f+1$ matching $\langle \text{ECHO} \rangle$ messages for B. At least one of the $\langle \text{ECHO} \rangle$ messages is from an honest replica. This replica must have verified B before echoing B, thus $Q(B, \sigma)$ holds. □

Lemma A.2 (Consistency). *If an honest replica tgpe-decides* $(B, *, *)$ *and another honest replica tgpe-decides* $(B', *, *)$, $B = B'$.

Proof. Assuming that p_i *tgpe-decides* $(B, *, *)$ and p_j *tgpe-decides* $(B', *, *)$ and $B \neq B'$. According to Lemma A.1, $Q(B, *)$ holds for at least one honest replica p_1 and $Q(B', *)$ holds for at least one honest replica p_2. In this case, p_1 must have sent an $\langle \text{ECHO} \rangle$ message for B at $t = \Delta$ and p_2 must have sent an $\langle \text{ECHO} \rangle$ message for B' at $t = \Delta$. As each honest replica sends an $\langle \text{ECHO} \rangle$ message for block B only if B is a winning input, p_1 must have forwarded B at $t = \Delta$. Similarly, p_2 has forwarded B' at $t = \Delta$. Therefore, every honest replica must have received $\langle \text{INPUT} \rangle$ messages for both B and B' by $t = 2\Delta$. At most one of these two inputs could be chosen as the winning input by any honest replica at $t = 2\Delta$.

Suppose the B is chosen by all honest replicas after $t = 2\Delta$. No honest replicas will send $\langle \text{WINNER1} \rangle$ or $\langle \text{READY} \rangle$ messages for B' at $t = 2\Delta$. No honest replicas will send $\langle \text{LOCK} \rangle$ messages for B' at $t = 3\Delta$. Since replicas need to receive $\langle \text{WINNER1} \rangle$, $\langle \text{READY} \rangle$, or $\langle \text{LOCK} \rangle$ messages from at least one honest replica to *tgpe-decide* block B' with grade 0, 1, and 2, none of them would *tgpe-decide* B', a contradiction. □

Corollary A.3. *If an honest replica receives a valid* $E(B)$ *and* $W_1(B)$ *for a block* B *and another honest replica receives a valid* $E(B')$ *and* $W_1(B')$ *for a block* B' *with* $view(B) = view(B')$, $B = B'$.

Proof. Suppose $view(B) = view(B') = v$. According to the protocol, an honest replica p_i will *tgpe-decide* B in VT-GPE$_v$ when it receives a valid $E(B)$ and $W_1(B)$ for B. Similarly, another honest replica p_j will *tgpe-decide* B' in VT-GPE$_v$ when it receives a valid $E(B')$ and $W_1(B')$ for B'. Due to Lemma A.2, $B = B'$. □

Lemma A.4 (Graded Delivery-1). *If an honest replica tgpe-decides* $(B, 1, *)$*, any honest replica tgpe-decides* $(B, 0, *)$.

Proof. If an honest replica p_1 *tgpe-decides* $(B, 1, *)$, it must have received at least $f+1$ valid $\langle \text{READY} \rangle$ messages for B and at least one honest replica p_2 has broadcast the $\langle \text{READY} \rangle$ message for B at $t = 2\Delta$. Before p_2 sent the $\langle \text{READY} \rangle$ message, it must have collected a valid $E(B)$ (i.e., $f+1$ $\langle \text{ECHO} \rangle$ messages) and forwarded $E(B)$. Therefore, at $t \geq 3\Delta$, every honest replica can collect a valid $E(B)$.

Let the proposer of B be p_3. Below we prove that all honest replicas must have observed a winning input for B at time $t = 2\Delta$. Firstly, according to the protocol, replica p_1 must have received at least $f+1$ valid $\langle \text{WINNER2} \rangle$ messages for B when it *tgpe-decides* B. In this case, an honest replica must have observed a winning input for B at $t = 3\Delta$. Therefore, at $t = 2\Delta$, no honest replica could observe a VRF evaluation higher than the VRF evaluation generated by p_3. Furthermore, no equivocation by p_3 is detected. Meanwhile, all honest replicas must have received the $\langle \text{INPUT} \rangle$ for B by $t = 2\Delta$. This is because p_2 already has $|E(B)| \geq f+1$ by $t = 2\Delta$ so at least one honest replica has previously set the $\langle \text{INPUT} \rangle$ message for B as its winning input by $t = \Delta$. As the honest replica forwards the $\langle \text{INPUT} \rangle$ message, any honest replicas awake at $t = 2\Delta$ must have considered the $\langle \text{INPUT} \rangle$ message for B as their winning input and sent $\langle \text{WINNER1} \rangle$ messages for B.

Since at least $h_a = f+1$ honest replicas are awake at $t = 2\Delta$, any honest replicas awake at time $t \geq 4\Delta$ must have $|E(B)| \geq f+1$ and $|W_1(B)| \geq f+1$ and then *tgpe-decide* $(B, 0, *)$. □

Lemma A.5 (Graded Delivery-2). *If an honest replica tgpe-decides* $(B, 2, *)$*, any honest replica tgpe-decides* $(B, 1, *)$.

Proof. If an honest replica p_1 *tgpe-decides* $(B, 2, *)$, it must have received at least $f+1$ valid $\langle \text{LOCK} \rangle$ messages for B and at least one honest replica p_2 has sent a $\langle \text{LOCK} \rangle$ message for B at $t = 3\Delta$. Before p_2 sent the $\langle \text{LOCK} \rangle$ message, it must have collected a valid $R(B)$ (at least $f+1$ matching $\langle \text{READY} \rangle$ messages) and forwarded $R(B)$. Therefore, at $t \geq 4\Delta$, every honest replica can collect a valid $R(B)$.

