DuoBFT: Resilience vs. Efficiency Trade-off in Byzantine Fault Tolerance

Balaji Arun and Binoy Ravindran
Virginia Tech
Blacksburg, VA

Abstract
This paper presents DuoBFT, a Byzantine fault-tolerant protocol that provides two features: Cheaper Resilience and Dual Fault Assumptions. First, by enhancing a fraction of replicas in the system with trusted components, DuoBFT enables commit decisions in the Hybrid fault model with quorums that are about half the size of regular byzantine quorums. Second, DuoBFT exposes both the Hybrid and BFT fault models to the clients and lets them make commit decisions under either of these models.

We first enable a notion called Flexible Quorums in the Hybrid fault model by revisiting the quorum intersection requirements in hybrid protocols. We apply the Flexible Quorums technique to MinBFT, a state of the art hybrid protocol, and consequently build on it to achieve DuoBFT.

I. INTRODUCTION

Byzantine fault tolerant (BFT) protocols are a building block of intrusion tolerant replicated services [1], [2]. Since the advent of the decentralized ledger or Blockchain technology, BFT protocols have seen revitalized interest due to their applicability in this new domain.

Traditionally, the design of a BFT protocol is driven by a set of assumptions that includes the timing model (synchrony, partial synchrony, or asynchrony) and the failure model (BFT, Hybrids [3], XFT [4]). The safety and liveness guarantees of a BFT protocol relies on these assumptions. A variety of Byzantine tolerant protocols [5]–[10] exist in literature that adopt various permutations of these assumptions and are being used in Blockchain systems [11], [12].

A. Challenges in the BFT model

The Byzantine fault model, being the most general one, allows any arbitrary behavior in the system that deviates from the protocol specification. An partially synchronous protocol in this fault model can optimally tolerate less than 1/3 failures. Thus, tolerating even a few tens of failures would require more than a hundred replicas.

One of the major malicious behavior exhibited in a BFT system is equivocation i.e. providing different information to different processes. This behavior is thwarted in the hybrid fault model [3]. The hybrid fault model assumes the existence of a trusted component within each replica that curbs equivocation or lying. The trusted component can fail only by crashing. This technique allows Hybrid protocols to tolerate less than 1/2 failures, about 50% more than the number of faults tolerated in the BFT model. Another consequence of the trusted component is a reduction of one communication step in the normal case protocol. Table I compares the different properties of Hybrid and BFT protocols. Trusted components in early hybrid protocols were realized using custom ASICs [13] and FPGAs [14], but these devices performed poorly and were very hard to program. However, these days the trusted subsystem can be realized easily in software by leveraging capabilities in commodity processors such as Intel SGX [15]. In capable processors, the software runs within the trusted execution environments (TEEs) that are isolated from other parts of the machine including the operating system.

The use of trusted execution environments raises some challenges. First, it greatly reduces the choice of hardware used to deploy such protocols. At the time of this writing, Intel SGX [16] and ARM TrustZone [17], were the only available capabilities for commodity processors. However, byzantine protocols implicitly/explicitly require diversity in the deployment stack to reduce the number of correlated failures [18]. Second, security vulnerabilities have been discovered in trusted execution environments recently [19], [20]. Although active research in the area aims to solve these problems, the impact of undiscovered vulnerabilities raises concerns. In addition, the fact that there are only a few vendors for such enclaves, only exemplifies the problem. This raises questions on their applicability to BFT protocols and blockchain systems.

|                | BFT | Hybrid |
|----------------|-----|--------|
| System size    | $3f + 1$ | $2f + 1$ |
| Quorum size    | $2f + 1$ | $f + 1$ |
| Communication steps | 3   | 2  |
| Trusted component | ×   | ✓    |
| Malicious behaviors prevented | None | Equivocation |
Despite the challenges, the hybrid fault model has some unique properties that can be appealing for certain applications. The model tolerates more faulty replicas in the partial synchrony model, that can only be matched under the synchronous timing model \[21\]. To cope with the challenges, we propose reducing the reliance on trusted execution environments, and further, offer mechanisms to fallback to a safer fault model during drastic situations.

B. Towards providing a dual fault model

We propose DuoBFT, a BFT protocol that solves the aforementioned problems by offering two unique features: Cheap Resilience and Dual Fault Assumption under the partially synchronous timing model.

**Cheap Resilience.** We introduce a technique that allows DuoBFT to make consensus decisions with only a quorum of little over 1/3 replicas. To achieve this, we revisit the quorum intersection in Hybrid protocols. We show that only quorums across different views must intersect while those within the same view need not. With this intuition, we present **Flexible MinBFT**, a modified version of the state-of-the-art hybrid fault tolerant protocol MinBFT \[3\], using the new **Flexible Hybrid Quorum** intersection technique. This technique separates the relationship between the number of replicas \(N\) and the number of faults tolerated \(f\) that is typical in traditional hybrid protocols, and instead enables adopting different values of \(f\) for the same \(N\). **Flexible MinBFT** uses normal commit quorums of size at least \(f + 1\) and view change quorums of size at least \(N - f\).

DuoBFT incorporates Flexible Hybrid Quorums in a traditional BFT protocol and enables hybrid quorums decisions with only 1/3 hybrid replicas\[1\]. There is no complete reliance on trusted execution environments since DuoBFT only needs a third of replicas to have the trusted hardware capability. In the common case, the fraction of hybrid replicas are enough to make commit decisions in two communication steps under the hybrid fault model. Alternatively, since DuoBFT is also a BFT protocol, commit decisions under the BFT model with a 2/3 quorum and three communication steps is also possible. This is useful when hybrid quorums are not possible or feasible.

**Dual Learner Assumptions under the Partial-Synchrony model.** DuoBFT exposes the dual fault model outside of the replicas by separating the commit rules from the protocol. This allows different learners to make commit decisions by choosing between the BFT and the Hybrid fault models. In doing so, DuoBFT provides an unique trade-off to the learners: quick decisions made possible by hybrid replicas versus uncompromising resilience to malicious behavior. Furthermore, our solution allows different learners to adapt their fault assumptions dynamically without depending on the replicas.

By letting the learners dynamically choose the fault model, DuoBFT empowers them to tailor end-user experiences at a finer granularity, depending on the operation. For instance, in a payment system, a $10 transaction may require faster and cheaper guarantees than a $1M transaction \[22\]. Thus, learners processing the $10 transaction can take advantage of the faster response under the hybrid fault model to offer better end-user experience, while learners processing the $1M transaction can spend the additional communication steps to decide under the BFT model and provide stronger resilience guarantees to the end-user.

In DuoBFT, the replicas propose and vote on blocks that contain client transactions or operations, and the learners make commit decisions on the blocks depending on their adopted fault model. Replicas internally collect two types of quorums to help the learners make their commit decisions. The Hybrid commit quorum consists of votes from the hybrid replicas, and the BFT commit quorum consists of votes from any 2/3 replicas. Clients can collect either of these quorums to make the respective commit decisions. However, under certain system conditions, learners may need to update their assumption to continue to make proper commit decisions. The hybrid replicas may be unreachable and since there are only about 1/3 such replicas, learners adopting the hybrid model need to switch to the BFT model to make progress. Similarly, a BFT quorum may not attainable in time, possibly due to a network partition, in which case learners can temporarily adopt the hybrid model. DuoBFT’s dual fault model provides such flexibility. Furthermore, if severe security vulnerabilities were discovered in the trusted execution environments, the learners can simply switch to using the BFT model for added safety while the vulnerabilities are being fixed.

Furthermore, the flexibility provided by the DuoBFT’s dual fault model is better than speculation \[8\], \[10\], or tentative \[23\] execution capabilities provided by other known protocols. While speculation requires 50% larger quorums than PBFT \[5\] and the tentative execution only reduces the execution overhead by overlapping the last communication step with execution, the hybrid model uses 50% smaller quorums than PBFT and reduces one overall communication step. At the same time, DuoBFT does not incur any more overall communication steps than that required to commit under the BFT model, unlike \[7\], \[10\].

This paper makes the following contributions:

- **Flexible Hybrid Quorums:** We show that the use of majority quorums in the Hybrid fault model can be relaxed and replaced with simple intersecting quorums. This allows flexibility in the sizes of quorums used for different parts of the protocol.

