BumbleBee: Best-of-Both-Worlds MVBA with Optimal Communication, Latency and Resilience Tradeoffs

FATIMA ELSHEIMY, Yale University, USA
SIMON HOLMGAARD KAMP, Ruhr University Bochum, Germany

Consensus among n parties tolerating up to t Byzantine faults requires $n > 2t$ in synchronous networks and $n > 3t$ in asynchronous networks. The higher resilience achievable in synchrony relies on a known message delay bound Δ, whereas asynchronous protocols make no timing assumptions but must tolerate fewer faults. Prior work addressed this gap only partially. Some protocols achieve responsiveness under synchrony, meaning that their running time adapts to the *actual* network delay, but offer no guarantees under asynchrony, while others guarantee correctness under both network conditions but sacrifice responsiveness. Only recently, Elsheimy, Kamp, Loss, and Nielsen (IACR 2026) showed for binary validated Byzantine agreement (VBA) that if t_s, t_a, and t_r denote the synchronous, asynchronous, and responsiveness thresholds, respectively, then the conditions $n > 2t_s + t_a$ and $n > t_s + 2t_r$ are necessary and sufficient to simultaneously achieve asynchronous security, synchronous security, and responsiveness. While binary BA (or VBA) can be extended to multi-valued Byzantine agreement (MVBA) via standard reductions, such transformations generally incur blow-up in the communication. Whether these tight resilience conditions can be achieved for MVBA *with optimal communication complexity* remained open.

In this work, we resolve this question. For the aforementioned optimal thresholds, we construct an MVBA protocol that is asynchronously secure when $f \leq t_a$, synchronously secure when $f \leq t_s$, and responsive when $f \leq t_r$, where f is the actual number of corruptions. Our construction builds on Dumbo-MVBA (Lu et al., PODC 2020) and preserves asymptotically optimal efficiency. When $n - 2t_s = \Theta(n)$, our first construction achieves $O(n^2\kappa + n\ell)$ communication for ℓ-bit inputs and computational security parameter κ, matching the best known bounds in asynchrony of Lu et al. (PODC 2020) and the best known synchronous bounds of Shrestha et al. (FC 2025). When $n - 2t_s$ is small, we provide an alternative construction with communication $O(n^2\kappa + n\ell)$ in synchrony and $O(\lambda(n^2\kappa + n\ell))$ in asynchrony, where λ is a statistical security parameter. Whenever $f \leq t_r$, both protocols terminate in expected $O(\delta)$ time, where δ is the actual network delay; otherwise, the expected running time is $O(\Delta)$.

1 INTRODUCTION

Large-scale decentralized systems demand consensus protocols that remain robust and efficient under all network conditions. Classic Byzantine consensus (also known as Byzantine agreement) designs force a difficult trade-off: purely synchronous protocols, where messages are delivered within a known upper bound Δ and parties have synchronized clocks, can tolerate almost half of the parties being faulty but rely on conservative timeouts. On the other hand, asynchronous protocols, where messages can be delayed arbitrarily but eventually delivered, guarantee safety without timing assumptions but only handle up to one-third faults. Modern blockchain and state-machine replication (SMR) platforms ideally require the best of all worlds – a single agreement protocol that is safe against unpredictable network delays, live under aggressive fault scenarios, and as fast as the network allows when conditions are favorable.

One line of work attempts to achieve this goal through *optimistic responsiveness* (OR). OR protocols—most notably Pass and Shi's Thunderella [PS18]—assume synchrony and tolerate up to $t < n/2$ faults. Moreover, when an optimistic condition holds (e.g., the designated leader is honest and the number of corruptions is sufficiently small), their running time depends only on the actual network delay rather than the worst-case bound Δ. The drawback, however, is that OR protocols are synchronous – if the network does not remain synchronous, they provide no guarantees. An

Authors' addresses: Fatima Elsheimy, Yale University, New Haven, USA; Simon Holmgaard Kamp, Ruhr University Bochum, Bochum, Germany.

orthogonal approach is taken by network-agnostic (NA) protocols, introduced by Blum, Katz, and Loss [BKL19]. These protocols guarantee correctness under all network conditions: they remain safe and live under asynchronous schedules while achieving higher resilience during synchrony. Concretely, an NA BA protocol tolerates up to t_s faults in synchrony and up to a smaller threshold t_a faults in asynchrony. Such protocols exist if and only if $t_a \leq t_s$ and $2t_s + t_a < n$, this trade-off between the two thresholds is provably optimal [BKL19]. However, almost all existing NA protocols suffer from inefficiencies: they typically run a conservative synchronous phase with a large timeout T (to account for the worst-case Δ) before switching to an asynchronous fallback. This means they forfeit responsiveness – even under good network conditions, an NA protocol might waste time waiting out the worst-case time.

Recently, Elsheimy et al. [EKLN26] achieved a breakthrough by unifying these approaches: they devised the first binary validated BA protocol that is simultaneously network-agnostic and optimistically responsive. Their protocols achieve the best possible fault thresholds (satisfying $n > 2t_s + t_a$ and $n > t_s + 2t_r$ as per their impossibility bounds) and decide values as fast as the network allows under optimistic conditions, while still never violating safety under asynchrony. However, these constructions were limited to binary agreement (where parties agree on a single bit). In practice, real-world systems often require multi-valued validated Byzantine agreement (MVBA) as the consensus layer. In an MVBA protocol, parties propose arbitrary values (e.g., blocks of transactions or state updates) and agree on a single value that satisfies a predefined validity predicate. This guarantees that the decided value is not only consistent across honest parties but also well-formed according to the application's rules—for example, a syntactically valid block or a correctly authorized state transition. Yet, no existing MVBA protocol achieves the same combination of responsiveness, asynchronous safety, and synchronous safety at the optimal fault thresholds and communication efficiency; in particular, the standard reduction from VBA to MVBA is costly, incurring an $O(\ell)$ communication blow-up, where ℓ denotes the input size in bits.

1.1 Our Contributions

We construct the first multi-valued validated Byzantine agreement protocol that simultaneously achieves asynchronous security, synchronous security, and optimistic responsiveness at the optimal resilience thresholds $n > 2t_s + t_a$ and $n > t_s + 2t_r$, while preserving asymptotically optimal efficiency. Our first construction achieves optimal communication whenever $n - 2t_s = \Theta(n)$.

Theorem 1.1. *[BumbleBee Protocol] Let $n > 2t_s + t_a$ and $n > t_s + 2t_r$. Let f denote the actual number of corruptions. Then, BumbleBee protocol (Fig. 6) is secure against strongly adaptive Byzantine adversaries. In particular, it is synchronously secure whenever $f \leq t_s$, asynchronously secure whenever $f \leq t_a$, and responsive whenever $f \leq t_r$. The expected communication complexity is $O(n^2\kappa + \frac{n^2}{n-2t_s}\ell)$, where ℓ is the input size and κ is the computational security parameter. In particular, when $n - 2t_s = \Theta(n)$, this becomes $O(n^2\kappa + n\ell)$. The expected running time is $O(\delta)$ whenever $f \leq t_r$, and $O(\Delta)$ otherwise.*

In our second construction, we remove the dependence on $n - 2t_s$ and obtain the following.

Theorem 1.2. *[BumbleBee+ Protocol] Let $n > 2t_s + t_a$ and $n > t_s + 2t_r$. Let f denote the actual number of corruptions. Then, BumbleBee+ protocol (Fig. 8) is secure against strongly adaptive Byzantine adversaries. In particular, it is synchronously secure whenever $f \leq t_s$, asynchronously secure whenever $f \leq t_a$, and responsive whenever $f \leq t_r$. The expected communication complexity is $O(n^2\kappa + n\ell)$ in synchrony and $O(\lambda(n^2\kappa + n\ell))$ in asynchrony, where λ is the statistical security parameter. The expected running time is $O(\delta)$ whenever $f \leq t_r$, and $O(\Delta)$ otherwise.*

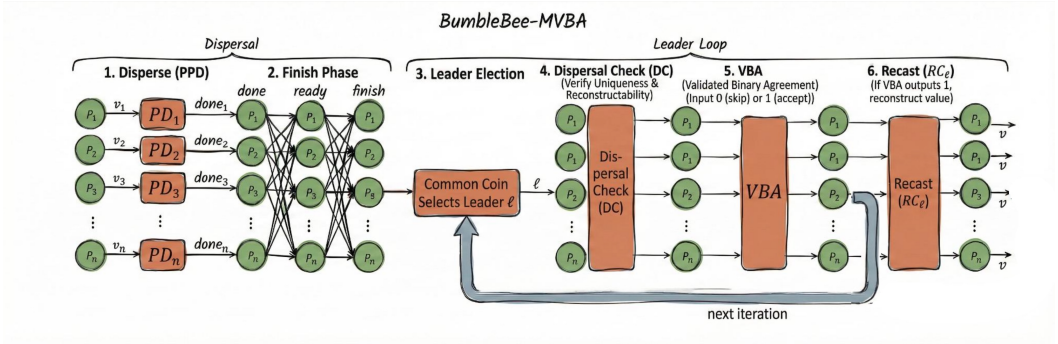


Fig. 1. BumbleBee-MVBA protocol overview.

1.2 Challenges and Techniques

Designing a network-agnostic and responsive MVBA protocol that simultaneously achieves optimal resilience, linear communication in the message size ℓ, and responsiveness requires resolving several tensions. We explain the core obstacles and how our construction addresses them.

From Network-Agnostic BA to Network-Agnostic MVBA. The network-agnostic responsive BA framework of Elsheimy *et al.* achieves optimal resilience under the thresholds $n > 2t_s + t_a$ and $n > t_s + 2t_r$. However, that framework operates in the binary setting. Messages are κ-sized and agreement requires only $O(n^2\kappa)$ communication. In MVBA, each party proposes an ℓ-bit value. Replacing bits by ℓ-bit values inside the NA-BA framework causes a multiplicative ℓ blowup in communication, yielding $O(n^2\kappa\ell)$ bits.

Overview of the Dumbo Paradigm. To restore linear communication, we adopt the Dumbo paradigm, which decouples the dissemination of large values from the core agreement logic. Each value v is first erasure-encoded into n fragments of size $O(\ell/n)$. Using the Provable Dispersal (Π_{PD}) algorithm, the leader sends exactly one unique fragment to each party along with a verifiable commitment, rather than broadcasting the full value, reducing the leader's communication to $n \cdot O(\ell/n + \kappa) = O(\ell + n\kappa)$ bits. Upon receiving and successfully verifying its fragment and commitment, each party P_i responds with a partial signature attesting to verification success. Aggregating a quorum of $n - t$ such signatures produces a constant-sized *lock-certificate* of size $O(\kappa)$, which acts as a transferable proof of data availability, ensuring enough honest parties hold valid fragments and that v is reconstructable. Subsequently, a *done-certificate* is formed to attest that enough honest parties hold valid lock-certificates. Agreement then proceeds over a binary decision—whether to accept or skip the elected leader—based on the availability of done-certificates, making the decision independent of ℓ.

Dual Fast and Slow Execution Paths. Following the network-agnostic framework of Elsheimy *et al.*, our protocol concurrently runs a fast and a slow path in which certificates in the form of threshold signatures with different thresholds are established. The fast path uses certificates of size $n - t_r$ and enables responsiveness; when $f \le t_r$, such certificates can be formed without waiting for synchronous timeouts, yielding expected $O(\delta)$ termination, where δ is the actual message delay. The slow path uses certificates of size $n - t_s$, which allows the protocol to remain live in synchronous executions where up to t_s parties are corrupted. These two certificate thresholds are essential for simultaneously obtaining responsiveness under low corruption and progress under higher corruption. However, this departs from the setting of Dumbo, whose correctness relies on

an honest supermajority ($n > 3t$). In particular, Dumbo uses certificates of size $n - t$ to guarantee uniqueness and reconstructability via intersection, and to ensure that a party cannot be skipped once a done-certificate exists. Once we allow certificates of size $n - t_s < 2n/3$, these guarantees no longer hold, and the safety arguments underlying the Dumbo break down.

The Role of Honest Supermajority. Dumbo relies on the classical asynchronous resilience of $n > 3t$ and collects certificates from $n-t$ partial signatures. First, any two such certificates share signatures from $n - 2t > t$ common parties, of which at least one is honest. This guarantees uniqueness for lock-certificates, which are formed by honest parties signing (at most once) a vector-commitment to the message when they receive this and a code fragment with a proof that it belongs to their designated position in the commitment. Second, if $n - t$ parties have signed that they received a fragment, then $n - 2t > n/3$ honest parties hold a fragment. Generally, if one can guarantee that ρ honest parties hold such verifiable coded fragments of an ℓ-bit value, the cost of reconstructing the value goes from the naive $O(n^2(\ell + \kappa))$ to $O(n^2\ell/\rho + n^2\kappa)$. For $\rho = n - 2t > n/3$ (and generally for $\rho = \Omega(n)$) this results in $O(n\ell + n^2\kappa)$ bits of communication to reconstruct a message towards all parties. Third, if $n - t$ parties sign that they have a lock (thereby forming a done-certificate), then it is impossible to form a skip-certificate for the same leader, because any collection of $n - t$ skipping votes must include at least one party that contributed to the done-certificate. Thus, any honest leader that produced a done-certificate cannot be skipped.

Dual Thresholds and Loss of Honest Supermajority. In the network-agnostic setting, synchronous safety must hold for $f \leq t_s$ under $n > 2t_s + t_a$. Certificates of size $n - t_s$ therefore contain at least $(n - t_s) - t_s = n - 2t_s$ honest contributions. Consequently, two such certificates do not necessarily intersect in an honest party in a synchronous execution. Hence, uniqueness can no longer be derived purely from quorum intersection at threshold $n-t_s$. A naive remedy would be to explicitly exchange all signed store and lock messages to detect equivocation in synchrony during the dispersal phase. However, this blows up the communication by an $\Omega(n)$ factor.

Our Approach. Our design therefore separates reconstructability from uniqueness. Reconstructability is ensured at the Π_{PD} layer: the formation of a lock-certificate implies that sufficiently many parties hold valid fragments, enabling reconstruction of the value. We discuss below how to choose the erasure rate to guarantee this property. This is straightforward in BumbleBee, but requires additional care in BumbleBee+, where the erasure rate is reduced to achieve optimal communication complexity. For uniqueness, we introduce a new protocol, Dispersal Check (Π_{DC}), which allows parties to determine whether the elected leader's value is uniquely reconstructable. After the leader is elected, each party forwards the lock-certificate it received from the leader during Π_{PD} to everyone. In synchrony, bounded-delay communication ensures that conflicting locks are detected within a known time bound. Thus, any party that detects a leader's equivocation sends a skip vote to all parties along with an equivocation proof. This ensures that enough honest parties send skip votes to prevent the leader from being elected. In asynchrony, uniqueness follows from the stronger intersection guarantees implied by $n > 2t_s + t_a$, ensuring that lock and done-certificates can only be constructed for a unique value. Finally, we ensure that an honest party that produced a done-certificate cannot be skipped. In asynchrony, this follows from the bound of t_a corruptions, guaranteeing an honest majority in any collection of $n - t_s$ or $n - t_a$ done-certificates. In synchrony, we ensure that all honest parties hold locks for honest leaders within a bounded time before proceeding to the leader election phase. Consequently, no honest party helps in constructing a conflicting lock-certificate, and no honest leader is skipped.