Let the proposer of B be p_3. Below we prove that all honest replicas must have observed a winning input for B at time $t = 3\Delta$. Firstly, when p_1 *tgpe-decides* B, it must have observed a winning input for B at $t = 4\Delta$. Therefore, at $t = 3\Delta$, no honest replica could observe a VRF evaluation higher than that of p_3 or any equivocating messages by p_3. Meanwhile, all honest replicas must have received the $\langle \text{INPUT} \rangle$ message for B by $t = 3\Delta$. This is because p_2 has $|R(B)| \geq f+1$ at time $t = 3\Delta$ and at least one honest replica has previously sent a $\langle \text{READY} \rangle$ message at time $t = 2\Delta$. The honest replica must have forwarded the $\langle \text{INPUT} \rangle$ message for B at $t = 2\Delta$. As a result, all honest replicas awake at $t = 3\Delta$ must have considered the $\langle \text{INPUT} \rangle$ message for B as their winning input and sent $\langle \text{WINNER2} \rangle$ messages for B.

Since at least $h_a = f+1$ honest replicas are awake at $t = 3\Delta$, any honest replicas awake at any $t \geq 4\Delta$ have $|R(B)| \geq f+1$ and $|W_2(B)| \geq f+1$ and then *tgpe-decide* $(B, 1, *)$. □

Lemma A.6 (Validity). *With a probability of* $\alpha > 1/2$*, all honest replicas tgpe-decide* $(B, 2, *)$ *where block* B *is tgpe-proposed by an honest replica.*

Proof. As at least $h_a = f+1$ honest replicas are awake at time $t = 0$ and there are at most f faulty replicas, with probability $\alpha > 1/2$, an honest replica's VRF evaluation will be the highest

among all awake replicas. Let the replica be p_1 and the block p_1 *tgpe-proposes* be (B, σ), where σ is the proof of block B. After p_1 broadcasts its $\langle \text{INPUT} \rangle$ message, all honest replicas awake at time $t \geq \Delta$ will set their winning input as the $\langle \text{INPUT} \rangle$ message for B.

As p_1 is an honest replica, $Q(B, \sigma)$ holds at all honest replicas. It is then not difficult to see that any honest replica broadcasts a $\langle \text{ECHO} \rangle$ message for B at $t = \Delta$. Each honest replica awake at $t = 2\Delta$ observes a valid $E(B)$ such that $|E(B)| \geq f + 1$ and broadcasts a $\langle \text{WINNER1} \rangle$ and a $\langle \text{READY} \rangle$ message for B. Similarly, all honest replicas awake at $t = 3\Delta$ observe a valid $R(B)$ such that $|R(B)| \geq f + 1$. Therefore, they broadcast $\langle \text{WINNER2} \rangle$ and $\langle \text{LOCK} \rangle$ messages for B. Finally, at $t \geq 4\Delta$, all awake honest replicas will observe a valid $L(B)$ such that $|L(B)| \geq f + 1$ and then *tgpe-decide* $(B, 2, *)$. □

Proof of Koala-1. We now prove the correctness of our ABC protocol. In this section, we prove the correctness of the protocol shown in Algorithm 2 (the none-pipelining mode).

THEOREM A.7 (SAFETY). *If an honest replica a-delivers a block B_1 before it a-delivers a block B_2, then no honest replica a-delivers the block B_2 without first a-delivering B_1.*

PROOF. Suppose an honest replica p_1 *a-delivers* block B_1 before it *a-delivers* B_2 and another honest replica p_2 *a-delivers* B_2 before it *a-delivers* B_1. W.l.o.g., we assume that p_1 *a-delivers* B_1 after it *tgpe-decides* $(B_1, 2, *)$ in VT-GPE$_{v_1}$. Additionally, p_2 *a-delivers* B_2 after it *tgpe-decides* $(B_2, 2, *)$ in VT-GPE$_{v_2}$. Obviously, $v_1 \neq v_2$, as otherwise the consistency property of VT-GPE is violated. W.l.o.g, let $v_1 < v_2$.

According to Lemma A.5, if p_1 *tgpe-decides* $(B_1, 2, *)$ for block B_1 in VT-GPE$_{v_1}$, any honest replica p_i (including p_2) *tgpe-decides* $(B_1, g, *)$ in VT-GPE$_{v_1}$ such that $g = 1$. Furthermore, if p_i *tgpe-decides* $(B_1, g, *)$ in VT-GPE$_{v_1}$ such that $g = 1$, by Lemma A.4, any honest replica will *tgpe-decide* $(B_1, 0, qc_1)$ in VT-GPE$_{v_1}$. According to our protocol, qc_1 is a valid *prepareQC* with $f+1$ $\langle \text{ECHO} \rangle$ messages and $f+1$ $\langle \text{WINNER1} \rangle$ messages for B_1. Therefore, any honest replica that enters the next view $v_1 + 1$ uses qc_1 as input. Furthermore, since any honest replica (including p_2) *tgpe-decides* $(B_1, 1, *)$, the replica sets its lock as B_1. The lock parameter can be set as a block that extends B_1 unless the replica becomes unlocked on B_1.

Since p_2 *tgpe-decides* $(B_2, 2, *)$ in view v_2 and is locked on B_1 in view v_1 (where $v_1 < v_2$), there must exist a view v_3 such that the following holds: 1) $v_1 < v_3 \leq v_2$; 2) an honest replica *tgpe-decides* a block B_3 in VT-GPE$_{v_3}$ and B_3 is conflicting with B_1; 3) a valid qc_3 is provided by the proposer of block B_3 and $Q(B_3, qc_3)$ is verified by at least one honest replica (as otherwise the external validity property of VT-GPE is violated). Here, $view(qc_3) < v_3$ as qc_3 is a proof included in the proposal of block B_3. W.l.o.g., suppose v_3 is the first view such that the above holds.