- **Flexible MinBFT:** We apply Flexible Hybrid Quorums to MinBFT and present a modified protocol that provides smaller commit quorums in exchange for a larger view change quorums. We show that Flexible MinBFT satisfies all the guarantees of the original protocol.

- **DuoBFT:** We present a protocol that provides Cheap Resilience and Dual Fault Assumption under the partial synchrony timing model. DuoBFT can make Hybrid commit decisions with 1/3 replicas and also make traditional BFT commit decisions on the blocks depending on their adopted fault model. Replicas internally collect two types of quorums to help the learners make their commit decisions. The Hybrid commit quorum consists of votes from the hybrid replicas, and the BFT commit quorum consists of votes from any 2/3 replicas. Clients can collect either of these quorums to make the respective commit decisions. However, under certain system conditions, learners may need to update their assumption to continue to make proper commit decisions. The hybrid replicas may be unreachable and since there are only about 1/3 such replicas, learners adopting the hybrid model need to switch to the BFT model to make progress. Similarly, a BFT quorum may not attainable in time, possibly due to a network partition, in which case learners can temporarily adopt the hybrid model. DuoBFT’s dual fault model provides such flexibility. Furthermore, if severe security vulnerabilities were discovered in the trusted execution environments, the learners can simply switch to using the BFT model for added safety while the vulnerabilities are being fixed.

1*Hybrid* replicas are those that have trusted components
decisions. The protocol supports learners under two fault models – Hybrid and BFT – to provide an unique trade-off: decisions made possible by hybrid replicas versus uncompromising resilience to malicious behavior. We also show that DuoBFT provides the guarantees of both fault models within the same protocol.

By letting the learners to choose their commit rules, DuoBFT falls into a class of similar protocols that include Flexible BFT [22] and Bitcoin [24]. However, DuoBFT provides a completely different set of properties than these protocols. For instance, the trust-based assumption that DuoBFT adopts results in an efficient algorithm that lets learners decide quickly and with strong resilience. Furthermore, as we will discuss later, the a-b-c fault model adopted in Flexible BFT can, in fact, be seen as complementary to the guarantees provided by DuoBFT.

The rest of the paper is organized as follows. Section II presents the terminology and system model. Section III presents Flexible Hybrid Quorums and Flexible MinBFT. Section IV presents the DuoBFT protocol, explanation of its properties along with proofs, and some optimizations. A discussion on some unique features of DuoBFT with respect to existing solutions is presented in Section V. Section VII concludes our work.

II. Preliminaries

We consider a system that consists of a set of nodes, called replicas that communicate via message passing. These replicas implement a replicated service that receive requests from client and ensure that the same sequence of totally ordered requests is learnt by the client. The goal of the consensus protocol is to ensure agreement on the replicated state among replicas withstanding a number of faulty servers.

Per Lamport’s terminology [25], a consensus protocol consists of three roles: proposers proposes values to add to the sequence, acceptors vote to add these values to the sequence, and learners learn the sequence of values and execute them. In a replicated service, the clients act as both proposers and learners, while the replicas act as the acceptors. A consensus protocol provides the following two guarantees:

- **Safety.** Any two learners learn the same sequence of values.
- **Liveness.** A value proposed by the proposer will eventually be executed by every learner.

Most consensus protocols offer a single fault model that the learners use to learn the sequence of values. In contrast, DuoBFT offers a dual fault model and supports learners with fault assumptions. Thus, based on the assumptions, learners may learn values differently. To support learners with different assumptions, DuoBFT provides the following guarantees similar to [22]:

- **Safety.** Any two learners with correct but potentially different assumptions learn the same sequence of values.
- **Liveness.** A value proposed by the proposer will be eventually executed by every learner with a correct assumption.

**Fault Model.** DuoBFT supports two fault models: the Byzantine fault model and the Hybrid fault model. In both the models, a replica is correct if it strictly follows the algorithm, otherwise it is faulty. In addition, faulty replicas can collude to cause harm to correct replicas.

In the Hybrid model, we assume the existence of a trusted execution environment in each hybrid replica that hosts the protocol’s trusted code. We use the term hybrid replica to refer to those replicas that have the trusted component and participate in the Hybrid fault model. In MinBFT, every replica is a hybrid replica, while in DuoBFT only a subset of replicas are hybrid ones. Despite some replicas being faulty, the trusted execution environment in each hybrid replica is assumed to be tamperproof and the code it executes strictly follows the algorithm. We will revisit this requirement later in Section V. Lastly, the trusted component can fail by crashing.

**Timing Model.** We assume the partially synchronous timing model [26]. Eventually, there exists a time during which correct replicas communicate synchronously and messages are timely. We, further, assume that the network can drop, reorder, and duplicate messages. To ensure reliable delivery of messages, we rely on generic retransmission techniques that use a buffer to store outgoing messages and retransmits them periodically. Furthermore, we do not assume any bounds on processing and communication delays except that such delays do not grow indefinitely.

**Cryptography.** We assume that the adversary cannot break cryptographic computation such as hashes and signatures. In addition, the hashing algorithms are collision resistant. Every replica is aware of other replicas’ public keys. Each replica can verify the messages they receive using the corresponding replica’s public key.

III. Flexible Quorums in the Hybrid Model

In this section, we revisit the quorum intersection in the Hybrid model, and introduce Flexible Hybrid Quorums, a technique that relaxes the majority quorum intersection requirement. With this technique, only quorums across views must intersect to ensure safety, while quorums within the same view need not intersect. Consequently, Hybrid protocols using the flexible quorum technique can opt for using smaller non-majority quorums during normal executions, in exchange for using much larger than majority quorums during view changes. We perform our analysis in the context of MinBFT, a state-of-the-art Hybrid protocol. Thus, we first overview MinBFT in Section III-B and then introduce the flexible quorum technique to produce Flexible MinBFT in Section III-C. First, we begin with a background on the hybrid model.
A MinBFT replica executed the following protocol.

1. **Prepare.** The primary assigns a sequence number to the client request and sends a Prepare message to all replicas.

2. **Commit.** Each replica receives the Prepare message and broadcasts a Commit message.

   - A replica accepts a request if it collects a commit certificate consisting of $f + 1$ Commit messages.
   - If a replica does not hear back from the primary in time, it will send a ReqViewChange message.
   - If a replica receives $f + 1$ ReqViewChange messages, it transitions to the next view and sends the ViewChange message.

---

**A. Background**

The BFT model allows malicious replicas to behave arbitrarily. Corrupt replicas can stop sending messages to one or more replicas, or send conflicting messages (equivocate) to different replicas with an intention to break safety. Preventing equivocation can reduce the number of replicas and size of quorums required to reach agreement [13], [27], [28]. This is accomplished in the Hybrid fault model using a trusted component. The algorithm hosted in the trusted component attests messages in such a way to prove their uniqueness. A simple monotonically increasing counter can be used for this purpose [27]. By assigning a unique counter value per message and signing it, the trusted component ensures that the replica hosting it cannot send different messages with the same counter value. This property allows correct replicas to detect an attempt to send conflicting statements without requiring any additional communication mechanisms.

**B. Revisiting MinBFT**

MinBFT [3] is a Hybrid fault tolerant protocol that uses a trusted service to require only $N = 2f + 1$ replicas to tolerate $f$ byzantine faults. The trusted service prevents malicious replicas from equivocating to correct replicas, providing an efficient solution to the consensus problem under the Hybrid fault model.

**USIG Trusted Service:** The protocol uses a trusted service called the Universal Sequential Identifier Generator (USIG) that is present in each replica and provides two interfaces: one for signing and another for verifying messages. The USIG service assigns monotonically increasing counter values to messages and signs them. The service provides the following properties: (i) Uniqueness: no two messages are assigned the same identifier; (ii) Monotonicity: a message is never assigned an identifier smaller than the previous one; and (iii) Sequentiality: the next counter value generated is always one more than the last generated value.

To access its service, USIG provides two interfaces:

- **CreateUI($i$, $m$)** creates a signed certificate $UI_i$ for message $m$ with the next value from the monotonic counter. The certificate is computed using the private key of the USIG instance $r$, $UI_i = \langle ctr, H(m)\rangle_r$, where $ctr$ is the counter value and $H(m)$ is the hash of the message.
- **VerifyUI($UI_i, m$)** uses the USIG instance $i$’s public key and verifies whether the certificate $UI_i$ was computed for message $m$.