Running fast and slow paths simultaneously also raises the question: how many honest fragment holders are guaranteed by each threshold, and how should the reconstruction parameter ρ be

chosen accordingly? A fast lock-certificate of size $n - t_r$ guarantees at least $(n - t_r) - t_s = n - t_s - t_r$ honest fragment holders. A slow lock-certificate of size $n - t_s$ guarantees only $(n - t_s) - t_s = n - 2t_s$ honest fragment holders. To gain asymptotically optimal communication, we want to reconstruct from at least a constant fraction of n fragments as explained above. In our first MVBA construction (BumbleBee), we set the reconstruction threshold to $\rho = n - 2t_s$. Under this choice, any slow $(n - t_s)$ certificate automatically guarantees reconstructability, since it certifies the existence of at least ρ honest fragment holders. In this sense, reconstruction comes "for free" once a slow certificate exists. This results in $O(n^2/(n - 2t_s)\ell + n^2\kappa)$ communication, which matches the optimal $O(n\ell + n^2\kappa)$ whenever $n - 2t_s = \Theta(n)$ corruptions are tolerated in asynchrony.

Optimal Communication beyond Linear Asynchronous Resilience. To enable optimal communication beyond this regime, BumbleBee+ reduces the reconstruction threshold to $\rho = n - t_s - t_r$, which is always at least $n/4$ under $t_s + 2t_r < n$ and $t_s < n/2$, ensuring constant-rate encoding independently of $n - 2t_s$. Under this smaller threshold, slow certificates no longer automatically guarantee that ρ honest fragment holders exist. Consequently, BumbleBee+ improves the Dispersal Check protocol, Π_{DC} to $\Pi_{\text{DC+}}$, to additionally verify reconstruction feasibility before accepting a leader. If reconstruction succeeds, a re-dispersal certificate certifies feasibility; otherwise, the leader is skipped.

Detecting Asynchrony via Repeated Failure. The modified Dispersal Check protocol in BumbleBee+ verifies whether an $(n - t_s)$-certificate indeed implies reconstructability. Concretely, upon observing a slow certificate, parties wait for fragments and, if they do not receive enough within $O(\Delta)$ time, they vote to skip the leader. This mechanism is sound in synchrony: if the leader dispersed honestly, all honest parties receive their fragments within bounded time. If reconstruction fails within the timeout window, then either the leader equivocated or did not disperse correctly. However, if the network is asynchronous, the adversary can delay fragment delivery and force parties to time out and skip even an honestly dispersed leader. If this behavior persists indefinitely, the protocol may never terminate.

To prevent this, we bound the number of iterations using $\Pi_{\text{DC+}}$. Parties execute the modified Dispersal Check for at most λ iterations. If all λ iterations fail, then either (i) the network is asynchronous, or (ii) the network is synchronous but every elected leader among these λ rounds was corrupted. In a synchronous execution, the probability that λ consecutive leaders are corrupted is at most $(c)^\lambda$ for constant $1/4 < c < 1$, which is negligible in the security parameter λ. Therefore, except with negligible probability, repeated failure implies that the network is asynchronous. After λ unsuccessful iterations, parties switch to the original BumbleBee protocol, where reconstructability follows from the guarantee that at least $n - t_s - t_a \geq \rho$ honest fragment holders exist.

Maintaining Constant Time in Asynchrony. Although the modified Dispersal Check may allow up to λ iterations before switching logic, this does not increase the asynchronous time complexity. The key observation is that in a fully asynchronous execution with $f \leq t_a$, the high-threshold dispersal phase produces $(n - t_r)$-certificates, and always completes in constant time, since $t_a < t_r$. In this regime, fast certificates are guaranteed to form without relying on synchronous timeouts. Consequently, the protocol rarely enters the timeout-based reconstruction logic. More concretely, the modified Dispersal Check protocol is designed to favor the fast path: as soon as a valid $n - t_r$ certificate is observed, parties immediately accept the leader and bypass the timeout mechanism. This ensures that in genuine asynchronous executions, the protocol behaves identically to the original construction and maintains expected $O(\delta)$ time complexity.

1.3 Additional Related Work

Optimistic Responsiveness. Pass and Shi [PS17] introduced *responsiveness*, requiring running time to depend on the actual network delay δ rather than an upper bound Δ. They showed that deterministic responsiveness is impossible when $n \leq 3t$, but later [PS18] achieved *optimistic responsiveness* via Thunderella, which commits in time $O(\delta)$ under favorable conditions while preserving safety and liveness in synchrony. Their protocol tolerates up to $t_s < n/2$ corruptions for synchronous security and is responsive when fewer than roughly $(n - t_s)/2$ additional faults occur, and this tradeoff is tight. However, optimistic responsive protocols assume synchrony for liveness and do not provide asynchronous safety guarantees. Thunderella achieves this behavior through a two-path design consisting of a responsive path and a fallback path. Under unfavorable conditions, the protocol falls back to a slower consensus mechanism—Nakamoto-style consensus in the permissionless setting or Dolev–Strong or another permissioned SMR protocol in the permissioned setting. Consequently, the worst-case latency and communication complexity inherit those of the fallback protocol.

Network-Agnostic Consensus. Blum, Katz, and Loss [BKL19] initiated the study of *network-agnostic* (NA) consensus, formalizing protocols that are safe under full asynchrony up to $t_a < n/3$ corruptions and live under synchrony up to $t_s < n/2$, provided $n > 2t_s + t_a$. A series of follow-up results extended this paradigm, including general MPC [BZL20], distributed key generation [BCLZL23, ACC22], state-machine replication [BKL21, ABKL22] (See more below), and other consensus variants [GLZW22]. However, all existing NA protocols essentially follow the same design blueprint as BKL19: they run a standard synchronous protocol Π_s for some predetermined running time (the worst-case time needed to run the protocol in case the network is synchronous), and if it has not terminated by then, parties assume the network is asynchronous and switch over to an asynchronous protocol Π_a (secure for t_a faults). Thus, they never achieve expected constant running time.

State Machine Replication (SMR) vs MVBA. State machine replication (SMR) asks a set of replicas to agree on a common ordered log of client commands; equivalently, it can be viewed as Byzantine atomic broadcast, where all honest parties deliver the same commands in the same order. In this sense, SMR is a repeated agreement task: one can construct SMR from Byzantine agreement or MVBA by running an agreement instance for each log position (or batch), while conversely an SMR protocol immediately yields MVBA by having parties atomically broadcast (run SMR on) their proposals and deciding, for example, the first delivered valid value according to a deterministic rule. Thus, MVBA is a natural building block for SMR, especially in asynchronous protocols based on reliable broadcast and common-subset style techniques. Classical Byzantine SMR protocols such as PBFT [CL99], Tendermint [Buc16], HotStuff [YMR+18] are designed for the partially synchronous model and tolerate up to $t < n/3$ Byzantine corruptions. They guarantee safety under arbitrary message delays, but liveness requires eventual synchrony together with sufficiently many successful leader views. However, this entire partially synchronous line fundamentally remains in the $t < n/3$ regime and does not by itself exploit full synchrony to tolerate up to $t < n/2$ faults.

A separate line studies synchronous SMR, where stronger timing assumptions permit resilience up to a Byzantine minority. For example, Sync HotStuff [AMN+19] gives a simple leader-based synchronous SMR protocol with expected constant running time, tolerance up to $t < n/2$, and an optimistic responsiveness when the leader is honest. The tradeoff is that such protocols rely essentially on synchrony and do not provide the asynchronous safety/liveness guarantees sought in network-agnostic or fully asynchronous settings. More recent work [BKL21, ABKL22] on network-agnostic SMR explicitly targets this gap by asking for one protocol that is secure both in synchronous and asynchronous executions, achieving optimal or near-optimal tradeoffs between the synchronous and asynchronous thresholds (See previous paragraph). Overall, the current SMR landscape is split

between partially synchronous responsive protocols with threshold $n/3$, synchronous protocols with threshold $n/2$ and responsiveness, and network-agnostic protocols without responsiveness; bridging all of these features at once *efficiently* remained a central challenge, which we resolved in this paper.

Efficient Multi-Valued Validated BA. In the asynchronous setting, Cachin et al. [CKPS01] was the long-standing state of the art until Abraham et al. [AMS19] constructed validated BA using threshold cryptography with expected $O(n^2(\ell + \kappa))$ communication for ℓ-bit inputs, which is asymptotically optimal when $\ell = O(\kappa)$. Lu et al. [LLTW20] introduced Dumbo, which separates data dissemination from agreement using erasure coding and vector commitments, achieving optimal expected communication $O(n\ell + n^2\kappa)$ for ℓ-bit inputs. Dumbo and its successors, including Dumbo-NG [GLL+22], represent the state of the art in asynchronous MVBA. In the synchronous setting, Shrestha, Abraham, and Nayak [SAN25] designed communication- and round-efficient parallel broadcast and MVBA protocols. Their MVBA achieves optimal expected communication $O(n\ell + n^2\kappa)$ and expected constant-round termination in synchrony. This establishes optimal communication and expected running time for MVBA in each model individually.

Quality and Censorship Resistance. In the context of SMR, and in particular blockchain protocols, quality is a central concern: beyond agreeing on a valid ledger, the protocol should ensure that the ledger is not controlled or censored by the adversary, and that honest participants obtain rewards roughly in proportion to their participation. In Nakamoto-style consensus, this is traditionally captured by *chain quality*, which guarantees that a certain fraction of the blocks on the main chain were contributed by honest parties. For SMR implemented through MVBA, the analogous notion is defined by Abraham et al. [AMS19] as *quality*: the probability that the agreed-upon value was proposed by an honest party. In the case of both Cachin et al. [CKPS01], Abraham et al. [AMS19] and Shrestha et al. [SAN25], the protocols achieve a probability of at least 1/2 that the output was proposed by an honest party. The first construction we present, BumbleBee, picks an honest proposal with probability at least $\frac{n-t_r-t_s}{n-t_r}$. This matches the probability of 1/2 of the works above when $t_s \sim t_r$, but approaches 1/3 when t_s approaches $n/2$. This limitation is inherent, as the protocol can be made to terminate faster than a single network delay in the face of t_r corruptions. Thus, it can also be made to terminate in the view of $n - t_r$ parties who do not observe any messages from t_r honest parties. Up to t_s of these $n - t_r$ parties could be corrupted (but follow the protocol code), and thus the protocol cannot pick an honest proposal with probability better than $\frac{n-t_r-t_s}{n-t_r}$ without distinguishing the corrupted parties from the honest, even when everyone follows the protocol. The second construction, BumbleBee+, which is optimized for the case where $n - 2t_s = o(n)$ achieves the same quality in synchrony, but no provable quality in asynchrony.

Merging Network-Agnostic and Optimistic Responsiveness. Elsheimy, Kamp, Loss, and Nielsen [EKLN26] were the first to merge network-agnostic security with optimistic responsiveness. They constructed a binary Byzantine agreement protocol that simultaneously achieves asynchronous safety up to t_a, synchronous security up to t_s, and responsiveness up to t_r, under the tight conditions $n > 2t_s + t_a$ and $n > t_s + 2t_r$. These bounds are known to be necessary, and their protocol is the first to achieve them simultaneously. However, their construction is restricted to binary agreement and does not address multi-valued validated Byzantine agreement with optimal communication. Our work closes this gap by extending the network-agnostic and optimistic responsiveness framework to multi-valued validated Byzantine agreement while preserving optimal resilience and matching the optimal $O(n\ell + n^2\kappa)$ communication achieved in the best synchronous and asynchronous constructions.

Line of Work	**Sync. Resilience t_s**	**Async. Resilience t_a**	**Responsiveness Cond.**
Partially Sync. SMR ([YMR+18])	$t_s < n/3$	$t_a < n/3$	Honest leader
Optimistically Responsive SMR ([PS18])	$t_s < n/2$	✗	$t_r < (n - t_s)/2$ and honest leader
Network-Agnostic SMR ([BKL21])	$t_s < n/2$	$t_a < n - 2t_s - 1$	✗
Synchronous MVBA ([SAN25])	$t_s < n/2$	✗	✗
Asynchronous MVBA ([LLTW20])	✗	$t_a < n/3$	✓
This Work	$t_s < n/2$	$t_a < n - 2t_s - 1$	$t_r < (n - t_s)/2$

Table 1. Comparison of resilience and responsiveness guarantees. Here t_s and t_a denote the maximum number of corruptions tolerated in the synchronous and asynchronous settings, respectively. The responsiveness column states conditions under which the protocol can terminate in time depending only on the actual network delay δ. In our protocol, responsiveness holds when t_r is below the stated threshold together with additional conditions.

Line of Work	**Sync. Time**	**Resp. Time**	**Async. Time**	**Sync. Communication**	**Async. Communication**
Partially Sync SMR ([YMR+18])	$O(n\Delta)^*$	$O(\delta)$	✗	$O(n\ell + n^2\kappa)^{**}$	✗
Optimistically Responsive SMR*** ([PS18])	$O(n\Delta)$	$O(\delta)$	✗	$O(n\ell + n^2\kappa)$	✗
Network-Agnostic SMR ([BKL21])	$O(n\Delta)$	✗	$O(n\Delta + \delta)$	$O(n^4\kappa + n^3\kappa^2 + n^3\ell + n^2\kappa)$	$O(n^4\kappa + n^3\kappa^2 + n^3\ell + n^2\kappa)$
Synchronous MVBA ([SAN25])	$O(\Delta)$	✗	✗	$O(n\ell + n^2\kappa)$	✗
Asynchronous MVBA ([LLTW20])	$O(\delta)$	$O(\delta)$	$O(\delta)$	✗	$O(n\ell + n^2\kappa)$
BumbleBee	$O(\Delta)$	$O(\delta)$	$O(\delta)$	$O(\frac{n^2\ell}{n-2t_s} + n^2\kappa)$	$O(\frac{n^2\ell}{n-2t_s} + n^2\kappa)$
BumbleBee+	$O(\Delta)$	$O(\delta)$	$O(\delta)$	$O(n\ell + n^2\kappa)$	$O(\lambda(n\ell + n^2\kappa))$

Table 2. Comparison of time and communication complexity.

* For a fair comparison, we consider the variant secure against an adaptive adversary.

** The term $n\ell$ accounts for disseminating the block of size ℓ using erasure coding, while $n^2\kappa$ captures worst-case voting and view-change communication.

*** We instantiate the fallback path with a synchronous permissioned SMR protocol [AMN+19]

2 PRELIMINARIES

System Model. We consider a set of n parties $\mathcal{P} = \{\mathsf{P}_1, \ldots, \mathsf{P}_n\}$, where each pair of parties shares an authenticated communication channel and message delivery is controlled by the adversary $\mathcal{A}$. We assume that $\mathcal{A}$ is computationally-bounded and adaptive, meaning that it may corrupt parties at any point during a protocol execution. Upon corrupting a party P, the adversary learns the state of P and may have it deviate arbitrarily from the protocol. This includes the possibility of not delivering messages that were sent by the party while it was honest. We denote by f the actual number of corrupted parties in an execution. We use three threshold parameters: t_s for the maximum tolerated

corruptions in synchrony, t_a for the maximum tolerated corruptions in asynchrony, and t_r for the maximum tolerated corruptions under which responsiveness is guaranteed.

Clocks and Time Complexity. We assume the existence of a global clock to model time. An execution is said to be *synchronous* if every message exchanged between honest parties is delivered within a known upper bound Δ. In contrast, in an *asynchronous* execution, no a priori bound is assumed on message delays, though messages exchanged between honest parties are guaranteed to be eventually delivered. Following the approach of Canetti and Rabin [CR93], we denote the maximal message delay in a finite execution by δ and use it to characterize asynchronous time complexity.