Towards a contradiction, we now show that B_3 cannot be a conflicting block of B_1. According to our protocol, qc_3 is a *prepareQC* and consists of $f+1$ matching $\langle \text{ECHO} \rangle$ and $f+1$ matching $\langle \text{WINNER1} \rangle$ messages. Any honest replica p_k that verifies $Q(B_3, qc_3)$ in view v_3 must have a lock (denoted as lock_k) such that $view(\text{lock}_k) \leq view(qc_3)$. As $view(\text{lock}_k) \geq v_1$, now there are two cases: $view(qc_3) = v_1$ and $view(qc_3) > v_1$. If $view(qc_3) = v_1$, qc_3 and qc_1 must have been formed in VT-GPE$_{v_1}$, where qc_1 is a valid *prepareQC* for B_1. Both qc_3 and qc_1 have been received by any honest replica awake after view v_1. According to Corollary A.3, the block for qc_3 is B_1, a contradiction. If $view(qc_3) > v_1$, we have $v_1 < view(qc_3) < v_3 \leq v_2$. The block corresponding to qc_3 is a conflicting block with B_1 and has been verified by at least one honest replica. However, we already assume that v_3 is the first view such that a conflicting block is proposed, a contradiction.

As B_3 cannot be a conflicting block of B_1, block B_2 extends block B_1. However, p_2 *a-delivers* B_1 after it *a-delivers* B_2, a contradiction. □

THEOREM A.8 (LIVENESS). *If an honest replica a-broadcasts a message m, then all awake honest replicas eventually a-deliver m.*

PROOF. We first prove that any block (B_1, qc) *tgpe-proposed* by any honest replica p_1 in a view v_1 can be verified by all honest replicas such that $Q(B_1, qc)$ holds. At the beginning of view v_1, B_1 extends the candidate of p_1 and qc is a *prepareQC* of candidate. As p_1 broadcasts an $\langle \text{INPUT} \rangle$ message for (B_1, qc) in VT-GPE$_{v_1}$, all awake honest replicas eventually receive the $\langle \text{INPUT} \rangle$ message for B_1. According to the graded delivery-1 property of VT-GPE, in any VT-GPE$_v$ such that $v < v_1$, if any honest replica *tgpe-decides* a block B with grade 1, p_1 must have *tgpe-decided* $(B, 0, *)$ and set its candidate as B. Therefore, the view number of p_1's candidate must be equal to or higher than that of the lock of any honest replica in view v_1. $Q(B_i, qc)$ thus holds at any honest replica.

According to the validity property of VT-GPE, with a probability of $\alpha > 1/2$, all honest replicas will *tgpe-decide* $(B, 2, *)$ for a block B in a VT-GPE instance. With trivial input dissemination, honest replicas can broadcast their *a-broadcast* messages and any honest replica can *a-broadcast* the messages that have not been *a-delivered*. It is then not difficult to see that any message m *a-broadcast* by an honest replica will eventually be *a-delivered* within a constant number of views. □

B THE PIPELINED KOALA-1

B.1 The Pseudocodes of Koala-1

We present the pseudocode of the wT-GPE in Algorithm 5 and the pseudocode of our pipelined Koala-1 protocol (with the fast path) in Algorithm 6.

B.2 Correctness Proof

Proof of our wT-GPE. We begin with the correctness of our wT-GPE protocol shown in Algorithm 5.

The proof of external validity and validity is similar to that of VT-GPE.

LEMMA B.1 (WEAK CONSISTENCY). *If an honest replica tgpe-decides $(B, g, *)$ with grade $g \geq 1$ and another honest replica tgpe-decides $(B', *, *)$, $B = B'$.*

PROOF. Assuming that p_i *tgpe-decides* $(B, g, *)$ with grade $g \geq 1$ and p_j *tgpe-decides* $(B', *, *)$ and $B \neq B'$. Similar to the proof of Lemma A.2, by the external validity property of wT-GPE, we can deduce that every honest replica must have received $\langle \text{INPUT} \rangle$ messages for both B and B' by $t = 2\Delta$. At most one of these two inputs could be chosen as the winning input by any honest replica at $t = 2\Delta$.

Algorithm 5 Validated Triple-graded Proposal Election with *Weak Consistency* for view v - wT-GPE$_v$.

Replica p_i executes the following algorithm at every time $t \geq 0$ after starting wT-GPE$_v$ in view v, and *tgpe-proposes* (B_i, σ_i) such that a global predicate $Q(B_i, \sigma_i)$ holds.
p_i maintains the following parameters for each received block B:
  $E(B) \leftarrow$ all received $\langle \text{ECHO}, B\rangle_*$ messages
  $R(B) \leftarrow$ all received $\langle \text{READY}, B\rangle_*$ messages
  $L(B) \leftarrow$ all received $\langle \text{LOCK}, B\rangle_*$ messages
if $t = 0$ **then**
  $(\rho_i, \pi_i) \leftarrow \text{VRF}_i(v)$
  broadcast $\langle \text{INPUT}, B_i, \sigma_i, \rho_i, \pi_i\rangle_i$
if $t = \Delta$ **then**
  if there exists a winning input $\langle \text{INPUT}, B_j, \sigma_j, \rho_j, \pi_j\rangle_j$ **then**
    forward the winning input (if not yet)
    if $Q(B_j, \sigma_j)$ holds **then**
      broadcast $\langle \text{ECHO}, B_j\rangle_i$
  else
    forward the equivocating INPUT messages by any replica
if $t = 2\Delta$ **then**
  update local winning input based on received $\langle \text{INPUT}\rangle$ messages
  if $\langle \text{INPUT}\rangle_j \neq \perp$ **then** // Let $\langle \text{INPUT}\rangle_j$ be the winning input
    forward $\langle \text{INPUT}\rangle_j$ (if not yet)
    if $|E(B_j)| \geq f + 1$ **then**
      broadcast $E(B_j)$ and $\langle \text{READY}, B_j\rangle_i$
if $2\Delta < t \leq 3\Delta$ **then**
  if $|R(B)| \geq f + 1$ for any block B **then**
    broadcast $R(B)$ and $\langle \text{LOCK}, B\rangle_i$ (if not yet)
if $t > 2\Delta$ **then**
  if $|L(B)| \geq f + 1$ for any block B **then**
    tgpe-decide $(B, 2, L(B))$
  if $|R(B)| \geq f + 1$ for any block B **then**
    tgpe-decide $(B, 1, R(B))$
if $t \geq 3\Delta$ **then**
  for each $\langle \text{INPUT}, B_j, \sigma_j, \rho_j, \pi_j\rangle_j$ **do** // from inputs with higher ρ_j
    if $|E(B_j)| \geq f + 1$ **then**
      tgpe-decide $(B_j, 0, E(B_j))$
      break
  if no block is *tgpe-decided* **then**
    tgpe-decide $\perp$