**MinBFT:** The MinBFT protocol proceeds in a sequence of views. The primary for each view is replica $r_i$ where $i = v \mod N$, $v$ is the view number and $N$ is the system size. The primary is responsible for handling client requests, assigning sequence number to those requests, and forwarding the requests to the replicas. The sequence numbers the primary assigns to requests is generated by the USIG service instance within the primary. The replicas accept the request and execute it once they collect a commit certificate. A commit certificate indicates that a majority of replicas have observed the same message from the primary.

When the primary receives a client request $m$ with operation $o$, it assigns a sequence number to the request. The sequence number is the one generated by the USIG service. The primary $r_i$ sends the request in a message $\langle\text{Prepare}, v, r_i, m, UI_i\rangle$, where $UI_i$ contains the unique sequence number and the signature obtained from the USIG module. Each replica $r_j$ in turn sends the $\langle\text{Commit}, v, r_j, r_i, m, UI_i, UI_j\rangle$ message to all other replicas. A client request is accepted at a replica if it receives $f + 1$ valid Commit messages, called a commit certificate.

Correct replicas only responds to the primary’s Prepare message if the following conditions hold: $v$ is the current view number and the sender of the message is the primary of $v$; the USIG signature is valid; and that the messages are received in sequential order of the USIG counter value. To prevent a faulty replica from executing the same operation twice, each replica maintains a $V_{req}$ to store the request identifier of the latest operation executed for each client. The messages are always processed in the order of the USIG sequence number to prevent duplication of operations and holes in the sequence number space. Replicas only execute an operation if it has not been executed already.

The view change protocol is triggered if the current primary fails to make timely progress. Replica sends a $\langle\text{ReqViewChange}, r_i, v, v'\rangle$ message to other replicas if it times out waiting for messages from the primary. A
C. MinBFT with Flexible Quorums

In this section, we introduce the notion of Flexible Hybrid Quorums. First, we show that not all quorums need to intersect and consequently show that the system size $N$ need not be a function of $f$. Specifically, we show that only quorums of different kinds must intersect. Thus, sizes of commit quorums $Q_c$ can be reduced at the cost of increasing the sizes of the view change quorums $Q_{vc}$. We apply this technique to MinBFT and call the resulting protocol as Flexible MinBFT.

MinBFT uses simple majority quorums for both the commit and the new view certificates. Thus, every quorum intersects with every other quorum. Consequently, commit quorums $Q_c$ intersect with other commit quorums. However, this is excessive. In MinBFT, the replica at the intersection of any two commit quorums, ensures that an operation is assigned only one $UI$ certificate. Note that the primary’s USIG service already ensures that an $UI$ is assigned only once. If the primary and the intersecting replica are malicious, the replica may still vote for the same operation at two different $UI$s. Correct replicas will handle this using the $V_{req}$ data structure. Since they process the messages in $UI$ order, they will observe that an operation has already been executed and not execute it again. This makes the intersection replica redundant. Thus, we relax the assumption that the different commit quorums intersect with each other. At the same time, to tolerate $f$ failures, the commit quorums should consists of more than $f$ replicas. Thus, we have that $|Q_c| > f$.

On the other hand, any view change quorum $Q_{vc}$ must intersect with any commit and view change quorums to ensure that the decisions made within a view are safely transitioned to future views. Hence, we have that $|Q_{vc}|$ + $|Q_c| > N$.

The Flexible Hybrid Quorum requirement is captured by the following equations:

1. $|Q_c| > f$
2. $|Q_{vc}| + |Q_c| > N$

By setting $Q_c$ to the smallest possible value i.e. $|Q_c| = f + 1$, we can observe that $|Q_{vc}|$ should equal $N - f$, to satisfy Equation 2. Consequently, the system size $N$ need not be a function of $f$.

We applied the Flexible Hybrid Quorums to MinBFT. The resulting algorithm, Flexible MinBFT remains largely the same, except for the quorums they use. First, the commit quorums size still remains the same $Q_c = f + 1$, except that the variable $f$, the number of tolerated faults, is independent of $N$, the system size. Second, the size of view change quorums $Q_{vc}$ now equals $N - f$ instead of $f + 1$. The protocol does not require any other changes.

The safety and liveness guarantees provided by Flexible MinBFT are given below. We only present the intuition and the related lemmas here. The complete proof is presented in Appendix A.

- **Safety within a view.** This is ensured by the trusted subsystem and the commit quorums. The trusted subsystem ensures non-equivocation, so a byzantine replica cannot send conflicting proposals. Correct replicas will only vote on the proposed operation if the proposal is valid and it has not voted for the same operation before. The following Lemma formalizes this notion.

  Lemma 5: In a view $v$, if a correct replica executes an operation $o$ with sequence number $i$, no correct replica will execute $o$ with sequence number $i' \neq i$.

- **Safety across views** This is ensured by the trusted subsystem and the view change quorums. The intersection of the commit and the view change quorum consists of at least one replica. Thanks to the trusted component, the replica in the intersection cannot equivocate, and must reveal the correct sequence of operations executed, as otherwise there will be holes that correct replicas can detect. Thus, a correct primary will gather the correct sequence and apply it in the next view.

  Lemma 6: If a correct replica executes an operation $o$ with sequence number $i$ in a view $v$, no correct replica will execute $o$ with sequence number $i' \neq i$ in any view $v' > v$.

- **Quorum availability with in a view.** A non-faulty primary will always receive responses from a quorum of $f + 1$ replicas and this quorum will contain at least one honest replica.

  Lemma 7: During a stable view, an operation requested by a correct client completes.
IV. DuoBFT

DuoBFT requires $N = 3f + 1$ replicas to tolerate $f$ faults. Learners can choose to tolerate either $f$ byzantine faults or $f$ hybrid faults by assuming the respective fault model. For ease of exposition, we use $f_B$ to denote byzantine faults and $f_H$ to denote hybrid faults, but $f_B = f_H = f$. Learners with different assumptions can coexist at the same time and they can also choose to update their assumptions at any time. To tolerate hybrid faults, at least $f_H + 1$ replicas must have the USIG trusted subsystem. We call these *hybrid replicas*. Learners can commit in the hybrid fault model by collecting votes from hybrid replicas only. However, to commit in the BFT model, learners can collect votes from any quorum of replicas including hybrid replicas. This flexible scheme allows learners assuming the hybrid model to make progress even when a trusted quorum is not possible by simply adjusting their fault assumptions. This enables DuoBFT to use as few hybrid replicas as possible to reach a hybrid quorum.

In the rest of this section, we will present the DuoBFT, overview its properties, and discuss its guarantees. Similar to other recent works such as Casper [12], Hotstuff [29], and Flexible BFT [22], we explain DuoBFT in terms of a blockchain protocol where the votes are pipelined. First, we present the terminologies.

A. Preliminaries

1) **Blockchain**: As presented in the previous sections, in classical BFT protocols, the agreement happens on a sequence of client requests or more generally values. Similarly, in a blockchain protocol, the agreement happens on a chain of blocks. We use the term *block* to refer to a value in the chain (or sequence). Each block has a reference to a predecessor block except the first block in the chain, also called the genesis block, which has no predecessors. Every block has a height parameter that indicates its position from the genesis block. A block $B_k$ at height $k$ has the following format: $B_k := (b_k, h_{k-1})$ where $b_k$ refers to the block’s value and $h_{k-1} = H(B_{k-1})$, the hash of the predecessor block in the chain. For the genesis block, the predecessor hash is null, thus $B_1 := (b_1, \perp)$. Note that only the genesis block can have a null predecessor hash; other block must specify a valid hash.

2) **Block Prefix and Equivocation**: Let $S$ be a sequence of blocks in increasing height order. The prefix of a Block $B_k$ at height $k$ in sequence $S$, denoted $\text{prefix}(S, k)$, is the prefix of the sequence $S$ containing the first $k$ blocks from the genesis block. Equivocation happens when the sequence of blocks diverges. Given two blocks $B_i$ and $B_j$, we say those blocks diverge when $B_i$ is not the ancestor of $B_j$ or vice versa.