Vector Commitment *[CF13, KZG10].* A position-binding vector commitment scheme allows committing to an ordered vector of messages while later proving the value stored at any specific position; it guarantees per-coordinate binding but not necessarily hiding. Formally, it consists of polynomial-time algorithms: $\mathsf{VC.Setup}(\lambda, n, \mathcal{M}) \to \mathsf{pp}$, which on input security parameter λ, vector length n, and message domain $\mathcal{M}$ (we assume $\mathcal{M} = \{0,1\}^*$) outputs public parameters pp; $\mathsf{VC.VCom}(\vec{m}) \to (\mathsf{vc}, \mathsf{aux})$, which deterministically commits to a vector $\vec{m} = (m_1, \ldots, m_n)$ producing a commitment vc and auxiliary information aux for openings; $\mathsf{VC.Open}(\mathsf{vc}, i, m_i; \mathsf{aux}) \to \pi_i$, which outputs a proof π_i that m_i is the value committed at position i; and $\mathsf{VC.Verify}(\mathsf{vc}, i, m_i, \pi_i) \to \{0,1\}$, which checks the validity of such an opening. Correctness requires that honestly generated openings always verify, and position binding requires that, except with negligible probability, no efficient adversary can produce two distinct valid openings for the same index under the same commitment.

(t,n)-Threshold Signature Scheme*[LJY14, Sho00].* A (t,n)-threshold signature scheme over message space $\mathcal{M}$ consists of probabilistic polynomial-time algorithms as follows: $\mathsf{TS.Setup}(1^\lambda) \to \mathsf{pp}$, which on input security parameter λ outputs global public parameters pp; $\mathsf{TS.KeyGen}(\mathsf{pp}, t, n) \to (\mathsf{PK}, \mathsf{pk}_1, \ldots, \mathsf{pk}_n, \mathsf{sk}_1, \ldots, \mathsf{sk}_n)$, a distributed key-generation protocol producing a global verification key PK, individual public verification keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_n$, and corresponding secret key shares $\mathsf{sk}_1, \ldots, \mathsf{sk}_n$ for threshold t among n parties; $\mathsf{TS.SignShare}_t(\mathsf{sk}_i, m) \to \sigma_i$, which allows party i to compute a partial signature share σ_i on message $m \in \mathcal{M}$ using its secret share; $\mathsf{TS.VerifyShare}_t(\mathsf{pk}_i, m, \sigma_i) \to \{\mathsf{accept}, \mathsf{reject}\}$, which checks the validity of a signature share; $\mathsf{TS.Combine}_t(\{\sigma_i\}_{i \in S}) \to \sigma$, which combines any set $S \subseteq \{1, \ldots, n\}$ of at least t valid shares on the same message into a full signature σ; and $\mathsf{TS.Verify}_t(\mathsf{PK}, m, \sigma) \to \{\mathsf{accept}, \mathsf{reject}\}$, which verifies the resulting threshold signature under PK. Correctness requires that honestly generated signature shares always verify and that any collection of at least t valid shares on the same message combines into a full signature that is accepted by the verification algorithm.

Common Coin. We assume an ideal protocol for common coin Π_{CC}. If all honest parties invoke Π_{CC} with an identifier id, the functionality gives a unique coin c_{id} to all honest parties. The value of this coin is unpredictable until the first honest party invokes Π_{CC} with id. This can be implemented asynchronously using $(n - t_s)$-threshold signatures [CKS00], assuming at least a single honest contribution (implied by $2t_s + t_a < n$ and $t_s + 2t_r < n$). Threshold-signature-based constructions typically incur $O(n^2\kappa)$ communication and have a single round of interaction.

Erasure Coding Scheme. An (k,n)-erasure code consists of two deterministic polynomial-time algorithms Enc and Dec. The encoding algorithm $\mathsf{Enc}(m, n) \to (s_1, \ldots, s_n)$ takes as input a message $m \in \{0,1\}^*$ and an integer n, and deterministically produces n coded shares $s_1, \ldots, s_n$; the decoding algorithm $\mathsf{Dec}(\{s_i\}_{i \in S}) \to m'$ takes any subset of shares indexed by $S \subseteq [n]$ and outputs a reconstructed message m'. Reconstruction correctness requires that for every message m, any subset of at least k shares produced by Enc uniquely reconstructs m.

Instantiation. We instantiate the erasure code using a Reed–Solomon construction [RS60], which provides optimal distance and deterministic reconstruction from any sufficiently large subset of shares. For the first protocol, BumbleBee, we set the reconstruction threshold so that the message can be recovered from any subset $S \subseteq [n]$ with $|S| \geq n - 2t_s$, where t_s denote the maximum numbers of missing or corrupted shares. For BumbleBee+, we set the reconstruction threshold such that any subset $S \subseteq [n]$ satisfying $|S| \geq n - t_s - t_r$ can recover the message.

Honest Parties and View-Based Certificate Validity. Some of our security proofs will be based on properties that hold for all certificates of a certain type, or the existence of a certificate at a certain time. We say that a valid certificate "exist" if it appears in the view of an honest party and satisfies a corresponding predicate. We stress that here, and whenever we say "honest party" without further qualifiers, we mean a "forever honest" party who does not become corrupted during the execution of the protocol. This is an important distinction as we want the existence of a certificate to be monotone, and because we will later define some of our security properties w.r.t. parties who are honest up to some point in the execution. For instance, while they dispersed a message. In those cases we always specify at which point the execution the party was still following the protocol.

2.1 Problem Definitions

We now formalize the problems solved by our construction. We begin with the standard definition of multi-valued validated Byzantine agreement. We then define an intermediate abstraction, Parallel Provable Dispersal Broadcast (PPDB), which encapsulates the dispersal and reconstruction mechanisms underlying BumbleBee and BumbleBee+. The MVBA protocol is obtained by combining PPDB with a validated binary agreement layer.

Definition 2.1 (MVBA). In an MVBA protocol with an external predicate $\mathsf{Pred} : \{0,1\}^\ell \to \{\top, \bot\}$, the parties take values satisfying Pred as inputs and aim to output a common value satisfying Pred. The MVBA protocol guarantees the following properties:

- **Termination.** If every honest party P_i is activated on identification id, taking as input a value v_i and auxiliary string aux such that $\mathsf{Pred}(v_i, aux) = \top$, then every honest party outputs a value v for id.
- **External-Validity.** If an honest party outputs a value v and auxiliary string aux for id, then $\mathsf{Pred}(v, aux) = \top$.
- **Agreement.** If any two honest parties output v and v' for id respectively, then $v = v'$.
- (q)-**Quality.** If an honest party outputs v for id, the probability that v was proposed by an honest party is at least q.

We use the binary VBA protocol by Elsheimy et al. [EKLN26] as a building block in our MVBA. That is, both the input and the output domains of the VBA are binary; such that, each party provides an input in $\{0,1\}$ and the protocol outputs a value in $\{0,1\}$.

Parallel Provable Dispersal Broadcast. The following definition extends the core conceptual contribution of Dumbo - its Asynchronous Provable Dispersal Broadcast - to the best-of-both-worlds setting.

Definition 2.2 (PPDB Protocol). A PPDB protocol among n parties $\mathsf{P}_1, \ldots, \mathsf{P}_n$ consists of three subprotocols: a parallel provable dispersal subprotocol (PPD), a dispersal check subprotocol (DC), and a recast subprotocol (RC). The protocol is parameterized by corruption thresholds t_s and t_r, where t_s denotes the synchronous corruption threshold and t_r denotes the responsive corruption threshold. For $t \in \{t_s, t_r\}$, we define a quorum size $T := n - t$. The protocol further defines the

predicates (ValidateStore, ValidateLock, ValidateReconstruct, ValidateFinish). Its syntax is defined as follows.

- PPD **subprotocol.** In the PPD subprotocol, each party P_i inputs a value $v_i \in \{0,1\}^\ell$ and aims to split v_i into n encoded fragments, dispersing one fragment to each party. For each threshold $T \in \{n - t_s, n - t_r\}$, a party may output a certificate $\texttt{Finish}_T$ together with corresponding vectors of store and lock-certificates $\left(\texttt{Store}^j\right)_{j\in[n]}$ and $\left(\texttt{Lock}^j_T\right)_{j\in[n]}$.
- DC$[i,k]$ **subprotocol.** In the DC$[i,k]$ subprotocol identified by party index $i \in [n]$ and phase $k \in \mathbb{N}$, all honest parties take as input the outputs of PPD corresponding to sender P_i; $(\texttt{Store}^i, \texttt{Lock}^j_T, \texttt{Finish}_T)$ for $T \in \{n-t_s, n-t_r\}$ and output a pair (b, cert), where $b \in \{0,1\}$ indicates whether dispersal by P_i succeeded, and cert is a corresponding proof. If $b = 1$, then cert is a proof that is valid according to the reconstruct predicate, guaranteeing reconstructability of the value v_i dispersed by P_i; otherwise, cert is a skip-certificate, certifying that reconstruction cannot be guaranteed, and the elected leader should be discarded.
- RC$[i,k]$ **subprotocol.** In the RC$[i,k]$ subprotocol identified by party index $i \in [n]$ and phase k, all honest parties take as input the outputs of PPD and DC$[i,k]$ corresponding to sender P_i and attempt to reconstruct the value v_i dispersed by P_i in PPD. Upon completion of RC$[i,k]$, all honest parties output a common value in $\{0,1\}^\ell \cup \{\perp\}$.

A secure PPDB protocol (PPD, DC, RC) achieves the following security properties:

- PPD-**Termination.** If all honest parties invoke PPD, then each of them eventually gets an output $\texttt{Finish}_{n-t_s}$. If additionally there are fewer than t_r corruptions, then they also eventually get an output $\texttt{Finish}_{n-t_r}$.
- DC-**Termination.** If all honest parties invoke DC$[s,k]$ with a valid $\texttt{Finish}_T$, then they each output some valid $\texttt{Skip}^{s,k}$ or $\texttt{Reconstruct}^{s,k}$ certificates.
- RC-**Termination.** If all honest parties invoke RC$[s,k]$ with a valid $\texttt{Reconstruct}^{s,k}$ certificate, then they all output some message m.
- RC-**Agreement.** If two honest parties output m and m' from RC$[s,k]$, then $m = m'$.
- (h,c)-**Two-Step-Locking.** For each time τ, let the number of distinct thresholds T for which there is a $\texttt{Finish}_T$ certificate in the view of a forever honest party at time τ be l_τ. If $l_\tau > 0$, then the choices of the adversary are limited in the following way. For any time $\tau' \geq \tau$ where $l_{\tau'} = l_\tau$ (i.e., until a time where another Finish certificate exist in the joint view of the honest parties), there is a set of at least h forever honest parties H_{l_τ} and a set of at most c parties C_{l_τ}, such that for any instance DC$[s,k]$ which is not invoked by an honest party before time τ the following holds:
  - if a valid $\texttt{Skip}^{s,k}$ is in the view of an honest party before time τ', then $s \notin H_{l_\tau}$.
  - if a valid $\texttt{Reconstruct}^{s,k}$ is in the view of an honest party before time τ', and an honest party receives output m from RC$[s,k]$, then: either P_s was honest at the time of receiving input $v_s = m$, or $s \in C_{l_\tau}$.

Intuitively, the (c,h)-Two-Step-Locking property gives us that, even though the adversary can choose to skip a leader after it is elected in the MVBA protocol: in order to do so, it must also move the execution from a state where only one Finish certificate exists (in the joint view of honest parties), to a state where two exists (one for each threshold). Since there are only two distinct thresholds, this happens at most once during the execution.

We additionally define the following relaxed Two-Step-Locking property, which will be needed to prove constant time complexity in asynchrony for BumbleBee+:

- (h, c)-**Weak-Two-Step-Locking.** The weaker version of two step locking allows all leaders to be skipped when only the lower threshold finish $\texttt{Finish}_{n-t_s}$ exists, and otherwise satisfies the same guarantees as (h, c)-Two-Step-Locking. Concretely:
  For each time τ, let the number of distinct thresholds T for which there is a $\texttt{Finish}_T$ certificate in the view of a forever honest party at time τ be l_τ. If $l_\tau > 0$, then the choices of the adversary are limited in the following way. For any time $\tau' \geq \tau$ where $l_{\tau'} = l_\tau$ (i.e., until a time where another Finish certificate exist in the joint view of the honest parties), there is a set of at least h forever honest parties H_{l_τ} and a set of at most c parties C_{l_τ}, such that for any instance $\mathsf{DC}[s, k]$ which is not invoked by an honest party before time τ the following holds:
  - if a valid $\mathsf{Skip}^{s,k}$ is in the view of an honest party before time τ', then $s \notin H_{l_\tau}$ *or no honest party had* $\texttt{Finish}_{n-t_r}$ *certificate in their view at time* l_τ.
  - if a valid $\texttt{Reconstruct}^{s,k}$ is in the view of an honest party before time τ', and an honest party receives output m from $\mathsf{RC}[s, k]$, then: either P_s was honest at the time of receiving input $v_s = m$, or $s \in C_{l_\tau}$.

3 BUMBLEBEE MVBA PROTOCOL

In this section, we present *BumbleBee*, our network-agnostic and responsive multi-valued validated Byzantine agreement protocol. The protocol is parameterized by thresholds t_s, t_a, t_r satisfying the optimal resilience conditions $2t_s + t_a < n$ and $t_s + 2t_r < n$, which are known to be necessary and sufficient for network-agnostic, and responsive MVBA.

In MVBA, each party proposes an ℓ-bit value satisfying an external predicate V, and the goal is for all honest parties to agree on a single value that also satisfies V. Unlike binary agreement, values are large objects (e.g., blocks of transactions), and therefore efficiency requires separating data dissemination from agreement. A naive all-to-all broadcast of full inputs would incur quadratic $\Theta(n^2\ell)$ communication. Recent MVBA protocols [GLT+20, LLTW20, GLL+22] instead follow a three-layer architecture. First, parties disperse their inputs using erasure coding and commitments, ensuring that values can later be reconstructed from sufficiently many fragments. Second, leaders are repeatedly elected using a common coin and parties vote based on whether they have seen a lock-certificate for the leader. They then proceed to run a binary validated Byzantine agreement (VBA) protocol to decide whether to accept or skip the elected leader. Third, once a leader is accepted, parties reconstruct the dispersed value.

Our construction follows this high-level architecture of Dumbo-MVBA, which achieves asymptotically optimal expected communication $O(n\ell + n^2\kappa)$ in the asynchronous setting. BumbleBee preserves Dumbo's three-layer structure but modifies the certificate logic. It consists of the following building blocks:

- *Parallel Provable Dispersal* (Π_{PPD}): Each party disperses its value and obtains transferable lock and done-certificates proving that sufficiently many parties received consistent fragments.
- *Dispersal Check* (Π_{DC}): Given an elected leader, parties determine whether its dispersal yields a unique reconstructable commitment or whether the leader must be skipped.
- *Binary Validated Byzantine Agreement* (Π_{VBA}): Coordinates leader selection and decides whether to reconstruct the elected leader's value or re-elect some other party.
- *Recast* (Π_{RC}): If uniqueness is certified and Π_{VBA} outputs 1, parties multicast the shares dispersed by the leader and reconstruct the value using erasure decoding.

The fundamental structural difference from Dumbo is that we simultaneously maintain two certificate thresholds, $n - t_r$ and $n - t_s$, rather than a single $(n - t)$ threshold. These two thresholds serve distinct purposes.