Suppose B is chosen by all honest replicas after $t = 2\Delta$. No honest replicas will send $\langle \text{READY}\rangle$ messages for B' at $t = 2\Delta$. No honest replicas will send $\langle \text{LOCK}\rangle$ messages for B' at $t = 3\Delta$. Since replicas need to receive $\langle \text{READY}\rangle$ or $\langle \text{LOCK}\rangle$ messages from at least one honest replica to *tgpe-decide* block B' with grade 1 and 2, p_j must *tgpe-decide* B' with grade 0.

On the other hand, according to the protocol, p_i must have received $f+1$ $\langle \text{READY}\rangle$ messages or $f+1$ $\langle \text{LOCK}\rangle$ messages for B. Therefore, at least one honest replica p_k has broadcast a $\langle \text{READY}\rangle$ message for B. p_k must have forwarded a valid $E(B)$ at $t = 2\Delta$. This $E(B)$ would be received by p_j by $t = 3\Delta$. p_j would *tgpe-decide* B with grade 0 instead of B', as the $\langle \text{INPUT}\rangle$ for B has a higher VRF than the $\langle \text{INPUT}\rangle$ for B'.

As a result, p_j would not *tgpe-decide* B', a contradiction. □

Algorithm 6 The pipelined ABC protocol. Code for p_i.

Initialize the following parameters
  $v \leftarrow 1$; candidate, lock $\leftarrow B_0$; *prepareQC*, *lockedQC* $\leftarrow \perp$.
  // *lockedQC* is used in the recovery protocol
Let Q be the following predicate for wT-GPE:
  Given (B, qc) *tgpe-proposed* by p_j, $Q(B, qc) \equiv (view(B) = v)$ **and**
  (qc is a valid *prepareQC*) **and** ($B.parent = qc.block$) **and**
  ($view(qc) > view(\text{lock})$ **or** $qc.block = \text{lock}$)
In each view v, replica p_i executes the following algorithm at every time $0 \leq t \leq 3\Delta$ w.r.t. view v, and then enter the next view $v + 1$.
if $t = 0$ **then**
  $B \leftarrow \langle \text{vals}, \text{H}(\text{candidate}), v\rangle_i$
  tgpe-propose $(B, prepareQC)$ in wT-GPE$_v$ with predicate Q
// The following events may be triggered after view v
upon p_i *tgpe-decides* $(B, 0, E(B))$ in wT-GPE$_v$ **do**
  if $view(B) > view(\text{candidate})$ **then**
    candidate $\leftarrow B$, $prepareQC \leftarrow E(B)$
upon p_i *tgpe-decides* $(B, 1, R(B))$ in wT-GPE$_v$ **do**
  if $view(B) > view(\text{lock})$ **then**
    lock $\leftarrow B$, $lockedQC \leftarrow R(B)$
upon p_i *tgpe-decides* $(B, 2, L(B))$ in wT-GPE$_v$ **do**
  if B has not been *a-delivered* **then**
    a-deliver B and all the ancestors of B

Lemma B.2 (Graded Delivery-1). *If an honest replica tgpe-decides* $(B, 1, *)$*, any honest replica tgpe-decides* $(B, 0, *)$.

Proof. If an honest replica p_1 *tgpe-decides* $(B, 1, *)$, we can deduce that every honest replica can collect a valid $E(B)$ at $t \geq 3\Delta$ (similar to the proof of Lemma A.4). Therefore, B can be *tgpe-decided* with grade 0 by any honest replica as long as no other block is *tgpe-decide* with grade 0. According to the weak consistency of wT-GPE, an honest replica could not *tgpe-decide* $(B', 0, *)$ with $B \neq B'$. As a result, all honest replicas *tgpe-decides* $(B, 0, *)$. □

Lemma B.3 (Graded Delivery-2). *If an honest replica tgpe-decides* $(B, 2, *)$*, any honest replica tgpe-decides* $(B, 1, *)$.

Proof. If an honest replica p_1 *tgpe-decides* $(B, 2, *)$, it must have received at least $f + 1$ valid $\langle \text{LOCK}\rangle$ messages for B and at least one honest replica p_2 has sent a $\langle \text{LOCK}\rangle$ message for B at $t \leq 3\Delta$. Before p_2 sent the $\langle \text{LOCK}\rangle$ message, it must have collected and forwarded a valid $R(B)$. Therefore, at $t \geq 4\Delta$, every honest replica can collect a valid $R(B)$. According to the protocol, every honest replica will *tgpe-decides* $(B, 1, *)$. □

Proof of pipelined Koala-1. We now prove the correctness of our ABC protocol. In this section, we prove the correctness of the protocol shown in Algorithm 6 (the pipelining mode).

Theorem B.4 (Safety). *If an honest replica a-delivers a block* B_1 *before it a-delivers a block* B_2*, then no honest replica a-delivers the block* B_2 *without first a-delivering* B_1.

Proof. Suppose an honest replica p_1 *a-delivers* block B_1 before it *a-delivers* B_2 and another honest replica p_2 *a-delivers* B_2 before it *a-delivers* B_1. W.l.o.g., we assume that p_1 *a-delivers* B_1 after it *tgpe-decides* $(B_1, 2, *)$ in wT-GPE$_{v_1}$. Additionally, p_2 *a-delivers* B_2 after it *tgpe-decides* $(B_2, 2, *)$ in wT-GPE$_{v_2}$. Obviously, $v_1 \neq v_2$, as

otherwise the weak consistency property of wT-GPE is violated. W.l.o.g, let $v_1 < v_2$.