3) **Block Certificates**: Replicas vote on the blocks by signing on the hash of the block $B_k$ when $B_k$ is the current view. Similar to MinBFT, DuoBFT replicas only process blocks in increasing height order. When blocks are received out of order, replicas wait to receive the predecessor blocks to $B_k$ and validates the blocks, casts its vote in the height order. If $R$ is a hybrid replica, it creates an $UI$ certificate for its vote; otherwise, it simply signs the vote. Replica $R$ sends its vote in a $\langle \text{Vote}, v, R, B_k, s_P, s_R \rangle$ message to other replicas.

The votes collected by a replica can form up to two kinds of quorum certificates for the block $B_k$ depending on whether or not the primary is a hybrid replica. Every replica records the following information for a block:

- $C_H(B_k)$: A set of $f + 1$ votes with $UI$ certificates form a hybrid quorum certificate for block $B_k$. Note that this certificate can be collected only if the primary is a hybrid replica.

B. Protocol

The DuoBFT protocol proceeds in a view by view fashion. For simplicity of prose, the primary of each view is decided using the formula $v \mod N$ i.e. primary roles are assigned to replicas in round-robin order. The primary of the view is responsible for proposing blocks that other replicas vote on. While the primary can be chosen using any known techniques [30], care should be taken to ensure that hybrid replicas are preferred to enable commit under both fault models. If the primary is not a hybrid replica, hybrid commits are not possible. We discuss methods for primary selection in Section V.

At a high level, DuoBFT works as follows. The primary proposes a block to replicas. Replicas vote on the block if it is safe to do so. A quorum of such votes on a block make a quorum certificate. After collecting a quorum certificate for a block, the primary moves on to propose the next block extending the previous one. We will discuss how the client commits later since they are the ones that make the actual commit decisions. Replicas use the view change protocol to install a new view if they are unable to make progress in the current view. The view change protocol begin only if $f + 1$ replicas request a view change.

1) **Normal protocol**: The normal protocol is executed when a view is stable. In a stable view $v$, the primary and $N - f$ replicas behave correctly and exchange messages in a timely manner. The primary creates a new block $B_k$ that extends the highest block in the chain it is aware of. If the primary is a hybrid replica i.e. it hosts the USIG component, it creates an $UI$ certificate for the block. Otherwise, it simply signs it. The primary $P$ sends the block to the replicas in a $\langle \text{Propose}, v, P, B_k, s \rangle$ message, where $v$ is the current view and $s$ is either the $UI$ or the signature.

A replica $R$ that receives the $\text{Propose}$ message for block $B_k$, votes on the block if it extends the previously proposed block in the view. Similar to MinBFT, DuoBFT replicas only process blocks in increasing height order. When blocks are received out of order, replicas wait to receive the predecessor blocks to $B_k$ and validates the blocks, casts its vote in the height order. If $R$ is a hybrid replica, it creates an $UI$ certificate for its vote; otherwise, it simply signs the vote. Replica $R$ sends its vote in a $\langle \text{Vote}, v, R, B_k, s_P, s_R \rangle$ message to other replicas.

The votes collected by a replica can form up to two kinds of quorum certificates for the block $B_k$ depending on whether or not the primary is a hybrid replica.
A DuoBFT replica executes the following protocol:

**Normal Protocol**

1) **Propose.** The primary $P$ creates a block and sends it in a $\langle \text{Propose}, v, P, B_k, s_P \rangle$ message to all replicas. It attests the block with a USIG identifier or a signature depending on its capability. The primary collects votes for the blocks from the replicas. The votes are either attested with a $UI$ certificate or a signature. The primary sends the next block when it receives a quorum certificate for its previous block.

2) **Vote.** A replica $R$ receives the propose message $\langle \text{Propose}, v, P, B_k, s_P \rangle$ from the primary, validates if it extends the last proposed block, and votes for it. The vote is sent in a $\langle \text{vote}, v, R, B_k, s_P, s_R \rangle$ message to other replicas. A $UI$ certificate or a signature is attached depending on the replica.

In addition, Replica $R$ records the following information for a block:
- $q_v(B_k)$: The votes received for block $B_k$ by replica $R$ from any other replica in view $v$.
- $C_H(B_k)$: A set of $f+1$ votes with $UI$ certificates form a hybrid quorum certificate for block $B_k$.
- $C_B(B_k)$: A set of $2f+1$ votes form a BFT quorum certificate for block $B_k$.

**View Change Protocol**

1) **View Change Request** A replica requests a view change if it does not receive proposals from the replicas in a timely manner, or if it observes equivocating blocks either via the proposal or the vote messages.
   - Replica $R$ sends a $\langle \text{ReqViewChange}, R, v, v' \rangle$ to request a view change from $v$ to $v' = v + 1$.
   - A replica that receives $f+1$ $\text{ReqViewChange}$ messages transitions to the new view and multicasts the $\text{ViewChange}$ message to other replicas.
   - A replica that receives $f+1$ $\text{ViewChange}$ message also transitions to the new view and sends its $\text{ViewChange}$ message to other replicas. The View Change message consists of all the blocks that the replicas have a quorum certificate for.

2) **New View.** The new primary $P'$ collects $2f+1 \text{ViewChange}$ messages and computes the sequence of blocks in the new view $v'$. It sends a $\langle \text{NewView} \rangle$ message to all replicas.

3) **New View Install.** A replica that receives the new primary $P'$’s $\text{NewView}$ messages, validates it, and installs $S$, the block sequence in the new view.

**Fig. 2. DuoBFT Protocol Execution**

**1) Hybrid Commit Rule:** A learner assuming the Hybrid Commit Rule commits a block $B_k$ iff it collects Hybrid quorum certificate for block $B_k$ i.e. $C_H(B_k)$.

**2) BFT Commit Rule:** A learner assuming the BFT Commit Rule commits a block $B_k$ iff it collects BFT quorum certificates for the blocks $B_k$ and $B_{k+1}$ i.e. $C_B(B_k)$ and $C_B(B_{k+1})$, and $B_{k+1}$ extends $B_k$.

**Fig. 3. DuoBFT Commit Rules**

- $C_B(B_k)$: A set of $2f+1$ votes form a BFT quorum certificate for block $B_k$. Since there are $f+1$ hybrid replicas, there will be at least one hybrid replica in any BFT quorum and that replica will produce a $UI$ certificate. Thus, the BFT quorum certificate will contain both $UI$s and signatures.

2) **View Change:** If a replica detects equivocation or lack of progress by the primary, it will start the view change procedure to move from the current view $v$ to the next stable view $v'$. A replica requests a view change by sending a $\langle \text{ReqViewChange}, R, v, v' \rangle$ message to other replicas. When a replica receives at least $f+1$ $\text{ReqViewChange}$ messages, it starts the view transition and sends a $\langle \text{ViewChange}, R, v', O, UI_i \rangle$ message, where $O$ contains the sequence of blocks for which $R$ has collected any quorum certificate. The primary $P'$ of the new view $v'$ will collect $2f+1$ valid $\text{ViewChange}$ messages to form the new view certificate. $P'$ will use this certificate to compute the set of blocks $S$ for which quorum certificates exist. $P'$ sends a $\langle \text{NewView}, r_i, v', V_{vc}, S, UI_i \rangle$, where $V_{vc}$ is the new view certificate that contains the set of $f+1$ $\text{ViewChange}$ messages used to construct the new view and $S$ is the set of prepared or committed requests. Replicas verify the validity of the $S$ by performing the same computation as the new primary using the new view certificate. Then, replicas adjust their local state according to $S$ and start voting in the new view $v'$.

The new view computation performed by the replicas and the primary is similar to the one presented in PBFT [23]. We omit the details for conciseness.

**C. Learners**

DuoBFT allows different learners to decide differently on the commitment status of the block based on how they perceive the system. The protocol supports two kinds of commit rules depending on whether the learner chooses the byzantine fault
model or the hybrid fault model.

1) Hybrid Commit Rule: A learner can choose to commit blocks adopting the hybrid fault model. In this case, a learner commits a block when it receives at least $f + 1$ votes from hybrid replicas. Note that the number of hybrid replicas can be capped at $f + 1$ for a system with $N = 3f + 1$ total replicas. Moreover, note that a learner can only decide to commit on a block only if there are enough hybrid replicas in the system. Otherwise, such a learner will never terminate and must recalibrate its assumptions and use the byzantine fault model instead.