Fast and Slow Thresholds. The protocol employs two thresholds that capture different network regimes. The fast threshold $n - t_r$ guarantees responsiveness when the actual number of corruptions satisfies $f \leq t_r$. In this case, any certificate of size $n - t_r$ contains at least $n - t_r - f \geq n - 2t_r$ honest contributions. The resilience condition $n > t_s + 2t_r$ implies $n - 2t_r > t_s$, and therefore any two certificates of size $n - t_r$ intersect in more than t_s parties and hence in at least one forever honest party. Since honest parties never sign conflicting values, fast certificates guarantee uniqueness and allow the protocol to terminate without waiting for Δ, yielding responsiveness. In contrast, the slow threshold $n - t_s$ ensures progress in the synchronous regime when up to t_s parties may be corrupted. In a synchronous network, we have $f \leq t_s$, and therefore at least $n - t_s$ honest parties exist, ensuring that certificates of size $n - t_s$ can always be formed and propagated once a leader's dispersal is acknowledged.

However, certificates of size $n - t_s$ do not necessarily enjoy quorum intersection when $t_s \geq n/3$, since two such sets may have no honest party in their intersection. Instead of relying purely on intersection arguments, BumbleBee detects equivocation explicitly in the Dispersal Check phase. After a finish certificate is formed, parties exchange the lock-certificates they have observed; in synchrony, messages are delivered within Δ time. Consequently, within a bounded window, either all honest parties observe a consistent lock and certify uniqueness, or conflicting locks are detected, or sufficiently many parties report the absence of a lock and produce a skip-certificate. If sufficiently many parties obtain such a skip-certificate, they input 0 to the VBA, ensuring that the leader is skipped and reconstruction does not proceed. On the other hand, in asynchrony, safety follows from the condition $n > 2t_s + t_a$: when $f \leq t_a$, any two slow certificates of size $n - t_s$ intersect in at least $n - 2t_s - t_a > 0$ parties, guaranteeing a forever honest intersection and thus preserving safety even without timing assumptions.

We now describe each component in detail.

3.1 Parallel Provable Dispersal Protocol

The goal of the Parallel Provable Dispersal (Π_{PPD}) protocol is to allow each party P_s to disperse its input in a way that later enables verifiable uniqueness and reconstruction.

Each party P_s with input m_s invokes a provable dispersal instance $\Pi_{\mathsf{PD}}(\rho)[s]$ as the designated sender, where ρ denotes the reconstruction threshold. In addition, it participates in the dispersal instances $\Pi_{\mathsf{PD}}(\rho)[j]$ for all other parties $j \in [n]$. To disperse its own value, P_s computes an (ρ, n) Reed–Solomon encoding $(s_1, \ldots, s_n) \leftarrow \mathsf{Enc}(m_i, \rho)$ and a vector commitment $\mathsf{vc} := \mathsf{VC.VCom}(s_1, \ldots, s_n)$. For our first protocol BumbleBee, we set the reconstruction threshold to $\rho = n - 2t_s$ and assume $n - 2t_s = \Theta(n)$, ensuring that the coding rate remains constant and that the total dispersal communication is linear in ℓ. Consequently, for each party P_j, it sends $(\mathsf{store}, s_j, \pi_j, \mathsf{vc})$, where π_j proves that s_j is the j-th coordinate of vc_i.

Upon receiving its first store message from P_s, party P_j verifies if it is *valid* by checking $\mathsf{VC.Verify}(\mathsf{vc}, j, s_j, \pi_j) = 1$. If so, it returns partial signatures for both thresholds $T \in \{n - t_r, n - t_s\}$ on the string $(\mathsf{Stored}_T, \mathsf{vc}, s)$. The sender aggregates T such shares to produce a lock-certificate $\mathsf{Lock}_T^{\mathsf{vc},s}$. After receiving a lock-certificate for threshold T, each party checks if it is a valid lock-certificate. A lock-certificate is considered valid if $\mathsf{ValidateLock}(T, \mathsf{Lock}_T^{\mathsf{vc},s}) = 1$, that is, if $\mathsf{TS.Verify}_T(\mathsf{PK}, (\mathtt{Stored}_T, \mathsf{vc}, s), \mathsf{Lock}_T^{\mathsf{vc},s}) = \mathsf{accept}$. Intuitively, such a certificate proves that at least T parties acknowledge having received consistent fragments corresponding to commitment

vc from sender P_s. If it is valid, parties return partial signatures on (locked_T, s), allowing the sender to combine T shares into a done-certificate $\mathtt{Done}^s_T$. The fast done-certificate $\mathtt{Done}^s_{n-t_r}$ is output immediately once formed. The slow done-certificate $\mathtt{Done}^s_{n-t_s}$ is only released after waiting long enough to guarantee that, in synchrony, all honest parties have completed dispersal. This waiting ensures that a slow certificate cannot be created before honest dispersals are globally visible.

Parties then enter the ready phase. Upon observing $n - t_r$ valid fast done-certificates, a party signs a readiness message for threshold $n - t_r$ by producing a partial signature on $\mathtt{Ready}_{n-t_r}$ with threshold $t_s + 1$. Similarly, upon observing $n - t_s$ valid slow done-certificates and waiting sufficiently long, a party signs $\mathtt{Ready}_{n-t_s}$. Once $t_s + 1$ valid readiness shares are collected, they are combined into a threshold certificate $\mathtt{Ready}_T$ for $T \in \{n - t_r, n - t_s\}$. Upon observing a valid $\mathtt{Ready}_T$ certificate for the first time, a party forwards it, abandons all remaining Π_{PD} instances for threshold T, and signs $\mathtt{Finish}_T$ with threshold T. Finally, once a finish certificate exists in the view of an honest party, it is forwarded to all parties before terminating Π_{PPD}. This ensures that if the network is synchronous, honest parties terminate the protocol Δ apart from each other (one step apart). Upon termination, parties output the set of store messages, lock and finish certificates that they collected during the Π_{PPD} protocol.

3.2 Dispersal Check Protocol

Assume P_s is the elected leader in iteration k, the dispersal check protocol $\Pi_{\mathsf{DC}}[s, k]$ determines whether the leader's value is uniquely reconstructable or must be skipped. To do so, parties invoke $\Pi_{\mathsf{DC}}[s, k]$ on the outputs of Π_{PPD}.

Upon receiving a finish certificate $\mathtt{Finish}_T$, each party checks whether it holds a valid lock-certificate for P_s. If a party observes a valid $\mathtt{Lock}^{vc,s}_{n-t_r}$ and no conflicting slow lock, it multicasts a partial signature on $(\mathsf{fastvote}, s, k, vc)$. Collecting $t_s + 1$ such shares produces a fast uniqueness certificate $\mathtt{Unique}^{vc,s,k}_{n-t_r}$. If instead a party observes a valid $\mathtt{Lock}^{vc,s}_{n-t_s}$ and, after waiting 2Δ, no conflicting lock-certificate or higher threshold lock, $\mathtt{Lock}^{\mathsf{vc},s}_{n-t_r}$, exists, it signs $(\mathsf{Vote}, s, k, vc)$. Collecting $t_s + 1$ such shares produces a slow uniqueness certificate $\mathtt{Unique}^{vc,s,k}_{n-t_s}$.

Equivocation is handled explicitly. If a party observes two conflicting lock-certificates for P_s or receives sufficiently many NoLock messages after a finish certificate, it signs $(\mathsf{Vote}, s, k, \perp)$. Collecting $n - t_s$ such shares produces a skip-certificate $\mathtt{Skip}^{s,k}$.

Thus, the output of $\Pi_{\mathsf{DC}}[s, k]$ is either a uniqueness certificate $\mathtt{Unique}^{\mathsf{vc},s,k}_T$ validated by ValidateUnique, or a no lock-certificate validated by ValidateNoLock. Intuitively, the dispersal check phase verifies the leader's behavior during the dispersal phase—whether its value is uniquely reconstructable or whether it equivocated. These certificates ensure that only well-formed and globally consistent evidence can be supplied as input to the subsequent validated binary agreement phase. In particular, a vote for 1 corresponds to a certified commitment that can safely be reconstructed (unique certificates), while a vote for 0 corresponds to certified evidence that the leader must be skipped (skip-certificate).

3.3 Recast Protocol

If $\Pi_{\mathsf{DC}}[s, k]$ outputs a valid uniqueness certificate $\mathtt{Unique}^{\mathsf{vc},s,k}_T$, parties invoke the binary validated Byzantine agreement (Π_{VBA}) protocol with input 1, justified by this certificate. By the external validity property of VBA, any output 1 must be accompanied by a valid justification π such that $\mathsf{ValidateIn}(1, \pi) = 1$. In particular, if VBA outputs 1, the justification must correspond to a valid uniqueness certificate for the leader P_s.

Upon obtaining a justified output 1 from VBA, parties proceed to reconstruct the value committed in vc. Each party forwards its stored fragment $(s_j, \pi_j, \mathsf{vc})$ corresponding to leader P_s. Once a party

$\Pi_{\mathrm{PD}}(\rho)[s]$

All parties $\mathsf{P}_1, \ldots, \mathsf{P}_n$ hold flags $\mathtt{stop}_{n-t_s} = 0$, $\mathtt{stop}_{n-t_r} = 0$ and a designated sender P_s additionally holds initially empty sets S_{n-t_r}, S_{n-t_s}, L_{n-t_r}, and L_{n-t_s}.
The designated sender, P_s, executes the following logic.

Store: Upon receiving an input value m, P_s records the current time as τ_i^{input}, and computes the RS encoding $s_1, \ldots, s_n := \mathsf{Enc}(m, \rho)$ and the vector commitment $vc := \mathsf{VC.VCom}(s_1, \ldots, s_n)$. Then for each party P_j, P_s computes the proof that s_j is in position j of vc, π_j and sends $(\mathtt{store}, s_j, \pi_j, vc)$ to party P_j.

Lock: Upon receiving a partial signature σ for threshold $T \in \{n - t_r, n - t_s\}$ where $\mathsf{TS.VerifyShare}_T(\mathsf{PK}, s, (\mathtt{Stored}_T, vc, s), \sigma) =$ accept: if $|S_T| < T$, $\sigma \notin S_T$, and $\mathtt{stop}_T = 0$, then P_s updates $S_T := S_T \cup \{\sigma\}$. If after this update $|S_T| = T$, then P_s computes $\mathsf{Lock}_T^{vc,s} = \mathsf{TS.Combine}_T(\mathsf{PK}, S_T)$ and sends $\mathsf{Lock}_T^{vc,s}$ to all parties.

Done: Upon receiving a partial signature σ for threshold $T \in \{n - t_r, n - t_s\}$ where $\mathsf{TS.VerifyShare}_T(\mathsf{PK}, s, (\mathtt{locked}_T, s), \sigma) =$ accept: if $|L_T| < T$, $\sigma \notin L_T$, and $\mathtt{stop}_T = 0$, then P_s updates $L_T := L_T \cup \{\sigma\}$. If after this update $|L_T| = T$, then P_s computes $\mathtt{Done}_T^s = \mathsf{TS.Combine}_T(\mathsf{PK}, L_T)$. Output $\mathtt{Done}_{n-t_r}^s$ once computed and output $\mathtt{Done}_{n-t_s}^s$ after waiting till $\tau_i^{input} + 4\Delta$.

Each party P_i (including P_s) executes the following logic.

Abandon: Upon input $\mathsf{abandon}_T$, set $\mathtt{stop}_T = 1$.

Receive Store: Upon receiving the first message of the form $(\mathtt{store}, s_i, \pi_i, vc)$ from P_s, if $\mathsf{VC.Verify}(vc, i, s_i, \pi_i) = 1$, then for each $T \in \{n - t_s, n - t_r\}$ where $\mathtt{stop}_T = 0$, P_i computes a partial signature for threshold T: $\sigma_T = \mathsf{TS.SignShare}_T(i, \mathsf{SK}_i, (\mathtt{Stored}_T, vc, s))$, and sends it to P_s. Additionally P_i outputs $(\mathtt{store}, s_i, \pi_i, vc)$.

Receive Lock: Upon receiving a threshold signature $\mathsf{Lock}_T^{vc,s}$ for some threshold $T \in \{n - t_r, n - t_s\}$ where $\mathsf{TS.Verify}_T(\mathsf{PK}, (\mathtt{Stored}_T, vc, s), \mathsf{Lock}_T^{vc,s}) =$ accept for the first time, if $\mathtt{stop}_T = 0$: If $T = n - t_s$, P_i computes a partial signature for threshold $n - t_s$: $\sigma_{n-t_s} = \mathsf{TS.SignShare}_{n-t_s}(i, \mathsf{SK}_i, (\mathtt{locked}_{n-t_s}, s))$, and sends σ_{n-t_s} to P_s, outputs $\mathsf{Lock}_{n-t_s}^{vc,s}$. Else, if $T = n - t_r$, P_i waits till it also receives $\mathsf{Lock}_{n-t_s}^{vc,s}$ from P_s before computing a partial signature for threshold $n - t_r$: $\sigma_{n-t_r} = \mathsf{TS.SignShare}_{n-t_r}(i, \mathsf{SK}_i, (\mathtt{locked}_{n-t_r}, s))$, and sends σ_{n-t_s} to P_s, outputs $\mathsf{Lock}_{n-t_s}^{vc,s}$.

Fig. 2. Protocol for Provable Dispersal with a single sender P_s. The parameter ρ determines how many fragments of the erasure code are needed to reconstruct the input m.

collects $\rho = n - 2t_s$ valid fragments for the same commitment vc, it reconstructs $m' := \mathsf{Dec}(\{s_j\}_{j \in S})$ and verifies that $\mathsf{vc} = \mathsf{VC.VCom}(\mathsf{Enc}(m', \rho))$. If this check succeeds, it outputs m'; otherwise it outputs $\perp$.

Reconstruction is guaranteed whenever a valid uniqueness certificate exists. Indeed, any valid lock-certificate underlying $\mathtt{Unique}_T^{\mathsf{vc},s,k}$ contains at least $T - t_s \geq n - 2t_s$ honest contributions, and we set the reconstruction threshold to $\rho = n - 2t_s$. Thus, sufficiently many correct fragments must exist for decoding to succeed. Moreover, the position-binding property of the vector commitment ensures that two honest parties cannot reconstruct different values from the same commitment vc except with negligible probability.

$\Pi_{\mathsf{PPD}}(\rho)$

All parties $\mathsf{P}_1, \ldots, \mathsf{P}_n$ initially hold empty sets S^{Store}, S_T^{Lock}, and hold flags $\mathtt{finishFlg}_T$ for each $T \in \{n - t_s, n - t_r\}$, all initially set to $\perp$.

Disperse: Upon receiving input m, P_i inputs m to $\Pi_{\mathsf{PD}}(\rho)[i]$, and participate in all $\{\Pi_{\mathsf{PD}}(\rho)[j]\}_{j\in[n]}$

Store: Upon $\Pi_{\mathsf{PD}}(\rho)[j]$ delivering output $\mathsf{Store}^{\mathsf{vc},j}$, add $\mathsf{Store}^{\mathsf{vc},j}$ to S^{Store}.

Lock: Upon $\Pi_{\mathsf{PD}}(\rho)[j]$ delivering output $\mathsf{Lock}_T^{\mathsf{vc},j}$, add $\mathsf{Lock}_T^{\mathsf{vc},j}$ to S_T^{Lock}.

Done: Upon $\Pi_{\mathsf{PD}}(\rho)[i]$ delivering an output of the form $\mathtt{Done}_T^i$ for $T \in \{n - t_s, n - t_r\}$, multicast $\mathtt{Done}_T^i$.