According to Lemma B.3, if p_1 *tgpe-decides* $(B_1, 2, *)$ for block B_1 in wT-GPE$_{v_1}$, any honest replica p_i (including p_2) *tgpe-decides* $(B_1, g, *)$ in wT-GPE$_{v_1}$ such that $g = 1$. Furthermore, if p_i *tgpe-decides* $(B_1, g, *)$ in wT-GPE$_{v_1}$ such that $g = 1$, by Lemma B.2, any honest replica will *tgpe-decide* $(B_1, 0, qc_1)$ in wT-GPE$_{v_1}$. According to our protocol, qc_1 is a valid *prepareQC* with $f+1$ ⟨ECHO⟩ messages for B_1. Therefore, any honest replica that enters the next view v_1+1 uses qc_1 as input. Furthermore, since any honest replica (including p_2) *tgpe-decides* $(B_1, 1, *)$, the replica sets its lock as B_1. The lock parameter can be set as a block that extends B_1 unless the replica becomes unlocked on B_1.

Since p_2 *tgpe-decides* $(B_2, 2, *)$ in view v_2 and is locked on B_1 in view v_1 (where $v_1 < v_2$), there must exist a view v_3 such that the following holds: 1) $v_1 < v_3 \leq v_2$; 2) an honest replica *tgpe-decides* a block B_3 in wT-GPE$_{v_3}$ and B_3 is conflicting with B_1; 3) a valid qc_3 is provided by the proposer of block B_3 and $Q(B_3, qc_3)$ is verified by at least one honest replica (as otherwise the external validity property of wT-GPE is violated). Here, $view(qc_3) < v_3$ as qc_3 is a proof included in the proposal of block B_3. W.l.o.g., suppose v_3 is the first view such that the above holds.

Towards a contradiction, we now show that B_3 cannot be a conflicting block of B_1. According to our protocol, qc_3 is a *prepareQC* and consists of $f+1$ matching ⟨ECHO⟩ messages. Any honest replica p_k that verifies $Q(B_3, qc_3)$ in view v_3 must have a lock (denoted as lock_k) such that $view(qc_3) > view(\text{lock}_k)$ or the block of qc_3 is the same as lock_k. As $view(\text{lock}_k) \geq v_1$, now there are two cases: $view(qc_3) = v_1$ and $view(qc_3) > v_1$. If $view(qc_3) = v_1$, we have $view(\text{lock}_k) = v_1$. Since p_1 *tgpe-decides* B_1 with grade 2 and p_k *tgpe-decides* lock_k with grade 1 in wT-GPE$_{v_1}$, according to the weak consistency of wT-GPE, $\text{lock}_k = B_1$. Therefore, the block of qc_3 is also equal to B_1, a contradiction. If $view(qc_3) > v_1$, we have $v_1 < view(qc_3) < v_3 \leq v_2$. The block corresponding to qc_3 is a conflicting block with B_1 and has been verified by at least one honest replica. However, we already assume that v_3 is the first view such that a conflicting block is proposed, a contradiction.

As B_3 cannot be a conflicting block of B_1, block B_2 extends block B_1. However, p_2 *a-delivers* B_1 after it *a-delivers* B_2, a contradiction. □

Theorem B.5 (Liveness). *If an honest replica a-broadcasts a message m, then all awake honest replicas eventually a-deliver m.*

The proof of Theorem B.5 is the same as that of Theorem A.8.

C PRACTICAL RECOVERY PROTOCOL FOR KOALA-1

In Koala-1 presented in Sec. 3, we have assumed stable storage and message delivery. In this section, we provide a practical recovery protocol to remove this assumption. Similar to prior works [31, 34], the recovery protocol is used for *recovering* replicas (that become awake after sleeping) to catch up with awake replicas.

We follow the notations used by prior works and define a third status (besides awake and asleep) called *recovering*. An asleep replica first enters the *recovering* status before it becomes awake. The recovery period lasts for $\Gamma \geq 2\Delta$ time. In practice, the value of Γ may be adjusted by each replica depending on the amount of data it needs to receive.

We present the recovery protocol for Koala-1 in Algorithm 7. When a replica p_i enters the recovering status, it first computes the current view number v through the global synchronous clock. Then it broadcasts a $\langle \text{RECOVER}, v \rangle_i$ message to all replicas, starts a timer with a duration of Γ, and waits for the reply from other replicas. Upon receiving a $\langle \text{RECOVER}, v \rangle_i$ message from p_i, an awake honest replica p_j responds by sending to p_i the values of all its local parameters. Additionally, let t be the time p_j receives the ⟨RECOVER⟩ messsage, p_j forwards all messages it receives for view v until time $t + \Delta$. When the timer of p_i expires, p_i updates each of its local parameters to the *latest* valid one it has received, e.g., QC with the highest view number. After that, p_i sets its status as awake.

The recovery protocol can remove the stable storage and message delivery assumption as every recovered replica is able to collect all the information needed from awake replicas. In fact, although p_i can collect messages of every view prior to v, it is sufficient for p_i to only collect all messages of view v. Consider a view $v' < v$ such that at least one honest replica *tgpe-decides* a block B with grade 1 or 2 in VT-GPE$_{v'}$ (or wT-GPE$_{v'}$). The graded delivery property of VT-GPE (or wT-GPE) ensures that all awake honest replicas set their candidate and lock as either block B or a block higher than B. The recovery protocol then ensures that p_i will update its local parameters accordingly, and safety across views will never be violated. Meanwhile, every awake replica only needs to send its received messages of view v until $t + \Delta$. This is because p_i becomes online since time t, as the awake replica receives the ⟨RECOVER⟩ message at time t. Therefore, it is sufficient to forward all the messages received before $t + \Delta$ to p_i.