Under the Hybrid Commit Rule, the protocol provides the same safety guarantees as Flexible MinBFT.

2) BFT Commit Rule: Alternatively, a learner can choose to commit blocks adopting the byzantine fault model. In such a case, the learner commits a block when it receives at least $2f + 1$ votes for the block $B_l$ and the consecutive block.

Under the BFT Commit Rule, the protocol provides the same safety guarantees as PBFT.

D. Proof

Lemma 1. If a learner commits a block $B_l$ in a view $v$, then no learner with the same assumptions will commit $B'_l$ that does not equal $B_l$ in view $v$.

Proof. BFT Commit Rule: We prove by contradiction. Say a learner commits block $B_l$. It will have $q_c$ votes for $B_l$ and its immediate successor. Suppose another learner commits block $B'_l$ then it will have $q_c$ votes for $B'_l$ and its immediate successor. However, the intersection of two $q_c$ quorums will have at least one correct replica that will not vote for two blocks at the same height. This is a contradiction.

Hybrid Commit Rule: We prove by contradiction. Say a learner commits block $B_l$. It will have $f + 1$ USIG votes for $B_l$. Suppose another learner commits block $B'_l$ then it will have $f + 1$ USIG votes for $B'_l$. However, a primary cannot sign two messages with the same USIG identifier. Thus, there is no way there can exist two blocks at the same height $l$. This is a contradiction.

Thus, it is not possible for any two learners to commit different blocks at the same height in view $v$.

Lemma 2. If a learner commits a block $B_l$ in a view $v$, no learner with the same assumptions will commit block $B'_l$ that does not equal $B_l$ at the same height $l$ in any view $v' > v$.

Proof. BFT Commit Rule: We prove by contradiction. A learner commits block $B_l$ in view $v$ then it will have $q_c$ votes for $B_l$ and its immediate successor. Suppose another learner commits block $B'_l$ in view $v' > v$ then it should have $q_c$ votes for $B'_l$ and its immediate successor.

Since $B_l$ is committed in view $v$, there should exist quorum certificates for blocks $B_l$ and $B_{l+1}$. In the new view $v' = v + 1$, the NewView message sent by the new primary includes a NewView certificate with $q_c$ ViewChange messages that contains at least one correct replica. The ViewChange message from the correct replica will have the correct certificate for $B_l$ and $B_{l+1}$. Thus, the new primary must enforce blocks $B_l$ and $B_{l+1}$ in the new view $v + 1$. It will receive votes only for $B_l$ in the new view $v + 1$. For $B'_l$ to be committed, $q_c$ replicas must vote for it, which cannot happen since there is at least one correct replica in the intersection of $q_c$ and $q_c$ that received the NewView message with correct certificates. This is a contradiction.

Thus, $B'_l$ cannot have been committed in $v + 1$.

Hybrid Commit Rule: We prove by contradiction. A learner commits block $B_l$ in view $v$ then it will have $f + 1$ votes for $B_l$. Suppose another learner commits block $B'_l$ in view $v' > v$ then it should have $f + 1$ votes for $B'_l$.

Since $B_l$ is committed in view $v$, there should exist a USIG quorum certificate for block $B_l$. In the new view $v' = v + 1$, the NewView message sent by the new primary includes a new view certificate with $N - f$ ViewChange messages that contains at least one correct replica. The correct replica's ViewChange message will have the correct quorum certificate for $B_l$. It might happen that the new primary might remove some block entries from the ViewChange message, but this will be detected as the USIG-signed NewView message will reveal the holes in the message log (See Lemma 8 for additional details.)

Thus, the new primary must enforce blocks $B_l$ and $B_{l+1}$ in the new view $v + 1$. It will receive votes only for $B_l$ in the new view $v + 1$. For $B'_l$ to be committed, $q_c$ replicas must vote for it, which cannot happen since there is at least one correct replica in the intersection of $q_c$ and $q_c$ that received the NewView message with correct certificates. This is a contradiction.

Thus, $B'_l$ cannot have been committed in $v + 1$.

For both commit rules above, the case for arbitrary $v' > v$ where $v' = v + k$ will fall under the case of $v + 1$, since at each view transition, the information from one view is propagated to the next view.

Theorem 1. Any two learners with the same commit rule commit the same sequence of blocks in the same order.

Proof. To elaborate on the theorem, if a learner following a commit rule commits the sequence of blocks $S = \langle B_1, \ldots, B_i \rangle$, then another learner that follows the same commit rule will commit the same sequence of blocks $S$ or a prefix of it. We use $\text{prefix}(S, i)$ to represent the first $i$ blocks of the sequence $S$. We use the • operator to concatenate any two sequences.

Assume the theorem is false i.e. there should exist two sequences $S$ and $S'$ committed by two learners that is not a prefix of each other. Assume the sequences conflict at $i$, such that $\text{prefix}(S, i) = \text{prefix}(S', i - 1) \bullet (B_l)$ and $\text{prefix}(S', i) = \text{prefix}(S, i - 1) \bullet (B'_l)$. Precisely, there exists two blocks $B_l$ and $B'_l$ at the same height $i$ committed by two different learners
with the same commit rule. Assume that block $B_i$ was committed in view $v$ and block $B_i'$ was committed in view $v'$. If $v = v'$, then this will contradict Lemma $\text{1}$. If $v' > v$, then this will contradict Lemma $\text{2}$. Hence, the theorem must hold. 

Lemma 3. During a stable view, a proposed block is committed by a learner.

Proof. In a stable view, the correct primary will propose blocks in a timely fashion. If the primary is hybrid, then it will generate an $UI = (i, H(b))_p$ for the block. Correct replicas that receive the proposal will vote for it. Replicas that are hybrid will generate an $UI$ for their votes. Since there are at most $f$ faulty replicas, they will remain $N - f$ correct ones. For a hybrid quorum, at least $f + 1$ of these $N - f$ replicas will reply on time. Similarly, for a BFT quorum $N - f = 2f + 1$ replicas will reply on time. Thus, a learner will receive the votes on time and will commit the block using their commit rule.

Lemma 4. A view $v$ will eventually transitioned to a new view $v' > v$ if at least $N - f$ replicas request for it.

Proof. A replica $R$ can request a view change by sending a $\langle \text{ReqViewChange}, R, v, v' \rangle$ message. The view change mechanism is triggered when replicas receive $f + 1$ $\text{ReqViewChange}$ messages for the same view. Assume that replicas collect $f + 1$ messages for transitioning from $v$ to $v + 1$. The primary for the new view is $(v + 1) \mod N$ by definition. Consider the two cases:

1) the new view is stable: correct replicas will receive the $\text{ReqViewChange}$ messages. Consequently, correct replicas that receive at least $f + 1$ $\text{ReqViewChange}$ messages will enter the new view $v'$ and send a $\text{ViewChange}$ message to all replicas. The primary $p$, being stable, for view $v + 1$ will send a valid $\text{NewView}$ message in time. Thus, correct replica that receive the message will transition to new view $v' = v + 1$.

2) the new view is not stable: We consider two cases:

a) the primary $p$ is faulty and does not send the $\text{NewView}$ message in time, or $p$ is faulty and sends an invalid $\text{NewView}$ message, or $p$ is not faulty but the network delays $p$’s message indefinitely. In all these cases, the timer on other correct replicas that sent the $\text{ViewChange}$ message will expire waiting for the $\text{NewView}$ message. These replicas will trigger another view change to view $v + 2$.

b) the primary $p$ is faulty and sends the $\text{NewView}$ message to only a quorum $Q'$ of $q_{vc}$ replicas but less than $q_{vc}$ replicas are correct, or $p$ is correct but there are communication delays. The replicas in quorum $Q'$ may enter the new view and process requests in time. However, the correct replicas that does not receive the $\text{NewView}$ message will timeout and request change to view $v + 2$. However, there will be less than $f + 1$ replicas, so a successful view change trigger will not happen. If the faulty replicas deviate from the algorithm, other correct replicas will join to change the view.

Theorem 2. A proposed block is eventually learnt by learners with correct commit rules.

Proof. When the view is stable, Lemma $\text{3}$ shows that the proposed block is learnt by the learners. When the view is not stable and the replica timers expire properly, $f + 1$ replicas will request a view change. By Lemma $\text{4}$ a new view $v'$ will be installed.