Fast Ready: Upon having received valid $\mathtt{Done}_{n-t_r}^j$ certificates for $n - t_r$ distinct parties P_j, P_i multicasts a partial signature with threshold $t_s + 1$ on the string $\mathtt{Ready}_{n-t_r}$: $\mathsf{Sig}_{t_s+1,\mathsf{sk}_i}(\mathtt{Ready}_{n-t_r})$.

Slow Ready: Upon having receiving valid $\mathtt{Done}_{n-t_s}^j$ certificates for $n - t_s$ distinct parties P_j *and* having waited until time $\tau_i^{input} + 5\Delta$, P_i multicasts a partial signature with threshold $t_s + 1$ on the string $\mathtt{Ready}_{n-t_s}$: $\mathsf{Sig}_{t_s+1,\mathsf{sk}_i}(\mathtt{Ready}_{n-t_s})$.

Combine Ready: Upon receiving $t_s + 1$ distinct valid partial signatures with threshold $t_s + 1$ on the string $\mathtt{Ready}_T$ for $T \in \{n - t_s, n - t_r\}$, compute and multicast $\mathtt{Ready}_T = \mathsf{TS.Combine}_{t_s+1}(\mathtt{Ready}_T)$.

Forward Ready: Upon receiving a valid $(t_s + 1)$ threshold signature on the string $\mathtt{Ready}_T$ for $T \in \{n - t_s, n - t_r\}$ for the first time, P_i multicasts it, inputs $\mathsf{Abandon}_T$ to all instances of Π_{PD}, and *only then* multicasts a partial signature with threshold T on the string $\mathtt{Finish}_T$: $\mathsf{Sig}_{T,\mathsf{sk}_i}(\mathtt{Finish}_T)$.

Combine Finish: Upon receiving T distinct valid partial signatures on the string $\mathtt{Finish}_T$ for $T \in \{n - t_s, n - t_r\}$, compute $\mathtt{Finish}_T = \mathsf{TS.Combine}_T(\mathtt{Finish}_T)$, multicast $\mathtt{Finish}_T$ and set $\mathtt{finishFlg}_T = 1$.

Forward Finish: Upon receiving a valid $\mathtt{Finish}_T$ and $\mathtt{finishFlg}_T = 0$, forward it to all parties, and set $\mathtt{finishFlg}_T = 1$.

Output if finish: If $\mathtt{finishFlg}_T = 1$, output $(S^{Store}, S_T^{Lock}, \mathtt{Finish}_T)$.

Fig. 3. Protocol for Parallel Provable Dispersal

3.4 Constructing The BumbleBee Protocol

We now summarize how MVBA is constructed from the previously described building blocks. Each party P_i begins by invoking $\Pi_{\mathsf{PPD}}(\rho)$ with input m_i. After obtaining a finish certificate, parties enter iterations indexed by $k = 1, 2, \ldots$.

In iteration k, parties invoke the common coin $\Pi_{\mathsf{CC}}[k]$ to elect a leader P_s, and then execute $\Pi_{\mathsf{DC}}[s, k]$. The dispersal check outputs either a uniqueness certificate $\mathtt{Unique}_T^{\mathsf{vc},s,k}$ or a skip-certificate $\mathsf{Skip}^{s,k}$. Crucially, this certificate determines the binary input to the validated Byzantine agreement Π_{VBA} layer. If $\Pi_{\mathsf{DC}}[s, k]$ outputs a uniqueness certificate, parties input 1 to Π_{VBA} justified by $\mathtt{Unique}_T^{\mathsf{vc},s,k}$; if it outputs a skip-certificate, parties input 0 justified by $\mathsf{Skip}^{s,k}$.

Because our VBA satisfies *provable agreement*, any output bit must be accompanied by a valid justification. In particular, no party can input or cause the agreement of an arbitrary value: a vote for 1 must be justified by a valid uniqueness certificate, and a vote for 0 must be justified by a valid skip-certificate. If Π_{VBA} outputs 0, parties proceed to the next iterations and elect a new leader. If Π_{VBA} outputs 1, the output is accompanied by a valid uniqueness certificate, and parties invoke the recast phase to reconstruct the value committed in vc. As argued above, reconstruction succeeds whenever a valid uniqueness certificate exists, and all honest parties reconstruct the same value

$\Pi_{\text{DC}}[s,k]$

The following logic is run by all parties $\mathsf{P}_i \in \mathcal{P}$ to certify that a unique message was dispersed by P_s. We use $\texttt{Finish}_T = \top$ to indicate that a valid $\texttt{Finish}_T$ certificate has been received as input or over the network.

Forward Finish: Upon receiving $\texttt{Finish}_T$ for $T \in \{n-t_s, n-t_r\}$ for the first time, set $\texttt{Finish}_T = \top$ and multicast to all parties.

No Fast Lock: Upon $\texttt{Finish}_{n-t_r} = \top$, if P_i does not have $\text{Lock}^{vc,s}_{n-t_r}$ certificate for P_s, it multicasts a message $(\texttt{NoLock}, s, k, n-t_r)$.

No Slow Lock: Upon $\texttt{Finish}_{n-t_s} = \top$, if P_i does not for any $T \in \{t_s, t_r\}$ have a valid $\text{Lock}^{vc,s}_{T}$ certificate for P_s, it multicasts a message $(\texttt{NoLock}, s, k, n-t_s)$.

Forward Slow Lock: Upon $\texttt{Finish}_{n-t_s} = \top$ and having received a valid certificate $\text{Lock}^{vc,s}_{n-t_s}$ for the first time, P_i multicasts $\text{Lock}^{vc,s}_{n-t_s}$ and records the current time as τ_i.

Forward and Confirm Fast lock: Upon having received a valid certificate, $\text{Lock}^{vc,s}_{n-t_r}$ for the first time, for P_s on vector commitment vc, P_i multicasts $\text{Lock}^{vc,s}_{n-t_r}$. Unless $\texttt{Finish}_{n-t_s} = \top$ and P_i has received a $\text{Lock}^{vc',s}_{n-t_s}$ where $vc' \neq vc$, P_i additionally multicasts a vote on vc in the form of a partial signature for threshold $t_s + 1$ on the string $(\texttt{fastvote}, s, k, vc)$ along with forwarding $\text{Lock}^{vc,s}_{n-t_r}$.

Equivocation Vote: Upon $\texttt{Finish}_{n-t_s} = \top$ and having received two valid lock-certificates $\text{Lock}^{vc,s}_{n-t_s}$ and $\text{Lock}^{vc',s}_{T}$ for $T \in \{n-t_s, n-t_r\}$ for P_s on conflicting vector commitments $vc' \neq vc$, P_i multicasts a vote for $\perp$ in the form of a partial signature for threshold $n-t_s$ on the string $(\texttt{Vote}, s, k, \perp)$.

Not Locked Vote: Upon $\texttt{Finish}_T = \top$ and having received $(\texttt{NoLock}, s, k, T)$ from T parties, P_i multicasts a vote for $\perp$ in the form of a partial signature for threshold $n-t_s$ on the string $(\texttt{Vote}, s, k, \perp)$.

Non-Equivocation Vote: After time $\tau_i + 2\Delta$, if at some point all of the following holds: P_i has received a valid $\text{Lock}^{vc,s}_{n-t_s}$, has not received a valid $\text{Lock}^{vc',s}_{n-t_s}$ where $vc' \neq vc$, and not received a valid $\text{Lock}^{vc'',s}_{n-t_r}$ for any vc'' (equal or different to vc), then P_i multicasts a vote for vc in the form of a $(n-t_s)$-threshold partial signature on the string $(\texttt{Vote}, s, k, vc)$.

Slow Certify Unique: Upon having received $t_s + 1$ distinct partial signatures on $(\texttt{Vote}, s, k, vc)$, P_i combines them into a (t_s+1)-threshold signature $\text{Unique}^{\text{vc},s,k}_{n-t_s}$, multicasts and outputs $\text{Unique}^{\text{vc},s,k}_{n-t_s}$ then terminates.

Fast Certify Unique: Upon having received t_s+1 valid partial signatures on $(\texttt{fastvote}, s, k, vc)$, P_i combines the partial signatures obtaining the t_s+1-threshold signature $\text{Unique}^{\text{vc},s,k}_{n-t_r}$, multicasts and outputs $\text{Unique}^{\text{vc},s,k}_{n-t_r}$ then terminates.

Certify Skip: Upon receiving $n-t_s$ partial signatures on $(\texttt{Vote}, s, k, \perp)$, if $\texttt{outcome}_{s,k} = \perp$, P_i combines the partial signature obtaining the $(n-t_s)$-threshold signature $\text{NoLock}^{s,k}$, multicasts and outputs $\text{NoLock}^{s,k}$ then terminates.

Fig. 4. Protocol for Dispersal Check with a single sender P_s.

except with negligible probability. If the reconstructed value satisfies the external predicate V, it is output and the protocol terminates; otherwise, the protocol proceeds to the next iteration.

3.5 BumbleBee Proof Sketch

We sketch why BumbleBee satisfies external validity, agreement, and termination under the optimal thresholds $n > 2t_s + t_a$ and $n > t_s + 2t_r$.

$\Pi_{\mathsf{RC}}(\rho)[s]$

Each party $\mathsf{P}_i \in \mathcal{P}$ holds an initially empty map of sets C.

Input: Upon receiving input $(store, cert)$,
- If $store \neq \perp$ parse it as $(\texttt{store}, s_i, \pi_i, \mathsf{vc})$ and send it to all parties.
- If $cert \neq \perp$ parse it as a valid $\texttt{Reconstruct}^{\mathsf{vc},s,k}$ certificate where $\mathsf{vc} \neq \perp$ and send it to all parties.

Receive certificate: Upon receiving either a valid $\texttt{Reconstruct}^{\mathsf{vc},s,k}$ certificate for P_s on vector commitment vc for the first time, send it to all parties and wait until $|C[\mathsf{vc}]| \geq \rho$. Once this happens reconstruct the message defined by the ρ authenticated shares. If the reconstructed message m satisfies $\mathsf{vc} := \mathsf{VC.VCom}(\mathsf{Enc}(m, \rho))$ output m, otherwise output $\perp$.

Receive share: Upon receiving $(\texttt{store}, s_j, \pi_j, \mathsf{vc})$ from P_j for the first time do:
- If $\mathsf{VC.Verify}(\mathsf{vc}, j, s_j, \pi_j) = 1$ then, $C[\mathsf{vc}] \leftarrow C[\mathsf{vc}] \cup \{(j, s_j)\}$
- else discard the invalid message.

Fig. 5. Protocol for recast with a single sender P_s.

BumbleBee$^{\mathsf{Pred}}$

Let $\rho = n - 2t_s$. Each party P_i executes the following logic to disperse its inputs and establish certificates of dispersal.

Parallel Disperse: Upon receiving input m, P_i invokes $\Pi_{\mathsf{PPD}}(\rho)$ with input m.

Leader Election: Upon receiving output $(S^{Store}, S^{Lock}_{n-t_s}, \texttt{Finish}_{n-t_s})$ or $(S^{Store}, S^{Lock}_{n-t_r}, \texttt{Finish}_{n-t_r})$ from Π_{PPD}, P_i invokes $\Pi_{\mathsf{CC}}[1]$ runs the following logic for $k \in 1, 2, 3, \ldots$:
- Let P_s be the output of $\Pi_{\mathsf{CC}}[k]$ and invoke $\Pi_{\mathsf{DC}}[s, k]$ on the output from $\Pi_{\mathsf{PPD}}(\rho)$.
- Upon receiving valid certificate c from $\Pi_{\mathsf{DC}}[s, k]$, if c is a $\mathsf{Skip}^{s,k}$, let $b = 0$, otherwise let $b = 1$. Invoke $\Pi_{\mathsf{VBA}}{}^{\mathsf{Validateln}_k}$ with input b justified by c
- Upon receiving output b' justified by c' from $\Pi_{\mathsf{VBA}}{}^{\mathsf{Validateln}_k}$. If $b' = 0$, invoke $\Pi_{\mathsf{CC}}[k+1]$ and proceed to the next iteration, otherwise invoke $\mathsf{RC}(\rho)[s, k]$ with input $(store, c')$.
- Let m' be the output of $\Pi_{\mathsf{RC}}(\rho)[s, k]$. If $m = \perp$ or $\mathsf{Pred}(m) = \perp$, invoke $\Pi_{\mathsf{CC}}[k+1]$ and go to the next iteration. Otherwise output m and terminate.

Fig. 6. BumbleBee: Protocol for multivalued validated Byzantine Agreement.

Correctness of dispersal is inherited from Π_{PPD}. If an honest party outputs a lock (or done) certificate for a commitment vc at threshold $T \in \{n - t_r, n - t_s\}$, then at least $T - t_s$ of the signers are honest and therefore hold valid coded fragments consistent with vc. With reconstruction threshold $\rho = n - 2t_s$, any $n - t_s$ lock contains at least $n - 2t_s = \rho$ honest fragment holders, so reconstructability follows immediately once a slow lock exists; fast locks only strengthen this guarantee.

Dispersal Check (Π_{DC}) enforces *uniqueness*: it outputs either a uniqueness certificate $\texttt{Unique}^{\mathsf{vc},s,k}_T$ for some $T \in \{n - t_r, n - t_s\}$ or a skip-certificate. In the fast case $T = n - t_r$, uniqueness follows from honest intersection implied by $n > t_s + 2t_r$, so two valid fast uniqueness certificates cannot exist for different commitments. In the slow case $T = n - t_s$, Π_{DC} leverages synchrony: within bounded time, honest parties either observe conflicting locks (forcing skip) or converge on a single locked commitment and certify it. In asynchrony with $f \leq t_a$, slow certificates regain honest intersection from $n > 2t_s + t_a$, so conflicting slow certificates cannot be formed.

Agreement follows from the binary validated agreement Π_{VBA} layer together with the above uniqueness guarantee: a party inputs 1 to Π_{VBA} only when it holds a valid uniqueness certificate,

and by provable agreement the output bit is accompanied by a valid certificate. Hence if any honest party proceeds to reconstruction, all honest parties that output do so for the same commitment vc and reconstruct the same value by position binding of the vector commitment. Validity holds because reconstructed messages are checked against vc and the external predicate V before output.

Termination holds in three regimes. If $f \leq t_r$, fast certificates form without waiting for timeouts and Π_{DC} outputs a fast uniqueness certificate, yielding expected $O(\delta)$ time (Responsiveness). If $f \leq t_s$ and the network is synchronous, slow certificates form within $O(\Delta)$ and Π_{DC} completes within bounded time, so the protocol terminates in $O(\Delta)$. If $f \leq t_a$ and the network is asynchronous, termination follows from the asynchronous guarantees of Π_{PPD} and the intersection implied by $n > 2t_s + t_a$, together with the standard leader-election/VBA progress argument inherited from the Dumbo-style MVBA architecture.

Communication is dominated by dispersal. Each party sends n fragments of size ℓ/ρ, yielding total cost $O(n^2\ell/\rho + n^2\kappa)$. When $\rho = n - 2t_s = \Theta(n)$, this simplifies to $O(n\ell + n^2\kappa)$, matching the asymptotic optimality of Dumbo.

4 BUMBLEBEE+ MVBA PROTOCOL

The construction of Section 3 achieves optimal communication whenever $n - 2t_s = \Theta(n)$, since reconstruction requires $\rho = n - 2t_s$ honest fragments. When $n - 2t_s$ becomes small, however, the redundancy of the erasure code increases by a factor of $n/(n - 2t_s)$, potentially leading to super-linear overhead. In this section we remove this assumption by setting the reconstruction threshold to $\rho = n - t_s - t_r$. Since $t_s + 2t_r < n$, we have $\rho > n/4$, guaranteeing constant-rate encoding independently of $n - 2t_s$. This yields linear communication in the message size ℓ without requiring $n - 2t_s = \Theta(n)$.