The communication complexity of the recovery protocol is $O(\kappa n^2 + n^2 L + lnL)$, where L is the size of a block, κ is the security parameter (i.e., length of the digital signature), and l is the length of the *a-delivered* history of the replicas.

Algorithm 7 Recovery protocol for Koala-1

1: Replica p_i executes the following algorithm.
2: **upon** going online **do**
3: Let v be the current view.
4: broadcast $\langle \text{RECOVER}, v \rangle_i$
5: **upon** the timer expires **do**
6: *prepareQC* ← the received *prepareQC* with the highest view number
7: candidate ← the block matching *prepareQC*
8: *lockedQC* ← the received *lockedQC* with the highest view number
9: lock ← the block matching *lockedQC*
10: set the state as *awake* and participate in the protocol's execution
11: // respond to a recovering replica
12: **upon** receiving $\langle \text{RECOVER}, v \rangle_j$ at time t **do**
13: send to p_j the current *a-delivered* history, candidate, lock, *prepareQC*, *lockedQC*
14: forward to p_j all messages of view v received until $t + \Delta$

D PROOF OF KOALA-2

We prove the correctness of the Koala-2 protocol shown in Algorithm 3 and Algorithm 4 (without the GAT assumption).

Lemma D.1. *If an honest replica falls asleep in a view v and later becomes awake, it will start the execution from view v' such that $v' \geq v + 1$.*

Proof. Consider an honest replica p_1 that falls asleep in a view v and p_1 is the first replica that wakes up later. Towards a contradiction, we assume that p_1 starts from view $v' \leq v$ after it recovers. According to our protocol, if p_1 previously entered view v, a quorum of replicas must have already entered view $v-1$ and sent their $\langle$TIMEOUT$\rangle$ messages to p_1 during the view change. Meanwhile, during the recovery, p_1 must have received *timeoutQC*s from a quorum of replicas. As p_1 enters view v' after it wakes up, the highest view number v_h among the *timeoutQC*s p_1 receives is no higher than $v' - 3$. This is because during the recovery process, p_1 waits for a *timeoutQC* of a view no lower than v_h+2 and starts the execution from view no lower than $v_h + 3$, where v_h is the view number of the highest *timeoutQC* p_1 receives in the $\langle$ECHO-1$\rangle$ messages. According to the quorum intersection rule, at least one honest replica must have sent a *timeoutQC* to p_1 for a view no higher than $v' - 3$ during the recovery process while its latest *timeoutQC* is for a view no lower than $v - 2$. Therefore, $v' \geq v + 1$, a contradiction. □

Lemma D.2. *Let qc_1 and qc_2 be valid QCs of any type (e.g., prepareQCs, precommitQCs or commitQCs). If the blocks of qc_1 and qc_2 are conflicting, $view(qc_1) \neq view(qc_2)$.*

Proof. Towards a contradiction, let $view(qc_1) = view(qc_2) = v$. Let B_1 be the block of qc_1 and B_2 be the block of qc_2. According to the protocol, qc_1 includes $n - f - s$ matching votes for B_1 and qc_2 includes $n - f - s$ matching votes for B_2. According to the quorum intersection rule, at least one honest replica must have voted for both B_1 and B_2 in view v, a contradiction. Note that the replica must be awake in view v as according to Lemma D.1, if it fell asleep in view v, it must have already entered view $v+1$. Therefore, $view(qc_1) \neq view(qc_2)$. □

Lemma D.3. *If commitQC of a block B is formed in view v, every honest replica that falls asleep in a view no lower than v and recovers in a view higher than v must become locked on either B or a block that extends B.*

Proof. Our proof consists of two parts. First, we prove that every honest replica that falls asleep in a view no lower than v and recovers in a view higher than v must become locked on a block B' no lower than B during the recovery. Then, we show that B' is either B or a block that extends B.

We begin with the first part. Consider an honest replica p_1, the first honest replica that falls asleep in a view no lower than v and wakes up in a view v' where $v' > v$. Towards a contradiction, assume p_1 is locked on a block B' after it recovers and B' is lower than B. In the recovery protocol, p_1 broadcasts a $\langle$RECOVERY-2$\rangle$ message and receives $\langle$ECHO-2, v', (*prepareQC*, *lockedQC*)$\rangle$ messages from a quorum of replicas. According to the protocol, p_1 only accept $\langle$ECHO-2$\rangle$ messages with a view number $v' > v_h+2$, where v_h is the view number of the highest *timeoutQC* p_1 receives. As p_1 already entered view v before it fell asleep, at least a quorum of replicas must have a *timeoutQC* for a view no lower than $v-2$ so $v_h+2 \geq v$. In our recovery protocol, every honest replica waits until its view is at least $v_h + 3 \geq v + 1$ before it sends an $\langle$ECHO-2$\rangle$ message to p_1.

Meanwhile, according to our assumption, p_1 is locked on a block lower than B after recovery. Therefore, the highest *lockedQC* p_1 receives during recovery is lower than $view(B)$. As a *commitQC* is formed in view v, a quorum of replicas must have become locked on B in view v. According to the quorum intersection rule, at least one honest replica p_2 has been locked on block B and it sends an $\langle$ECHO-2$\rangle$ message with a *lockedQC* for a block lower than B. This can only happen when p_2 falls asleep in a view no lower than v and later recovers, contradicting our assumption that p_1 is the first replica that recovers in a view higher than v. This completes the first part of the proof.

We now show that B' is either B or a block that extends B. Towards a contradiction, we assume that B' is conflicting with B. Let $view(B')$ be v_1. We already show that B' is no lower than B, we have $v_1 > v$, as otherwise Lemma D.2 is violated. As p_1 is locked on B' after it recovers, p_1 must have received a *lockedQC* for B' in the $\langle$ECHO-2$\rangle$ messages. Therefore, at least a quorum of replicas have voted for block B' in view v_1.