However, if less than $f + 1$ replicas request the view change, then the remaining replicas that do not request the view change will follow the protocol properly. Thus, the system will stay in view $v$ and the learners will continue to commit blocks in the view. When proposals are not committed in time or when more than $f$ replicas request a view change, then all correct replicas will request a view change and it will be processed as in Lemma $\text{3}$.

Even after a view change, the new view $v'$ may not necessarily be stable. If the new primary deviates from the algorithm or does not process messages in time, this will cause correct replicas to request another view change and move to the next view. Since there can only be at most $f$ faulty replicas, after at most $f + 1$ view changes, a stable view will be installed. Furthermore, if the faulty primary follows the algorithm enough such that a view change cannot be triggered, by Lemma $\text{3}$ learners will continue to commit the blocks.

E. Optimizations

Reducing Message Complexity. Our presentation of DuoBFT closely follows the message pattern of the signature-based PBFT protocol $\text{23}$. Therefore, we assume that the replicas multicast their votes to all other replicas incurring a $O(n^2)$ normal case message complexity. The complexity can be reduced to $O(n)$ by modifying the replicas to send their votes only to the primary, and enabling the primary to collect the votes and share the quorum certificate with the replicas. The primary can also piggyback the quorum certificate along with the proposal for the next block. This is a known technique adopted by many existing protocols $\text{12, 22}$.

Choice of Primary Replica. The hybrid commit rule requires that blocks are proposed and voted for by hybrid replicas. Since there are only $f + 1$ hybrid replicas out of $3f + 1$ replicas, the round-robin primary assignment can install a non-hybrid replica as the primary. In this case, commits under the hybrid commit rule is no longer possible until another view change is triggered and a hybrid primary replica is installed. To favor hybrid replicas, we propose using a blacklisting mechanism similar to the one presented in $\text{31}$. Every replica maintains a blacklist that initially consists of all non-hybrid replicas. The
blacklist has a maximum size of $N - f$ and the items are arranged in FIFO order. When an item is added to a full blacklist, it removes the oldest item from it. When a correct replica requests a view change, it will choose the next view such that the new primary is not in the blacklist. However, to ensure progress at times when hybrid replicas cannot establish a stable view, we remove replicas from the blacklist. When a view change happens, the old primary is added to the blacklist and the least recently added replica is removed from the blacklist.

V. Discussion

Implications of Trusted Environment Compromises. As mentioned in the introduction, trusted execution environments are increasingly being scrutinized for security vulnerabilities. In the hybrid fault model, the compromise of the trusted component is enough to break the safety of the protocol. However, DuoBFT holds safety in such cases. Note that there are only $f + 1$ hybrid replicas. Even if the trusted component is compromised, per our assumption, this can only affect at most $f$ hybrid replicas. Thus, the remaining $2f + 1$ replicas will follow the algorithm correctly. While the hybrid quorum certificates can become invalid, recall that replicas also collect BFT quorum certificates. Thus, safety is still preserved for the sequence of blocks that have collected the BFT quorum certificates.

Comparison to FlexibleBFT. We now highlight the differences of our protocol from Flexible BFT [22], a recent protocol that provides diverse learner assumptions. The first important distinction is that Flexible BFT provides the a-b-c fault model in addition to the BFT model. The replicas under the a-b-c model are allowed to attack the safety of the system, but when they aren’t able to attack safety, they will ensure liveness. However, the implication of using this fault model is that Flexible BFT quorums are much larger than our flexible hybrid quorums. In Flexible BFT, the commit quorums used by the client $q_c$ should be at least as large as the view change quorum $q_v$ used by the replicas, i.e $q_c \geq q_v$. In contrast, DuoBFT uses hybrid commit quorums that are smaller than the view change quorums, and the BFT commit quorums are as large as the view change quorums. That is, in DuoBFT, $q_c \leq q_v$.

Furthermore, Flexible BFT uses the synchrony timing model as a means to provide commits using simple majority quorums, which are smaller than byzantine quorums. The protocol also tolerate $< 1/2$ failures. On the other hand, DuoBFT tolerates only $1/3$ failures, but under the hybrid model, its commit quorums sizes are really efficient, only a little over $1/3$ replicas. Thus, Flexible BFT uses the timing model to reduce quorum sizes, while DuoBFT uses the trusted component to achieve a very similar purpose. Furthermore, partially synchrony model enables “network-speed” replicas those that do not need lock-step executions unlike in the synchrony model. Thus, assumptions such as globally synchronized clocks are not required in our case.

VI. Related Work

Since Lamport et al. formulated the Byzantine Generals problem [32], numerous solutions have been proposed to solve the agreement problem in the face of byzantine failures. These solutions have varied widely in terms of their fault assumptions and the timing models. A review of these solutions is beyond the scope of the paper. We defer the interested readers to books on distributed computing [25], [33].

The literature is rich in protocols that adopt the hybrid fault model [34]. While early protocols depended on an attested append-only log abstraction provided by the trusted component [13], the counter-based abstraction [27] became widely adopted due to its simplicity, and have been adopted by numerous protocols [8], [14], [15], [34], [35]. We used MinBFT to perform our analysis and construction, because of its presentation as the hybrid counterpart to PBFT, and its use of a simple counter-based trusted attestation mechanism.

Speculation. Some partially-synchronous BFT protocols [7], [10], [36], [37] use speculation to make commit decisions using fewer communication steps. These protocol adopt fast BFT quorums that are at least $50\%$ larger than normal BFT quorums, and reduce communication delays under favorable conditions. At times, a fast quorum may not be enough or unattainable, in which case, the replicas fall back to a slow protocol with normal quorums and additional communication steps. Thus, such protocol collect larger quorum than normal protocols in the best case, and spend more communication steps than normal protocols in the worst case. DuoBFT’s provides an cheaper and stronger alternative to these solutions. The hybrid commit rule can be used to make commit decisions on a block by collecting votes from quorums of $1/3$ replicas. To fall back to the BFT commit rule, a learner only needs to collect additional $1/3$ votes for the block and $2/3$ votes for the next block. In total, a learner does not need to collect more votes or spend more communication steps than that in normal BFT protocols.

Similarly, Thunderella [38] and Sync Hotstuff [39] commit optimistically under the partial synchrony model using quorums of size $\geq 3/4$, and fallback to a synchronous slow commit rule. DuoBFT’s guarantees are completely based in the partial synchrony model.

Tentative Execution and Optimistic Agreement. Some protocols like PBFT and MinBFT use tentative execution [23] to execute the proposed operations before the final commit step. This can improve the overall performance under favorable conditions. Furthermore, the optimistic agreement [40] technique uses only a subset of replicas to run agreement, while the remaining replicas update their state passively. Such techniques are orthogonal and applicable to DuoBFT as well.
Hierarchical Protocols. Steward [9, 41] and GeoBFT [42] follow the hierarchical fault model by using a combination of crash and byzantine fault tolerant mechanisms. The replicas are divided into groups. Replicas with a group run a BFT protocol while inter-group agreement is achieved using a crash-fault tolerant protocol. However, the protocol exposes a single combined fault model to the learners: the protocols can tolerate $f_z$ failures in $z$ groups of which at most $f$ can happen in a single group. Such techniques are aimed towards WAN deployments.

Flexible Quorums. In Flexible Paxos [43], Howard et al. introduced the notion of flexible quorums in the crash fault model. Malkhi et al. then developed the flexible quorums approach in the Byzantine fault model [22]. The Flexible Hybrid Quorums presented in this paper can be seen as the hybrid variant of the flexible quorums technique. DuoBFT adds support for the BFT fault model over the hybrid fault model and exposes the choice to the learners.

Adversaries. Different kinds of adversaries have been explored in prior works. Both the BAR and a-b-c fault models [2, 22] consider an adversary that does not collude. With the hybrid fault model, we consider an adversary that does not break the protections around the trusted component, but they can otherwise collude with other byzantine replicas.

Diverse learners. Bitcoin [24] uses a probabilistic commit rule that depends on the depth of the confirmation. Typically a block depth of six implies a commit with a very high probability, although a block depth of one is enough to commit for some learners. The Cross fault tolerant (XFT) [4] model offers two kinds of learners: learners that follow the crash fault model under the asynchronous timing model, or learners that from the byzantine fault model under the synchronous timing model.