High-Level Structure. Importantly, BumbleBee+ preserves the overall architecture of the original protocol. The Parallel Provable Dispersal (Π_{PPD}) and Recast (Π_{RC}) components remain unchanged, except that they now invoke the erasure coding scheme with reconstruction threshold $\rho = n - t_s - t_r$ instead of $n - 2t_s$. All agreement logic and threshold signatures remain identical. The only structural modification occurs in the dispersal-check layer. Since reconstruction is no longer guaranteed from an $n - t_s$ lock-certificate alone, we introduce a refined Dispersal Check protocol $\Pi_{\text{DC+}}$ that explicitly verifies reconstruction feasibility before committing to a leader. In particular, $\Pi_{\text{DC+}}$ ensures that either sufficient evidence exists to guarantee successful reconstruction, allowing parties to safely proceed to the Recast phase, or the leader is skipped and a new leader is elected.

The Core Difficulty. Lowering the reconstruction threshold introduces a fundamental subtlety. If only an $n - t_s$ lock-certificate exists, then at most $(n - t_s) - t_s = n - 2t_s$ honest parties are guaranteed to hold a fragment. Since $\rho = n - t_s - t_r$ may exceed $n - 2t_s$, reconstruction is no longer guaranteed from a slow certificate alone. By contrast, if an $n - t_r$ lock-certificate exists, then at least $(n - t_r) - t_s = n - t_s - t_r = \rho$ honest parties hold fragments, and reconstruction is immediately safe.

BumbleBee$^+$ resolves this by performing an explicit reconstruction feasibility check inside the $\Pi_{\text{DC+}}$ phase whenever only a slow certificate is available, which we further explain in details below.

4.1 Dispersal Check+ protocol

The $\Pi_{\text{DC+}}(\rho)[s, k]$ protocol builds on the original dispersal-check logic in order to guarantee reconstruction with the lower reconstruction threshold of $\rho = n - t_s - t_r$. As in the simpler construction, its goal is to certify whether the elected leader P_s has dispersed a unique reconstructable value in iteration k, or whether the leader must be skipped. The key difference is that reconstruction feasibility must now be explicitly verified whenever only a slow lock-certificate, $\text{Lock}_{n-t_s}^{\text{vc},s,k}$ is available.

The $\Pi_{\text{DC+}}$ protocol goes as follows. First, each party first invokes the base dispersal-check protocol $\Pi_{\text{DC}}[s,k]$. Upon outputting, or upon receiving for the first time, a certificate of the form $\texttt{Unique}^{\text{vc},s,k}_{n-t_r}$, $\texttt{Unique}^{\text{vc},s,k}_{n-t_s}$, or $\text{NoLock}^{s,k}$, the party multicasts the certificate to all other parties. This ensures that certificates propagate quickly if the network is synchronous and that all honest parties are within bounded time apart in the protocol steps.

If a valid fast uniqueness certificate $\texttt{Unique}^{\text{vc},s,k}_{n-t_r}$ is observed, or a valid no lock-certificate $\text{NoLock}^{s,k}$ is observed, the party immediately outputs the certificate and terminates the protocol instance. A fast certificate guarantees that at least $(n-t_r)-t_s = n-t_s-t_r = \rho$ honest parties hold valid fragments, and therefore reconstruction is immediately safe. Thus the responsive behavior of the original construction is preserved. If instead only a slow uniqueness certificate $\texttt{Unique}^{\text{vc},s,k}_{n-t_s}$ is observed and $k \leq \lambda$, the party needs to verify that the value committed using vc is reconstructable. At this point the party records the current time and begins explicitly verifying that reconstruction is feasible.

To perform this verification, whenever a party receives for the first time a valid $(\texttt{store}, s_i, \pi_i, \text{vc})$ message corresponding to the commitment vc (which it may have received during the Π_{PPD} protocol), it multicasts this fragment to all other parties. This guarantees that if sufficiently many honest parties hold fragments, they will eventually become known to all honest parties in synchrony.

Upon collecting $n-t_s-t_r = \rho$ valid fragments corresponding to the same commitment vc appearing in the slow uniqueness certificate, the party attempts to reconstruct a message m. It verifies that the reconstructed message is consistent with the commitment by checking that $\text{vc} = \text{VC.VCom}(\text{Enc}(m,\rho))$. If this check succeeds, the party re-encodes m under threshold ρ, redistributes the corresponding fragments, and multicasts a partial signature with threshold t_s+1 on the string $(\texttt{Redispersed}, s, k, \text{vc})$. Furthermore, it encodes the message and sends $(\texttt{store}, s_j, \pi_j, \text{vc})$ to every party P_j.

When t_s+1 such partial signatures are collected, they are combined into a (t_s+1)-threshold signature $\text{Redisperse}^{\text{vc},s,k}$. This certificate serves as explicit evidence that reconstruction has succeeded and that at least ρ honest parties now hold valid fragments. The party multicasts this certificate, outputs it, and terminates the dispersal-check instance.

If reconstruction does not succeed within 2Δ time from the moment the slow certificate was first observed, the party signs a timeout message $(\texttt{Timeout}, s, k)$. Once t_s+1 partial signatures on this message are collected, they are combined into a timeout certificate $\text{TimeoutCert}^{s,k}$, which is multicast and output. The timeout certificate indicates that reconstruction feasibility could not be established under synchronous assumptions and that the leader should be skipped.

Finally, if $k > \lambda$ and a valid slow uniqueness certificate is observed, the party outputs it immediately without attempting reconstruction. After λ unsuccessful reconstruction attempts in previous iterations, the probability that the network is synchronous and that we missed electing an honest leader is at most $(n-t_s-t_r/n)^{\lambda}$. Except with negligible probability, the execution must therefore be asynchronous, in which case an $n-t_s$ lock-certificate suffices for reconstruction as in the original construction.

4.2 Constructing The BumbleBee+ Protocol

As in the original construction, each party P_s begins by invoking $\Pi_{\text{PPD}}(\rho)$ with input m_s to disperse its value and establish the corresponding store, lock, and finish certificates. The only modification is that the erasure code now uses a different reconstruction threshold, $\rho = n-t_s-t_r$.

Once a party obtains either a slow finish certificate $\texttt{Finish}_{n-t_s}$ or a fast finish certificate $\texttt{Finish}_{n-t_r}$, the dispersal phase concludes and the protocol proceeds to leader selection. The

$\Pi_{\text{DC+}}(\rho)[s,k]$

The following logic is run by all parties $\mathsf{P}_i \in \mathcal{P}$ to certify that a unique message was dispersed by P_s.

Invoke DC **and Forward Certs:** Each P_i invokes $\mathsf{DC}[s,k]$, and upon outputting or receiving a certificate of the form $\mathtt{Unique}^{vc,s,k}_{n-t_r}$, $\mathtt{Unique}^{vc,s,k}_{n-t_s}$, or $\mathsf{NoLock}^{s,k}$ for the first time from $\mathsf{DC}[s,k]$, P_i multicasts it to all parties.

Fast Decide: Upon having received a valid $\mathtt{Unique}^{vc,s,k}_{n-t_r}$ or $\mathsf{NoLock}^{s,k}$ certificate, P_i outputs the certificate and terminates. If $k > \lambda$, and received a valid $\mathtt{Unique}^{vc,s,k}_{n-t_s}$, P_i outputs the certificate and terminates.

Slow Decide: Upon receiving input $\mathtt{Unique}^{\mathsf{vc},s,k}_{n-t_s}$ and $k \leq \lambda$, record current time τ_i.

Send Own Share: Upon receiving a valid $(\mathtt{store}, s_i, \pi_i, \mathsf{vc})$ message for the first time, P_i multicasts that message to all parties.

Reconstruct and Redisperse: Upon receiving $n - t_s - t_r$ valid shares of the form $(\mathtt{store}, s_j, \pi_j, \mathsf{vc})$ that verify w.r.t. the same vc appearing in $\mathtt{Unique}^{\mathsf{vc},s,k}_{n-t_s}$, P_i reconstructs a candidate message m from these shares. If $\mathsf{vc} = \mathsf{VC.VCom}(\mathsf{Enc}(m,\rho))$, then P_i re-encodes m as $\mathsf{Enc}(m,\rho)$, sends $(\mathtt{store}, s_j, \pi_j, \mathsf{vc})$ to every party P_j, and multicasts a partial signature with threshold $t_s + 1$ on the string $(\mathtt{Redispersed}, s, k, \mathsf{vc})$.

Certify Redispersed: Upon receiving $t_s + 1$ valid partial signatures on $(\mathtt{Redispersed}, s, k, \mathsf{vc})$, P_i combines them into a (t_s+1)-threshold signature $\mathsf{Redisperse}^{\mathsf{vc},s,k}$, multicasts and outputs $\mathsf{Redisperse}^{\mathsf{vc},s,k}$, and then terminates.

Timeout: At time $\tau_i + 2\Delta$, send a partial signature on the string $(\mathtt{Timeout}, s, k)$. At time $\tau_i + 2\Delta$, if P_i has not received , it multicasts a partial signature with threshold $t_s + 1$ on the string $(\mathtt{Timeout}, s, k)$.

Certify Timeout: Upon receiving $t_s + 1$ valid partial signatures on $(\mathtt{Timeout}, s, k)$, P_i combines them into a (t_s+1)-threshold signature $\mathsf{TimeoutCert}^{s,k}$, multicasts and outputs $\mathsf{TimeoutCert}^{s,k}$, and then terminates.

Fig. 7. Protocol for low redundancy dispersal check with a single sender P_s.

protocol then enters a leader-election loop, repeated for at most λ iterations. Iterations are indexed by $k = 1, 2, \ldots, \lambda$, and in each iteration a leader P_s is elected via the common coin protocol $\Pi_{\text{CC}}[k]$.

For the elected leader, parties invoke the new dispersal-check protocol $\Pi_{\text{DC+}}[s,k]$. The output of $\Pi_{\text{DC+}}[s,k]$ is a certificate c that determines the binary input to the validated Byzantine agreement Π_{VBA} layer. If c is a skip-certificate $\mathsf{NoLock}^{s,k}$ or a timeout certificate $\mathsf{TimeoutCert}^{s,k}$, the party invokes Π_{VBA} with input 0 justified by c. If c is a fast uniqueness certificate $\mathtt{Unique}^{\mathsf{vc},s,k}_{n-t_r}$, or a slow uniqueness certificate paired with a valid re-dispersal certificate $\mathsf{Redisperse}^{\mathsf{vc},s,k}$, the party invokes Π_{VBA} with input 1 justified by c. Crucially, Π_{VBA} satisfies provable agreement: any output bit must be accompanied by a valid justification. Therefore, no party can cause agreement on 1 without presenting either a fast uniqueness certificate or a successfully validated slow certificate. This ensures that agreement decisions are always backed by explicit evidence that reconstruction is guaranteed.

If Π_{VBA} outputs 0, the leader is skipped and the protocol proceeds to the next iteration. If Π_{VBA} outputs 1, the accompanying certificate identifies a uniquely reconstructable commitment vc. Parties then invoke the recast protocol $\Pi_{\text{RC}}(\rho)[s,k]$ to reconstruct the leader's value. Since $\Pi_{\text{DC+}}$ guarantees that reconstruction feasibility has already been verified, decoding succeeds except with negligible probability. If the reconstructed message satisfies the external validity predicate V, it is output and the protocol terminates. Otherwise, the protocol proceeds to the next iteration.

4.3 BumbleBee+ Proof Sketch

BumbleBee$^+$ uses the smaller reconstruction threshold $\rho = n - t_s - t_r$, which ensures constant-rate erasure coding under $t_s + 2t_r < n$. The core new difficulty is that an $n - t_s$ lock no longer automatically implies ρ honest fragment holders. Accordingly, the only structural change is the refined dispersal-check wrapper $\Pi_{\text{DC+}}$.

Soundness of the *fast* path is unchanged: if $\Pi_{\text{DC+}}$ outputs a fast uniqueness certificate at threshold $n-t_r$, then at least $(n-t_r)-t_s = n-t_s-t_r = \rho$ honest parties hold valid fragments, so reconstruction is guaranteed and uniqueness follows from honest intersection as in BumbleBee.

For the *slow* path, $\Pi_{\text{DC+}}$ explicitly verifies reconstructability. Upon observing a slow uniqueness certificate $\texttt{Unique}^{\text{vc},s,k}_{n-t_s}$, parties multicast their stored fragments for vc and attempt to reconstruct from ρ valid fragments. If reconstruction succeeds, parties produce a re-dispersal certificate $\text{Redisperse}^{\text{vc},s,k}$ (threshold $t_s + 1$), which certifies that enough honest parties have obtained consistent fragments; only then can a party vote 1 in Π_{VBA} based on the slow path. If reconstruction does not succeed within bounded time in synchrony, $\Pi_{\text{DC+}}$ outputs a timeout/noLock-certificate, ensuring safety by preventing acceptance of a non-reconstructable leader.

Agreement and validity follow as in BumbleBee: any Π_{VBA} output of 1 must be justified by either a fast uniqueness certificate or a slow certificate paired with $\text{Redisperse}^{\text{vc},s,k}$, so all honest parties that reconstruct do so for the same commitment and obtain the same value by position binding, and output only if $V(m) = \top$.

Termination is as follows. If $f \le t_r$, the fast path produces a valid $n - t_r$ certificate and $\Pi_{\text{DC+}}$ immediately accepts it, yielding expected $O(\delta)$ time. In synchronous executions with $f \le t_s$, once an honest leader is elected, reconstruction succeeds within $O(\Delta)$ and produces $\text{Redisperse}^{\text{vc},s,k}$ when needed, so the protocol terminates in $O(\Delta)$. To avoid indefinite skipping under asynchrony (where timeouts may spuriously trigger), parties run the slow-path reconstruction test for at most λ iterations; if all fail, they switch to the original BumbleBee logic. Except with negligible probability in λ, repeated failure implies the network is asynchronous (or that λ consecutive leaders were corrupted), and after switching, termination follows from the asynchronous guarantees and the fact that under $f \le t_a$ there are at least $n - t_s - t_a \ge \rho$ honest fragment holders, making reconstruction feasible.

The communication cost of dispersal is $O(n^2 \frac{\ell}{\rho} + n^2\kappa)$. Since $\rho = n - t_s - t_r > tr > n/4$, this simplifies to $O(n\ell + n^2\kappa)$ in synchrony. In asynchronous executions, at most λ iterations may be attempted before switching, leading to a multiplicative λ overhead in communication. Time complexity remains unchanged from the original construction.

5 BUMBLEBEE SECURITY ANALYSIS

We begin by defining a notion of PPD-honesty, which will be used throughout the security analysis to characterize the behavior of honest senders under different timing conditions.

BumbleBee+$^{\text{Pred}}$

Let $\rho = n - t_s - t_r$. Each party P_i executes the following logic disperse its inputs and establish certificates of dispersal.

Parallel Disperse: Upon receiving input m, P_i invokes $\mathsf{PPD}(\rho)$ with input m.