There must exist a view v_2 such that the following holds: 1) $v < v_2 \leq v_1$; 2) a block B_2 is proposed in view v_2 and B_2 conflicts with B; 3) a valid *prepareQC* qc_2 is provided by the proposer of block B_2 and at least a quorum of replicas in view v_2 vote for block B_2. Here, $view(qc_2) < v_2$ as qc_2 is a proof included in the proposal of block B_2. W.l.o.g, let v_2 be the first view such that the above holds. We already proved that at least a quorum of replicas are locked on block B in view v. Therefore, according to the quorum intersection rule, at least one honest replica p_3 is locked on block B in view v and votes for block B_2 in view v_2. According to the protocol, if p_3 is awake between view v and v_2, this can only happen if B_2 is non-conflicting with B. Note that the case where p_3 falls asleep and later recovers violates our assumption that p_i is the first replica that recovers in a view higher than v. This completes the proof of the lemma. □

Corollary D.4. *If the commitQC of a block B is formed in view v, every honest replica that becomes locked on B in view v will always be locked on B or a block that extends B.*

Proof. Suppose p_1 is an honest replica that becomes locked on B in view v. If it remains awake after view v, it will never update its *lockedQC* to a lower one. According to the second part in the proof of Lemma D.3, the locked block of p_1 must be B or a block that extends B. We therefore focus on the case that p_1 falls asleep in a view no lower than v and later recovers. According to Lemma D.3, p_1 must have become locked on either B or a block that extends B during its recovery. □

Theorem D.5 (Consistency). *If an honest replica delivers a transaction tx and another honest replica delivers a transaction tx', both with the same order, $tx = tx'$.*

Proof. Towards a contradiction, suppose an honest replica p_1 delivers a transaction tx and another honest replica p_2 *delivers* a transaction tx' with the same order, $tx' \neq tx$. Let B_1 be the block

that contains tx and B_2 be the block that contains tx'. Obviously, $B_1 \neq B_2$, as otherwise the order of tx and tx' would not be the same. We assume that an honest replica p_1 delivers B_1 after it receives a *commitQC* qc_1 in view v_1 and another honest replica p_2 delivers B_2 after it receives a *commitQC* qc_2 in view v_2. According to Lemma D.2, $v_1 \neq v_2$. W.l.o.g, let $v_1 < v_2$.

When the *commitQC* of B_1 was formed in view v_1, a quorum of replicas must have become locked on B_1. According to Corollary D.4, these replicas will always be locked on B_1 or a block that extends B_1. As the *commitQC* of B_2 is formed in view v_2, a quorum of replicas must have voted for B_2. According to the quorum intersection rules, at least one honest replica was locked on B_1 or a block that extends B_1 while it voted for B_2 in view v_2. Since B_1 conflicts with B_2, B_2's parent block must be higher than B_1. By induction, there must exist an ancestor block (denoted as B_3) of B_2 that is higher than B_1 and the parent block of B_3 is lower than B_1. Similarly, at least one honest replica was locked on B_1 or a block that extends B_1 while it voted for B_3 in view $view(B_3)$. However, this can only happen if B_3 extends B_1, contradicting our assumption that B_2 conflicts with B_1. □

Lemma D.6. *If the precommitQC of a block B is formed in view v, every honest replica that falls asleep in a view no lower than v and recovers in a view higher than v must have a prepareQC of a view $v' \geq v$.*

The proof is similar to that for Lemma D.3 and we omit the details.

Corollary D.7. *If the precommitQC of a block B is formed in view v, every honest replica that sets a prepareQC corresponding to B in view v will always have a prepareQC with a view number no lower than v.*

We can deduce this corollary from Lemma D.6. The deduction process is similar to that for Corollary D.4.

Lemma D.8. *After GST, there exists a bounded time period T_a such that if the leader of view v is honest and awake and a quorum of awake honest replicas are in view v at any moment of T_a, then a block is delivered.*

Proof. Suppose after GST, the leader p_i is honest in a new view v. p_i can collect ⟨new-view⟩ messages from a quorum of replicas. It computes the highest *prepareQC* (denoted as *highQC*) among them and broadcasts a new block B extending the block of *highQC*. Any honest replica that receives block B will compare B with its *lockedQC* and considers B valid only if B extends the block for the *lockedQC* or *highQC* is higher than the *lockedQC*. Let qc_h be the highest *lockedQC* among all honest replicas and B_h be the block for qc_h. When qc_h was formed, a quorum of replicas must have set their *prepareQC* as the *prepareQC* for B_h in a view $v' < v$. According to Corollary D.7, the quorum of replicas will always have a *prepareQC* with a view number no lower than v'. By quorum intersection, at least one honest replica within the quorum must have sent a ⟨new-view⟩ message to the leader at the beginning of view v. Therefore, the leader must obtain a *highQC* with a view number no lower than v'. Due to Lemma D.2, the block of *highQC* must be B_h or a block higher than B_h. Therefore, Any honest replica that receives B in view v will vote for B in the prepare phase.

Under the assumption that a quorum of awake honest replicas are synchronized in view v and p_i remains awake, p_i is able to receive the ⟨prepare⟩ votes for B from a quorum of replicas and form a *prepareQC*. Similarly, a quorum of honest replicas will vote in other phases and B will be delivered.

□

It is worth mentioning that our proof for the above lemma assumes that the leader p_i is awake for a sufficiently long time so that block B is eventually delivered. We believe this is reasonable assumption as otherwise the protocol will never be live anyway.