VII. CONCLUSION AND FUTURE WORK

In this paper, we present DuoBFT, a protocol that provides two fault models under the partial synchrony setting and lets learners make commit decisions under either of them. We show that a fraction of hybrid replicas is enough to enable commit decisions under the hybrid fault model and still preserve safety guarantees. Furthermore, the incorporation of the flexible hybrid quorum technique is what makes the common view change protocol that supports both the fault models feasible. This also simplifies the overall protocol.

DuoBFT uses a fixed value for $f$, the number of tolerated faults. In Flexible BFT, the a-b-c fault model allows learners to choose different values of $f$, within constraints. Introducing variable $f$ to DuoBFT either by incorporating the a-b-c fault model or exploring other means remains interesting future work. Moreover, it is interesting to understand the performance characteristics of the dual fault model offered by DuoBFT by performing an evaluation study.

APPENDIX

A. Flexible MinBFT Correctness

Note that our proof structure overlaps with MinBFT’s original proof for ease of exposition.

Lemma 5. In a view $v$, if a correct replica executes an operation $o$ with sequence number $i$, no correct replica will execute $o$ with sequence number $i' \neq i$.

Proof. If $r$ executes $o$ with sequence number $i$, then it will have $f + 1$ valid Commit messages for $\langle o, i \rangle$ from a quorum $Q_c$.

We prove by contradiction. Suppose another correct server $r'$ execute $o$ with sequence number $i' > i$. This can happen if $s'$ received $f + 1$ valid Commit messages for $\langle o, i' \rangle$ from quorum $Q'_c$. Note that $\exists Q_c, Q'_c : Q_c \cap Q'_c = \emptyset$ i.e. any two $Q_c$ quorums need not intersect. There are two cases to consider depending on the primary:

1) Primary is correct: This is trivial since a correct primary will not generate two UIs for the same operation $o$.
2) Primary is faulty: Let's say $r$ sends $\langle \text{Commit}, v, r, s, UI_p, UI_r \rangle$ to $s$ for $\langle o, i \rangle$ and say $r'$ sends $\langle \text{Commit}, v, r', s', UI_p, UI_r \rangle$ to $s$ for $\langle o, i' \rangle$.
   a) $s'$ executed some operation at $i$: Primary cannot generate two messages with the same sequence number. Thus, $i$ must have been $o$. Since $o, seq \leq V_{req}[c]$, $o$ will not be executed again.
   b) $s'$ did not execute $i$: Replicas can only execute in sequence number order. Since $i < i'$, $s'$ must wait to execute $i$. Once its executes $i$, executing $i'$ fall under previous case.

Thus, it is not possible for any two replicas to execute the same operation with different sequence number in view $v$. □

Lemma 6. If a correct replica executes an operation $o$ with sequence number $i$ in a view $v$, no correct replica will execute $o$ with sequence number $i' \neq i$ in any view $v' > v$.

Proof. If a correct replica $s$ executes $o$ with sequence number $i$ in view $v$ it must have received at least $f + 1$ valid Commit messages for $\langle o, i, v \rangle$ from quorum $Q_c$ replicas.

Proof by contradiction: Let us suppose that another correct replica $s'$ executes $o$ at sequence number $i' > i$ in view $v' > v$, then $s'$ would have received $f + 1$ valid Commit messages for $\langle o, i', v' \rangle$ from quorum $Q'_c$ replicas.

Note that we have that any two commit quorums need not intersect i.e. $Q_c \cap Q'_c = \emptyset$.

We first deal with the case where $v' = v + 1$ and generalize later for arbitrary values of $v' > v$.

Let $p$ be the primary of new view $v'$. First, we show that the primary $p$ in view $v'$ cannot deny the fact that $o$ was accepted/executed before $v'$. Then, we show that no correct replica will execute $o$ with $i' \neq i$ in $v'$ and prove the contradiction.
The NewView message sent by the new primary includes the new view certificate $V_{vc}$ that contains $N - f$ ViewChange messages from a quorum $Q_{vc}$ that contains at least one correct replica, say $r \in Q_{vc}$ that will send the correct ViewChange message to the new primary. Given this, we consider the following cases:

1) **Primary is correct and replica $r \in Q_{vc}$ is correct:** If primary $p$ is correct, it inserts $N - f$ ViewChange messages into $V_{vc}$ including the one from $r$. There are two possibilities.

   a) $o$ was executed after checkpoint: $r$ is correct, so $O$ contains Commit that $r$ sent for $o$, therefore $V_{vc}$ and $S$ in the NewView message assert $O$ was executed explicitly.

   b) $o$ was executed before the checkpoint: The latest checkpoint $C_l$ shared by $r$ implies that $o$ was executed explicitly. Since the $V_{vc}$ sent in the NewView contains the $C_l$, it implies that $o$ was executed.

2) the primary $p$ is correct, but there is no correct replica in $Q_{vc}$ that executed $o$: There should exist at least a faulty replica $r \in Q'_c$ that accepted $o$ because $Q'_c \cap Q_{vc} \neq \emptyset$ ($q_{cnt} + q_{vc} > 1$).

   a) $o$ was executed after checkpoint: $r$ might be tempted to not include $o$'s messages in $O$, but if it did, $p$ being correct would not put $r$'s ViewChange message in $V_{vc}$ as $p$ can detect its invalid. There are two possible ways a detection will happen: (i) if $r$ executed a request $o'$ after $o$, $r$ might put the Commit message for $o'$ but not $o$ leaving a hole in the log that $p$ will detect. (ii) if $r$ sent Commit for $o$ with a USIG value $cv$, it might leave out all commit after $o$ with $cv'' > cv$ from the $O$ log. But, this log will also be considered invalid by $p$ since $r$ must sign the ViewChange message containing $O$ before sending it. The USIG will sign with a $cv'' > cv + 1$ that will allow a correct $p$ to detect an incomplete $O$. Thus, $r$ must include all commits and Case 1 above will apply.

   b) $o$ was executed before the stable checkpoint: One way this can happen is when $r$ includes an older checkpoint message but $p$ will detect the invalid ViewChange message because the USIG value of the message will disclose that there are messages since the checkpoint message that $r$ failed to disclose.

3) Primary is faulty but $r \in Q_{vc}$ is correct and executed $o$. In this case, the faulty primary $p$ may attempt to modify the contents of $O$ that it receives from $r$ before inserting into $V_{vc}$. However, this will leave a hole and other correct replicas will detect this misbehavior since they run the same procedure the primary runs for computing the NewView message. If $p$ removes $o$ and all further operations after $o$, correct replicas can also detect it because the USIG value of the $r$'s ViewChange message inside $V_{vc}$ will indicate the missing messages (as in Case 2). Similarly, if the primary tries to add an older checkpoint certificate, correct replicas will detect it from the holes in the USIG values. Therefore, a fault primary cannot tamper with a ViewChange message without detection. Thus, Case 1 will happen.

4) Primary is faulty and no correct $r \in Q_{vc}$ has executed $o$. A faulty $r \in Q_{vc}$ may exist. Given $|Q_{vc}| + |Q_{vc}| > N$, $r$ cannot successfully convince the primary to behave as if it did not execute $o$. Even if the primary being faulty uses $r$'s ViewChange message in $V_{vc}$, other replicas will detect the missing sequence number and the corresponding commit message for $o$. Thus, we will fall back to Cases 2 and 3 above.

The above four cases show that $p$ in view $v'$ must assert $O$ was executed before $v'$ in the certificate $V_{vc}$. Now, we show no correct replicas will execute $o$ with $i' \neq i$ in $v'$. There are two cases:

1) **Primary is correct:** A correct primary $p$ will never generate a second USIG certificate for the same operation and correct replicas will not send a commit message $o$ with $i'$ in view $v'$.

2) **Primary is faulty:** It is possible for a faulty primary to create a new Prepare message for $o$ and successfully create a new USIG $UI_p = (i, H(o))_p$ and send it to a replica $r$. However, every replica maintains the $V_{req}$ that holds the last executed operation identifier $seq$ for each client. Thus, $r$ will discover that $o$ was already executed since $seq \leq V_{req}[c]$. Thus, $o$ will not be executed again.

This proves that if a correct replica executed $o$ at sequence number $i$ in view $v$, then no correct replica will execute $o$ at sequence number $i' \neq i$ in view $v' = v + 1$.