Leader Election: Upon receiving output $(S^{Store}, S^{Lock}_{n-t_s}, \mathtt{Finish}_{n-t_s})$ or $(S^{Store}, S^{Lock}_{n-t_r}, \mathtt{Finish}_{n-t_r})$ from PPD, P_i invokes $\Pi_{\mathrm{CC}}[k]$ for $k \in 1, 2, 3, \ldots$ and runs the following logic :

- Let P_L be the output of $\Pi_{\mathrm{CC}}[k]$ and invoke $\Pi_{\mathrm{DC+}}[l, k]$ on the output of PPD.
- Upon receiving valid certificate c from $\Pi_{\mathrm{DC+}}[l, k]$, if c is a valid $\mathsf{NoLock}^{s,k}$ certificate or a $\mathsf{TimeoutCert}^{s,k}$, P_i invokes VBA with input $0, c$. Else, if c is a valid $\mathtt{Unique}^{\mathsf{vc},s,k}_{T}$ for $T \in \{n - t_s, n - t_r\}$ or a valid $(\mathtt{Unique}^{\mathsf{vc},s,k}_{n-t_s}, \mathsf{Redisperse}^{\mathsf{vc},s,k})$ pair, P_i invokes $\mathsf{VBA}^{\mathsf{ValidateIn}_k}$ with input $1, c$
- Upon receiving output b'', c'' from $\mathsf{VBA}^{\mathsf{ValidateIn}_k}$. If $b'' = 0$, invoke $\Pi_{\mathrm{CC}}[k+1]$ and proceed to the next iteration, otherwise invoke $\mathsf{RC}(\rho)[l, k]$ with input $(store, c'')$
- Let m' be the output of $\mathsf{RC}(\rho)[l, k]$. If $m = \perp$ or $\mathsf{Pred}(m) = \perp$, invoke $\Pi_{\mathrm{CC}}[k+1]$ and go to the next iteration. Otherwise output m and terminate.

Fig. 8. BumbleBee+: Protocol for multivalued validated Byzantine Agreement.

Definition 5.1 (Fast and Slow PPD-Honest). We say that a party P_i is τ-Fast PPD-honest if it is honest and outputs $\mathtt{Done}^{i}_{n-t_r}$ at time τ. Similarly, we say that a party it is τ-Fast PPD-honest if it is honest and outputs $\mathtt{Done}^{i}_{n-t_s}$ at time τ.

Lemma 5.2 (PPD-Honest Sender). *Assume the network is synchronous and all honest parties activate PD$[s]$ at time τ. If the sender is $\tau + 4\Delta$-slow* PPD*-honest, every honest party outputs a valid* $\mathsf{Store}^{\mathsf{vc},s}$ *and a valid* $\mathsf{Lock}^{\mathsf{vc},s}_{n-t_s}$ *by time $t \leq \tau + 3\Delta$. Moreover, the sender* P_s *outputs a valid* $\mathtt{Done}^{s}_{n-t_s}$ *by time $\tau + 4\Delta$.*

Proof. By synchronous network assumption, there are at most t_s corruptions. Furthermore, since the sender is $\tau + 4$ slow PPD-honest, then it output $\mathtt{Done}^{s}_{n-t_s}$ at time $\tau + 4$ before getting corrupted. First, since P_s is honest, it sends to every party P_j a message of the form $(\mathtt{store}, s_j, \pi_j, vc)$ where $\mathsf{VC.Verify}(\mathsf{vc}, \mathtt{store}, s, \pi) = 1$. Hence, every honest party can verify that its share is valid and outputs a valid $\mathsf{Store}^{\mathsf{vc},s_j}$. Next we argue that P_s can form a valid lock-certificate, and that every honest party outputs it. Since all honest parties follow the protocol and do not abandon $\Pi_{\mathrm{PD}}[s]$ before time $\tau + 5\Delta$, every honest party that receives a valid store message signs and sends $(\mathtt{Stored}_{n-t_s}, vc, s)$ to P_s. Hence, P_s collects at least $n - t_s$ valid signatures a time $\tau' \leq \tau + 2\Delta$ and can form a $\mathsf{Lock}^{\mathsf{vc},s}_{n-t_s}$ certificate. Finally, after broadcasting the valid lock $\mathsf{Lock}^{\mathsf{vc},s}_{n-t_s}$, all honest parties receive the valid lock by time $\tau'' \leq \tau + 3\Delta$, signs and send $(\mathtt{locked}_{n-t_s}, s)$ to P_s. Finally, P_s collects enough signature shares outputs a valid $\mathtt{Done}^{s}_{n-t_s}$. □

Lemma 5.3 (No Honest is Abandoned on Synchronous Path). *Assume the network is synchronous. Assume some party produces a* $\mathtt{Finish}_{n-t_s}$ *certificate at time τ. Then every party who is honest at time τ is a τ'-slow* PPD*-honest sender for some time $\tau' < \tau$.*

Proof. By Lemma 5.2, the slow PPD-honest sender P_s outputs a valid $\mathtt{Done}^{s}_{n-t_s}$ within 4Δ of receiving its input. A valid $\mathtt{Finish}_{n-t_s}$ certificate consists of $n - t_s$ valid signatures on string $\mathtt{Finish}_{n-t_s}$. A party computes such partial signature if it observes $t_s + 1$ valid partial signatures on $\mathtt{Ready}_{n-t_s}$. Since at most t_s parties are corrupted, at least one of these $\mathtt{Ready}_{n-t_s}$ signatures must be produced by an honest party. By the PPD code, an honest party issues a $\mathtt{Ready}_{n-t_s}$ signature

only after waiting 5Δ from the time it received its input. Therefore, no $\texttt{Finish}_{n-t_s}$ certificate can be formed before time 5Δ, whereas $\texttt{Done}^{s}_{n-t_s}$ is output by time 4Δ. □

Lemma 5.4 (Lock Implies $(T-t)$-Dispersion). *Let t denote the maximum number of corruptions. If an honest party receives a valid $\mathsf{Lock}^{vc,s}_{T}$ certificate, then $T-t$ forever honest parties P_i received valid store certificates $(\texttt{store}, s_i, \pi, vc)$.*

Proof. By assumption the $\mathsf{Lock}^{vc,s}_{T}$ is valid. Thus, $\mathsf{Lock}^{vc,s}_{T}$ consists of T valid signature shares from distinct parties. As at most t parties are corrupted, at least $T-t$ of these shares were produced by honest parties. By protocol definition, an honest party P_i produces such a share only after receiving a valid store certificate $(\texttt{store}, s_i, \pi, vc)$. Hence at least $T-t$ forever honest parties received valid store certificates. □

Lemma 5.5 (Done Implies Unmissable Stored). *Let t denote the maximum number of corruptions. Assume a party P_s outputs a valid Done^{s}_{T} certificate, then $T-t$ forever honest parties P_i received valid store certificates $(\texttt{store}, s_i, \pi, vc)$.*

Proof. Since $\mathsf{ValidateDone}(T, \mathsf{Done}^{s}_{T}) = 1$, the certificate Done^{s}_{T} consists of T valid signature shares on $(\texttt{locked}_T, s)$ from distinct parties. By the protocol definition, a party produces such a share only after receiving a valid $\mathsf{Lock}^{vc,s}_{T}$ certificate. Hence at least $T-t$ of these shares were produced by honest parties that received a valid lock-certificate. By Lemma 5.4, each such honest party must have received a valid store certificate $(\texttt{store}, s_i, \pi, vc)$. □

Lemma 5.6 (No Conflicting Locks in Asynchrony). *Assume the network is asynchronous. Fix a sender P_s. Then two valid conflicting lock certificates $\mathsf{Lock}^{vc,s}_{T}$ and $\mathsf{Lock}^{vc',s}_{T'}$ with $vc \neq vc'$ cannot exist for $T, T' \in \{n-t_s, n-t_r\}$.*

Proof. As honest parties honest parties send their contribution to the two different $\mathsf{Lock}^{vc,s}$ thresholds in one message, it suffices to prove that two lock-certificates with threshold $n-t_s$ cannot both be valid for conflicting values. Indeed, such certificates have weaker quorum-intersection guarantees than locks with thresholds $n-t_s$ and $n-t_r$ as $n > t_s + 2t_r$ and $t_s > t_r$, so ruling out this case implies the stronger cases as well. Suppose for contradiction that both $\mathsf{Lock}^{vc,s}_{n-t_s}$ and $\mathsf{Lock}^{vc',s}_{n-t_s}$ are valid. By definition of $\mathsf{ValidateLock}()$, each certificate contains $n-t_s$ signature shares from distinct parties on the corresponding lock message for the same sender P_s. Let Q and Q' denote the sets of signers of the two certificates. Their intersection satisfies $|Q \cap Q'| \geq |Q| + |Q'| - n = 2(n-t_s) - n = n - 2t_s$. Since $n > 2t_s + t_a$, we have $n - 2t_s > t_a$, and thus $|Q \cap Q'| > t_a$. Therefore $Q \cap Q'$ contains at least one honest party. An honest party can not produce signature shares for two different lock messages corresponding to conflicting values $vc \neq vc'$ for the same sender P_s. Therefore two conflicting valid lock-certificates cannot both exist. □

Lemma 5.7 (No Conflicting Locks on Fast Path). *Fix a sender P_s. Then two valid conflicting lock certificates $\mathsf{Lock}^{vc,s}_{n-t_r}$ and $\mathsf{Lock}^{vc',s}_{n-t_r}$ with $vc \neq vc'$ cannot exist.*

Proof. Assume for contradiction that there exist two valid conflicting lock-certificates $\mathsf{Lock}^{vc,s}_{n-t_r}$ and $\mathsf{Lock}^{vc',s}_{n-t_r}$ with $vc \neq vc'$. By definition of $\mathsf{ValidateLock}()$, each certificate consists of $n-t_r$ signature shares from distinct parties. Let Q and Q' be the sets of signers of $\mathsf{Lock}^{vc,s}_{n-t_r}$ and $\mathsf{Lock}^{vc',s}_{n-t_r}$, respectively. Then $|Q| \geq |Q'| \geq n-t_r$, and hence $|Q \cap Q'| \geq |Q| + |Q'| - n = 2(n-t_r) - n = n - 2t_r$. By the assumption $n > t_s + 2t_r$, we have $n - 2t_r > t_s$. Since at most t_s parties are corrupted under any network conditions, it follows that $Q \cap Q'$ contains at least one honest party. Hence, contradiction as honest parties only contributes to either lock.

□

Lemma 5.8 (No Conflicting Locks for Slow PPD-Honest). *Assume that some party* P_s *is* τ*-slow* PPD*-honest. Then, there cannot exist two valid conflicting lock-certificates* $\mathsf{Lock}_T^{vc,s}$ *and* $\mathsf{Lock}_{T'}^{vc',s}$ *with* $vc \neq vc'$ *and* $T, T' \in \{n - t_s, n - t_r\}$.

Proof. If the network is asynchronous, the statement follows from Lemma 5.6. If $T = T' = n - t_r$, the claim follows from Lemma 5.7. So, assume a synchronous network and assume at least one lock has threshold $n - t_s$. We will show that no conflicting $\mathsf{Lock}_{n-t_s}^{\mathsf{vc},s}$ and $\mathsf{Lock}_{T'}^{\mathsf{vc},s}$ can be formed. First observe from definition a τ-slow PPD-honest sender follows the protocol and never produces conflicting shares. By assumption that the sender is τ-slow PPD-honest and from Lemma 5.2, the sender output $\mathsf{Done}_{n-t_s}^{s}$ before any honest party outputs $\mathtt{Finish}_{n-t_s}$. In particular, the sender multicasts $(\mathtt{store}, s_j, \pi_j, \mathsf{vc})$ to all parties and every party stores and responds to this sender's message. Furthermore, according to the algorithm, they do not respond to any new store messages from the sender (which could be sent post-corruption). Suppose for contradiction that there exists a conflicting lock $\mathsf{Lock}_{T'}^{\mathsf{vc}',s}$ with $\mathsf{vc}' \neq \mathsf{vc}$ and $T' \in \{n - t_s, n - t_r\}$. Since honest parties do not send conflicting shares, every share supporting vc' must originate from corrupted parties. However, there are at most t_s corrupted parties, while validity of $\mathsf{Lock}_{T'}^{\mathsf{vc}',s}$ requires at least $T' \geq n - t_s$ distinct shares. Because $n - t_s > t_s$, this is impossible. Therefore, no two valid conflicting lock-certificates can exist. □

Lemma 5.9 (Fast PPD-Honest Unskippable in Asynchrony). *Assume the network is asynchronous. Assume a party* P_s *is* τ*-fast* PPD*-honest and every honest party invokes* $\Pi_{\mathrm{DC}}[s, k]$ *at time* $\tau' \geq \tau$. *Then, no valid* $\mathsf{Skip}^{s,k}$ *can be output from* $\Pi_{\mathrm{DC}}[s, k]$.

Proof. A valid skip-certificate $\mathsf{Skip}^{s,k}$ consists of $n - t_s$ signature shares on the message $(\mathtt{Vote}, s, k, \perp)$. An honest party produces such a share if any of the following cases: (1) it observes two conflicting valid lock-certificates $\mathsf{Lock}_T^{vc,s}$ and $\mathsf{Lock}_T^{vc',s}$ with $vc \neq vc'$, or (2) upon $\mathtt{Finish}_T$, it receives T valid signature shares on messages $(\mathtt{NoLock}, s, k, T)$. A party sends (a) $(\mathtt{NoLock}, s, k, n-t_r)$ only if it has not received a valid $\mathsf{Lock}_{n-t_r}^{\mathsf{vc},s}$ certificate for P_s upon receiving $\mathtt{Finish}_{n-t_r}$, and (b) $(\mathtt{NoLock}, s, k, n - t_s)$ upon receiving $\mathtt{Finish}_{n-t_s}$ only if it has not received any valid $\mathsf{Lock}_T^{\mathsf{vc},s}$ for P_s for any T. For (1), from Lemma 5.6, no two valid conflicting lock certificates $\mathsf{Lock}_T^{vc,s}$ and $\mathsf{Lock}_{T'}^{vc',s}$ with $vc \neq vc'$ can exist for $T, T' \in \{n - t_s, n - t_r\}$. Thus, we only need to argue (2). We distinguish whether $\mathtt{Finish}_{n-t_r}$ or $\mathtt{Finish}_{n-t_s}$ is constructed first. *Case 1:* $\mathtt{Finish}_{n-t_r}$. A valid $\mathsf{Done}_{n-t_r}^{s}$ implies that at least $T - t_a = n - t_r - t_a$ honest parties received valid lock-certificates. So at most t_r honest parties may have not received any lock. Therefore, the total number of parties that may sign $(\mathtt{NoLock}, s, k, T)$ is at most $t_a + t_r$. Since $n > t_s + 2t_r$, $n > 2t_s + t_a$ and $t_s > t_r \geq t_a$, we have $t_a + t_r < n - t_s < n - t_r$, and hence fewer than $n - t_s < n - t_r$ shares on $(\mathtt{NoLock}, s, k, T)$ can be collected, and consequently, no honest party sends $(\mathtt{Vote}, s, k, \perp)$. Consequently, no valid $\mathsf{Skip}^{s,k}$ certificate can be produced. *Case 1:* $\mathtt{Finish}_{n-t_s}$. Similarly, a party sends $(\mathtt{NoLock}, s, k, n - t_s)$ upon receiving $\mathtt{Finish}_{n-t_s}$ only if it has not received any valid $\mathsf{Lock}_T^{\mathsf{vc},s}$ for P_s for any T. From earlier, at most t_r honest parties do not have any locks, and thus, the argument follows from before.