Theorem D.9 (Liveness). *If an honest replica a-broadcasts a message m, then all awake honest replicas eventually deliver m.*

Proof. We first prove that the recovery protocol is non-blocking and all recovering replicas can eventually become awake. In the recovery protocol, a recovering replica p_i needs to wait for $n-f-s$ ⟨echo-1⟩ messages, a *timeoutQC* for view $v_h + 2$, and $n - f - s$ ⟨echo-2⟩ messages with view numbers higher than $v_h + 2$, where v_h is the view number of the highest *timeoutQC* among received ⟨echo-1⟩ messages. Since at least $n-f-s$ honest replicas are awake at any time, $n - f - s$ ⟨echo-1⟩ messages and $n - f - s$ ⟨timeout⟩ messages of view $v_h + 2$ can be eventually received. Thus, p_i must observe a *timeoutQC* at least for view $v_h + 2$. After that, p_i sends a ⟨recovery-2, *timeoutQC*⟩ to all awake replicas. Each awake honest replica receiving the message must enter a view $v' > v_h + 2$ and then send its ⟨echo-2⟩ message. Therefore, p_i must receive $n-f-s$ ⟨echo-2⟩ messages with view numbers higher than v_h+2. After processing all these messages, p_i can complete the recovery protocol. Thus, all recovering replicas can eventually become awake.

Finally, due to Lemma D.8, the liveness of the protocol follows that of HotStuff. □

E KOALA-2 UNDER THE GAT ASSUMPTION.

Koala-2 can be adapted to the GAT assumption with some minor modifications. First, we increase the quorum size in the normal case operation and the view change protocol from $n-f-s$ to $n-f$. In this way, we can reduce the lower bound to $n \geq 3f + s + 1$ as is proved in Sec. 5.1. Second, we slightly modify the recovery protocol. Specifically, after receiving the ⟨echo-1⟩ messages and obtaining v_h, the recovering replica participates in the view change protocol for both view $v_h + 1$ and $v_h + 2$ and broadcasts two ⟨timeout⟩ messages for view $v_h + 1$ and $v_h + 2$. The motivation is to ensure that after GAT, each recovering replica can eventually receive a *timeoutQC* for view $v_h + 2$ and complete the recovery protocol. Indeed, a *timeoutQC* requires $n - f$ ⟨timeout⟩ messages. To form the *timeoutQC* of any view, all honest replicas including the recovering replicas need to participate in the view change and broadcast their ⟨timeout⟩ messages.

F TRANSFORMING OTHER BFT TO SLEEPY CONSENSUS

Our transformation approach for Koala-2 is generic and can be extended to other partially synchronous BFT. Using PBFT as an example [12], we show that by slightly modifying the view change

and recovery protocols, we can transform PBFT into a sleepy consensus protocol. For completeness, we show the pseudocode of the recovery protocol for PBFT in Algorithm 8.

- **View change.** We only need to insert one step before the view change protocol of PBFT. In particular, whenever a replica starts the view change, it first broadcasts a $\langle \text{TIMEOUT} \rangle$ message. Upon receiving $n - f$ matching $\langle \text{TIMEOUT} \rangle$ messages and forming a *timeoutQC*, the replica starts the view change protocol as specified in PBFT. The *timeoutQC* for a view v can be used as proof that a sufficiently large number of honest replicas have entered view v, as honest replicas collect the *timeoutQC* before they actually start the view change.
- **Recovery protocol.** In PBFT, replicas do not maintain *lockedQC* but use a set of *prepare certificates* instead to denote the status for blocks. Namely, in PBFT, the leader may propose multiple blocks concurrently. For blocks of each height (i.e., *sequence number* in PBFT), each replica maintains at most one prepare certificate. To build the recovery protocol for PBFT, a recovering replica needs to synchronize with other replicas all the prepare certificates for blocks that are not delivered yet. Such a protocol achieves a similar goal with the synchronization of *lockedQC*.

Algorithm 8 Recovery protocol for PBFT (for replica p_i).

Let *curView* be the current view number and *s-checkpoint* be the stable checkpoint.
Let $P[k]$ denote the prepare certificate for a block with sequence number k. The highest sequence number k such that $P[k] \neq \perp$ is viewed as the *max-s* of P.
as a recovering replica
    broadcast a $\langle \text{recovery-1} \rangle_i$ message
    wait for $n - f - s$ $\langle \text{ECHO-1}, \textit{timeoutQC} \rangle_*$
    $v_h \leftarrow$ the view number of the highest *timeoutQC* among all received $\langle \text{ECHO-1} \rangle$ messages.
    wait for a *timeoutQC* tc such that $view(tc) \geq v_h + 2$
    $\textit{timeoutQC} \leftarrow tc$
    broadcast $\langle \text{RECOVERY-2}, \textit{timeoutQC} \rangle_i$
    wait for $n - f - s$ $\langle \text{ECHO-2}, v', (\textit{s-checkpoint}, P) \rangle_*$ such that $v' > v_h + 2$
    s-checkpoint $\leftarrow$ the *s-checkpoint* with the highest sequence number among received $\langle \text{ECHO-2} \rangle$ messages
    min-s $\leftarrow$ the sequence number of *s-checkpoint*
    max-s $\leftarrow$ the highest *max-s* among all P of received $\langle \text{ECHO-2} \rangle$ messages
    for $k = \textit{min-s}, \textit{min-s} + 1, ..., \textit{max-s}$ **do**
        $P[k] \leftarrow$ the $P[k]$ with the highest view number among all P of received $\langle \text{ECHO-2} \rangle$ messages
        if no P contains a block of sequence number k **then**
            $P[k] \leftarrow \perp$
    $\textit{curView} \leftarrow view(\textit{timeoutQC}) + 1$
    send $\langle \text{VIEW-CHANGE}, \textit{curView}, \textit{min-s}, \textit{s-checkpoint}, P \rangle$ to the leader of *curView*
    set the state as *awake* and rejoin the main protocol's execution
as an awake replica
    upon receiving $\langle \text{RECOVERY-1} \rangle_j$ **do**
        send $\langle \text{ECHO-1}, \textit{timeoutQC} \rangle_i$ to replica p_j
    upon receiving $\langle \text{RECOVERY-2}, \textit{timeoutQC} \rangle_j$ **do**
        if $view(\textit{timeoutQC}) \geq \textit{curView}$ **then**
            start view change and proceed to view $view(\textit{timeoutQC})+1$
        send $\langle \text{ECHO-2}, \textit{curView}, (\textit{s-checkpoint}, P) \rangle_i$ to replica p_j