We now generalize for arbitrary values of $v' > v$. There are two cases:

1) $v' = v + k$ but no request was accepted in view $v''$ such that $v' < v'' < v + k$: This case is trivial and falls under the case of $v' = v + 1$ since only view change related messages are sent in $v''$ which mirrors the $v$ to $v'$ transition.

2) $v' = v + k$ but requests were prepared/accepted in view $v''$ such that $v' < v'' < v + k$: At each view change, replicas must propagate information about operations from one view to its consecutive view (e.g. $v$ to $v + 1$, and so on). This is done either via the checkpoint certificate or the via the $O$ log set. Thus, each transition becomes the case of $v' = v + 1$ above.

\[ \square \]

**Theorem 3.** Let $s$ be a correct replica that executed more operations of all correct replicas up to a certain instant. If $s$ executed the sequence of operations $S = \langle o_1, \ldots, o_l \rangle$, then all other correct replicas executed this same sequence of operations or a prefix of it.

**Proof.** Let $prefix(S, k)$ be a function that gets the prefix of sequence $S$ containing the first $k$ operations, with $prefix(S, 0)$ being the empty sequence. Let $\bullet$ be an operation that concatenates sequences.
We prove by contradiction. Assume the theorem is false, i.e. there exists a correct replica \( r' \) that executed some sequence of operations \( S' \) that is not a prefix of \( S \). Let \( \text{prefix}(S, i) = \text{prefix}(S', i - 1) \cdot \langle o_i \rangle \) and \( \text{prefix}(S', i) = \text{prefix}(S, i - 1) \cdot \langle o'_i \rangle \) such that \( o_i \neq o'_i \). In this case, \( o_i \) was executed as the \( i \)th operation by replica \( r \) and \( o'_i \) was executed as the \( i \)th operation by replica \( r' \). Assume \( o_i \) was executed in view \( v \) and \( o'_i \) was executed in view \( v' \). Setting \( v = v' \) will contradict Lemma 5 and setting \( v' \neq v \) will contradict Lemma 6.

Lemma 7. During a stable view, an operation requested by a correct client completes.

Proof. A correct client \( c \) will send an operation \( o \) with an identifier larger than any previous identifiers to the replicas. The primary \( p \) being correct, will construct a valid \text{Prepare} message with a valid USIG certificate \( UI_p = \langle i, H(o) \rangle_p \) and send it to all replicas. At least \( f + 1 \) correct replicas will validate the \text{Prepare} message, verify the \( UI \), and send a corresponding \text{Commit} message. Since there can be only \( f \) faults, there should exist at least \( N - f \) correct replicas, out of which \( f + 1 \) (\( q_{\text{cmnt}} \)), should successfully produce these \text{Commit} messages. When a correct replica receives \( q_{\text{cmnt}} \) valid \text{Commit} messages, \( o \) will be executed and replied to the client \( c \). Since \( q_{\text{cmnt}} \) correct replicas exist, a correct client will receive \( f + 1 \) same replies indicating that operation \( o \) was properly executed at sequence number \( i \).

Lemma 8. A view \( v \) eventually will be changed to a new view \( v' > v \) if at least \( N - f \) correct replicas request its change.

Proof. A correct replica \( r \) sends a \( \langle \text{ReqViewChange}, r, v, v' \rangle \) message requesting a view change to all replicas. However, at least \( N - f \) correct replicas must send such a message to actually trigger a view change. Say a set of \( N - f \) correct replicas request a view change from \( v \) to \( v + 1 \) by sending the \text{ReqViewChange} message. The primary for the new view is \( p = (v + 1) \mod N \). Consider the two cases:

1) the new view is stable: correct replicas will receive the \text{ReqViewChange} messages. Consequently, correct replicas that receive at least \( N - f \) \text{ReqViewChange} messages will enter the new view \( v' \) and send a \text{ViewChange} message to all replicas. The primary \( p \), being stable, for view \( v + 1 \) will send a valid \text{NewView} message in time. Thus, correct replica that receive the message will transition to new view \( v' = v + 1 \).

2) the new view is not stable: We consider two cases:

a) the primary \( p \) is faulty and does not send the \text{NewView} message in time, or \( p \) is faulty and sends an invalid \text{NewView} message, or \( p \) is not faulty but the network delays \( p \)'s message indefinitely. In all these cases, the timer on other correct replicas that sent the \text{ViewChange} message will expire waiting for the new view message. These replicas will trigger another view change to view \( v + 2 \).

b) the primary \( p \) is faulty and sends the \text{NewView} message to only a quorum \( Q_{vc} \) of \( N - f \) replicas but less than \( N - f \) replicas are correct, or \( p \) is correct but there are communication delays. The replicas in quorum \( Q_{vc} \) may enter the new view and process requests in time. However, the correct replicas that does not receive the \text{NewView} message will timeout and request change to view \( v + 2 \). However, there will be less than \( N - f \) replicas, so a successful view change trigger will not happen. If the faulty replicas deviate from the algorithm, other correct replicas will join to change the view.

Theorem 4. An operation requested by a correct client eventually completes.

Proof. The proof follows from the Lemmas 7 and 8.

When the view is stable, Lemma 7 shows that the client operations are properly committed. However, when the view \( v \) is not stable, there are two possibilities:

1) at least \( f + 1 \) replicas timeout waiting for messages and request a view change: Lemma 8 handles this case and ensures that a stable view \( v' > v \) is established.

2) less than \( f + 1 \) replicas request a view change: There should exist at least a quorum \( Q \) of \( f + 1 \) replicas that are in the current view \( v \). As long as these replicas continue to follow the algorithm, they will continue to stay in view \( v \) and client requests will be committed in time. However, if the replicas are not timely, then the correct replica from \( Q \) in view \( v \) will send the \text{ReqViewChange} message. With this message, a successful view change is triggered and the previous case takes happens.

If the new view \( v' \) is not stable, another view change will be triggered depending on whether Cases 1 or 2 above holds. However, this process will not continue forever. Since there are only \( f \) byzantine replicas and due to the assumption that the network delays do not grow indefinitely, eventually there should exist a view \( v'' \) that is stable such that the primary responds in a timely manner and follows the algorithm.
[37] B. Arun, S. Peluso, and B. Ravindran, “ezbft: Decentralizing byzantine fault-tolerant state machine replication,” in 2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS), July 2019, pp. 565–577.

[38] R. Pass and E. Shi, “Thunderella: Blockchains with optimistic instant confirmation,” in Advances in Cryptology – EUROCRYPT 2018, ser. Lecture Notes in Computer Science, J. B. Nielsen and V. Rijmen, Eds. Springer International Publishing, 2018, p. 3–33.

[39] I. Abraham, D. Malkhi, K. Nayak, L. Ren, and M. Yin, “Sync hotstuff: Simple and practical synchronous state machine replication,” Cryptology ePrint Archive, Report 2019/270, 2019, https://eprint.iacr.org/2019/270

[40] T. Distler, C. Cachin, and R. Kapitza, “Resource-efficient byzantine fault tolerance,” IEEE Transactions on Computers, vol. 65, no. 9, pp. 2807–2819, Sep. 2016.

[41] Y. Amir, C. Danilov, J. Kirsch, J. Lane, D. Dolev, C. Nita-Rotaru, J. Olsen, and D. Zage, “Scaling byzantine fault-tolerant replication to wide area networks,” in International Conference on Dependable Systems and Networks (DSN'06), June 2006, pp. 105–114.

[42] S. Gupta, S. Rahnama, J. Hellings, and M. Sadoghi, “ResilientDB: Global scale resilient blockchain fabric,” Proc. VLDB Endow., vol. 13, no. 6, p. 868–883, Feb. 2020. [Online]. Available: https://doi.org/10.14778/3380750.3380757

[43] H. Howard, D. Malkhi, and A. Spiegelman, “Flexible Paxos: Quorum Intersection Revisited,” in 20th International Conference on Principles of Distributed Systems (OPODIS 2016), ser. Leibniz International Proceedings in Informatics (LIPIcs), P. Fatourou, E. Jiménez, and F. Pedone, Eds., vol. 70. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017, pp. 25:1–25:14. [Online]. Available: http://drops.dagstuhl.de/opus/volltexte/2017/7094