□

Lemma 5.10 (Slow PPD-Honest Maybe Skipped in Asynchrony). *Assume the network is asynchronous. Assume a party* P_s *is* τ*-slow* PPD*-honest, and all honest parties invoke* $\Pi_{\mathrm{DC}}[s, k]$ *later than* τ. *Then either no valid* $\mathsf{Skip}^{s,k}$ *can exist in* $\Pi_{\mathrm{DC}}[s, k]$, *or before a valid* $\mathsf{Skip}^{s,k}$ *can exist at time* τ', *a valid* $\mathtt{Finish}_{n-t_r}$ *certificate exists in the view of some party at time* $\tau'' < \tau'$.

Proof. A valid skip-certificate $\mathsf{Skip}^{s,k}$ consists of $n - t_s$ signature shares on the message $(\mathtt{Vote}, s, k, \perp)$. An honest party produces such a share if any of the following cases: (1) it observes

two conflicting valid lock-certificates $\mathsf{Lock}_T^{vc,s}$ and $\mathsf{Lock}_T^{vc',s}$ with $vc \neq vc'$, or (2) upon $\mathtt{Finish}_T$, it receives T valid signature shares on messages $(\mathtt{NoLock}, s, k, T)$. A party sends (a) $(\mathtt{NoLock}, s, k, n-t_r)$ only if it has not received a valid $\mathsf{Lock}_{n-t_r}^{vc,s}$ certificate for P_s upon receiving $\mathtt{Finish}_{n-t_r}$, and (b) $(\mathtt{NoLock}, s, k, n-t_s)$ upon receiving $\mathtt{Finish}_{n-t_s}$ only if it has not received any valid $\mathsf{Lock}_T^{vc,s}$ for P_s for any T. For (1), from Lemma 5.6, no two valid conflicting lock certificates $\mathsf{Lock}_T^{vc,s}$ and $\mathsf{Lock}_{T'}^{vc',s}$ with $vc \neq vc'$ can exist for $T, T' \in \{n-t_s, n-t_r\}$. Thus, we only need to argue (2). Suppose that some honest party outputs a valid $\mathsf{Skip}^{s,k}$, and let t be the first time at which any honest party outputs such a skip-certificate. The honest party that outputs $\mathsf{Skip}^{s,k}$ at time t must have obtained $n-t_s$ shares on $(\mathtt{Vote}, s, k, \perp)$; since condition (1) is impossible, each honest share among these must have been generated via condition (2), i.e., after observing some $\mathtt{Finish}_T$ and collecting T shares on $(\mathtt{NoLock}, s, k, T)$. We first rule out the possibility that an honest party sends a $(\mathtt{Vote}, s, k, \perp)$ share due to $\mathtt{Finish}_{n-t_s}$. By assumption, party P_s outputs a valid $\mathsf{Done}_{n-t_s}^s$. A valid $\mathsf{Done}_{n-t_s}^s$ implies that at least $T - t_a = n - t_s - t_a > t_s$ honest parties received valid lock-certificates. So at most t_s honest parties may have not received any lock. Therefore, the total number of parties that may sign $(\mathtt{NoLock}, s, k, n-t_s)$ is at most $t_a + t_s$. Since $n > 2t_s + t_a$, we have $t_a + t_s < n - t_s < n - t_r$, and hence fewer than $n - t_s$ shares on $(\mathtt{NoLock}, s, k, n-t_s)$ can be collected, and consequently, no honest party sends $(\mathtt{Vote}, s, k, \perp)$. Therefore, if an honest party outputs a valid $\mathsf{Skip}^{s,k}$ at time t, any honest $(\mathtt{Vote}, s, k, \perp)$ shares contributing to this certificate must have been triggered by $\mathtt{Finish}_{n-t_r}$. Indeed, honest parties send $(\mathtt{Vote}, s, k, \perp)$ only (1) upon observing equivocation in the form $(\mathtt{Finish}_{n-t_r}, \mathsf{Lock}_{n-t_r}^{vc,s})$ and $(\mathtt{Finish}_{n-t_s}, \mathsf{Lock}_{n-t_s}^{vc,s})$ or (2) upon receiving $\mathtt{Finish}_{n-t_r}$ and $n - t_r$ $(\mathtt{NoLock}, s, k, n-t_r)$ messages.

□

Lemma 5.11 (Fast PPD-Honest Maybe skippable in Synchrony). *Assume the network is synchronous. Assume the sender is τ-fast* PPD*-honest and all honest parties invoke $\Pi_{\mathrm{DC}}[s,k]$ upon receiving $\mathtt{Finish}_{n-t_r}$ later than τ. Then either no valid $\mathsf{Skip}^{s,k}$ can exist in $\Pi_{\mathrm{DC}}[s,k]$, or before a valid $\mathsf{Skip}^{s,k}$ can exist at time τ', a valid $\mathtt{Finish}_{n-t_s}$ certificate exists in the view of some party at time $\tau'' \leq \tau'$.*

Proof. A valid skip-certificate $\mathsf{Skip}^{s,k}$ consists of $n - t_s$ signature shares on the message $(\mathtt{Vote}, s, k, \perp)$. An honest party produces such a share if any of the following cases: (1) it observes two conflicting valid lock-certificates $\mathsf{Lock}_T^{vc,s}$ and $\mathsf{Lock}_T^{vc',s}$ with $vc \neq vc'$, or (2) upon $\mathtt{Finish}_T$, it receives T valid signature shares on messages $(\mathtt{NoLock}, s, k, T)$. A party sends (a) $(\mathtt{NoLock}, s, k, n-t_r)$ only if it has not received a valid $\mathsf{Lock}_{n-t_r}^{vc,s}$ certificate for P_s upon receiving $\mathtt{Finish}_{n-t_r}$, and (b) $(\mathtt{NoLock}, s, k, n-t_s)$ upon receiving $\mathtt{Finish}_{n-t_s}$ only if it has not received any valid $\mathsf{Lock}_T^{vc,s}$ for P_s for any T. First, assume that no $\mathtt{Finish}_{n-t_s}$ exists in any parties' view, we show that no valid $\mathsf{Skip}^{s,k}$ can be created. For condition 1, a party may only observe conflicting locks if they are in the the form $(\mathtt{Finish}_{n-t_s}, \mathsf{Lock}_{n-t_s}^{vc,s})$ and $(\mathtt{Finish}_{n-t_r}, \mathsf{Lock}_{n-t_r}^{vc,s})$. From the assumption that $\mathtt{Finish}_{n-t_s}$ does not exist, no party sends $(\mathtt{Vote}, s, k, \perp)$ due to condition (1). For condition (2) By validity of $\mathsf{Done}_{n-t_r}^s$, at least $n - t_r - t_s$ honest parties have valid $\mathsf{Lock}_{n-t_r}^{vc,i}$. These honest parties do not send $(\mathtt{NoLock}, s, k, T)$ message. Thus, at most $t_r + t_s$ parties send $(\mathtt{NoLock}, s, k, T)$, and consequently no honest party sends $(\mathtt{Vote}, s, k, \perp)$ message. Therefore, if an honest party outputs a valid $\mathsf{Skip}^{s,k}$ at time t, any honest $(\mathtt{Vote}, s, k, \perp)$ shares contributing to this certificate must have been triggered by $\mathtt{Finish}_{n-t_s}$. Indeed, honest parties send $(\mathtt{Vote}, s, k, \perp)$ only (1) upon observing equivocation in the form $(\mathtt{Finish}_{n-t_r}, \mathsf{Lock}_{n-t_r}^{vc,s})$ and $(\mathtt{Finish}_{n-t_s}, \mathsf{Lock}_{n-t_s}^{vc,s})$ messages or (2) upon receiving $\mathtt{Finish}_{n-t_s}$ and $n - t_s$ $(\mathtt{NoLock}, s, k, n-t_s)$ messages.

□

Lemma 5.12 (Slow PPD-Honest Maybe skippable in Synchrony). *Assume the network is synchronous. Assume the sender is τ-slow* PPD*-honest and all honest parties invoke DC upon receiving* $\texttt{Finish}_{n-t_s}$ *later than τ. Then either no valid* $\mathsf{Skip}^{s,k}$ *can exist in* $\Pi_{\mathrm{DC}}[s,k]$*, or before a valid* $\mathsf{Skip}^{s,k}$ *can exist at time τ', a valid* $\texttt{Finish}_{n-t_r}$ *certificate exists in the view of some party.*

Proof. As mentioned previously: A valid skip-certificate $\mathsf{Skip}^{s,k}$ consists of $n-t_s$ signature shares on the message $(\texttt{Vote}, s, k, \perp)$. An honest party produces such a share if any of the following cases: (1) it observes two conflicting valid lock-certificates $\mathsf{Lock}_T^{vc,s}$ and $\mathsf{Lock}_T^{vc',s}$ with $vc \neq vc'$, or (2) upon $\texttt{Finish}_T$, it receives T valid signature shares on messages $(\texttt{NoLock}, s, k, T)$. A party sends (a) $(\texttt{NoLock}, s, k, n-t_r)$ only if it has not received a valid $\mathsf{Lock}_{n-t_r}^{\mathrm{vc},s}$ certificate for P_s upon receiving $\texttt{Finish}_{n-t_r}$, and (b) $(\texttt{NoLock}, s, k, n-t_s)$ upon receiving $\texttt{Finish}_{n-t_s}$ only if it has not received any valid $\mathsf{Lock}_T^{\mathrm{vc},s}$ for P_s for any T. For (1), from Lemma 5.8, no two valid conflicting lock certificates $\mathsf{Lock}_{n-t_s}^{vc,s}$ and $\mathsf{Lock}_{T'}^{vc',s}$ with $vc \neq vc'$ can exist for $T \in \{n-t_s, n-t_r\}$. Thus, we only need to argue (2). We first rule out the possibility that an honest party sends a $(\texttt{Vote}, s, k, \perp)$ share due to $\texttt{Finish}_{n-t_s}$ for observing $n-t_s$ messages of $(\texttt{NoLock}, s, k, n-t_s)$. By Lemma 5.2, every honest party outputs a valid $\mathsf{Lock}_{n-t_s}^{\mathrm{vc},s}$ before the sender outputs $\texttt{Done}_{n-t_s}^{s}$. By assumption that the sender is PPD-honest and parties invoke $\Pi_{\mathrm{DC}}[s]$ after $\texttt{Finish}_{n-t_s}$. Hence no honest party ever sends $(\texttt{NoLock}, s, k, n-t_s)$. Therefore at most $t_s < n-t_s$ shares on $(\texttt{NoLock}, s, k, n-t_s)$ can be collected, as no honest party signs $(\texttt{Vote}, s, k, \perp)$. Thus no valid $\mathsf{Skip}^{s,k}$ can be formed. Therefore, if an honest party outputs a valid $\mathsf{Skip}^{s,k}$ at time t, any honest $(\texttt{Vote}, s, k, \perp)$ shares contributing to this certificate must have been triggered by $\texttt{Finish}_{n-t_r}$. Indeed, honest parties send $(\texttt{Vote}, s, k, \perp)$ only (1) upon observing equivocation in the form $(\texttt{Finish}_{n-t_r}, \mathsf{Lock}_{n-t_r}^{\mathrm{vc},s})$ and $(\texttt{Finish}_{n-t_s}, \mathsf{Lock}_{n-t_s}^{\mathrm{vc},s})$ or (2) upon receiving $\texttt{Finish}_{n-t_r}$ and $n-t_r$ $(\texttt{NoLock}, s, k, n-t_r)$ messages □

Lemma 5.13 (Full PPD-Honest Never Skipped). *Assume party* P_s *is both τ-slow Honest and τ'-fast honest. if parties invoke* Π_{DC} *at time τ''* $\max(\tau, \tau')$*, then no valid* $\mathsf{Skip}^{s,k}$ *can exist in* $\Pi_{\mathrm{DC}}[s,k]$.

Proof. If the network is asynchronous, then the proof follow from Lemma 5.9. So assume the network is synchronous. As the sender is t-slow honest, it sends $(\texttt{store}, s_j, \pi_j, \mathrm{vc})$ to every honest party for the same vc, and every honest party receives this message from the sender and responds to i. Furthermore, according to the algorithm, they do not respond to any new sender's store messages. Note that valid $\mathsf{Skip}^{s,k}$ certificate consists of $n-t_s$ signature shares on the message $(\texttt{Vote}, s, k, \perp)$. An honest party produces such a share if any of the following cases: (1) it observes two conflicting valid lock-certificates $\mathsf{Lock}_T^{vc,s}$ and $\mathsf{Lock}_T^{vc',s}$ with $vc \neq vc'$, or (2) upon $\texttt{Finish}_T$, it receives T valid signature shares on messages $(\texttt{NoLock}, s, k, T)$. A party sends (a) $(\texttt{NoLock}, s, k, n-t_r)$ only if it has not received a valid $\mathsf{Lock}_{n-t_r}^{\mathrm{vc},s}$ certificate for P_s upon receiving $\texttt{Finish}_{n-t_r}$, and (b) $(\texttt{NoLock}, s, k, n-t_s)$ upon receiving $\texttt{Finish}_{n-t_s}$ only if it has not received any valid $\mathsf{Lock}_T^{\mathrm{vc},s}$ for P_s for any T. For (1), it can not happen as an honest party never sends equivocating store messages. For (2)(a): since P_s is t'-fast honest, at least $n-t_r$ parties have $\mathsf{Lock}_{n-t_r}^{\mathrm{vc},s}$, and thus, $n-t_r$ $(\texttt{NoLock}, s, k, n-t_r)$ messages cannot be sent due to quorum intersection. For (2)(b): every honest party holds a valid $\mathsf{Lock}_{n-t_s}^{\mathrm{vc},s}$ from Lemma 5.2, and thus, never sends $(\texttt{NoLock}, s, k, n-t_s)$.

□

Lemma 5.14 (Full PPD-Honest is Unique). *Assume a party* P_s *is τ-slow* PPD*-honest and τ'-fast* PPD*-honest, and that every party invokes* $\Pi_{\mathrm{DC}}[s,k]$ *at time $\tau'' > \max(\tau', \tau)$. Then every honest party outputs* $(1, \mathsf{Unique}_T^{\mathrm{vc},s})$ *in* $\Pi_{\mathrm{DC}}[s,k]$*, where* $\mathsf{Unique}_T^{\mathrm{vc},s}$ *is a valid certificate.*

Proof. Recall that $\mathsf{Unique}_T^{\mathrm{vc},s}$ is valid iff either (i) $\mathsf{TS.Verify}_{t_s+1}(\mathsf{PK}, (\texttt{fastvote}, s, k, vc), \mathsf{Unique}_T^{\mathrm{vc},s}) =$ accept when $T = n-t_r$, or (ii) $\mathsf{TS.Verify}_{n-t_s}(\mathsf{PK}, (\texttt{Vote}, s, k, \mathrm{vc}), \mathsf{Unique}_T^{\mathrm{vc},s}) =$ accept when $T = n-t_s$.

From Lemmata 5.6, 5.7 and 5.8, no honest party observes conflicting valid $\mathsf{Lock}_T^{\mathsf{vc},s}$ for sender P_s. Thus, it is enough to argue that each honest party collects enough signature shares to form $\mathsf{Unique}_T^{\mathsf{vc},s}$. We distinguish between $T = n - t_r$ and $T = n - t_s$.

Case 1: $T = n - t_r$**.** Since $\mathsf{ValidateDone}(n - t_r, \mathsf{Done}^s_{n-t_r}) = 1$, the certificate $\mathsf{Done}^s_{n-t_r}$ is a valid threshold signature on $(\mathtt{locked}_{n-t_r}, s)$. Hence at least $n - t_r - t_s > 1$ honest parties hold this lock, and forwards it to all parties in $\Pi_{\mathrm{DC}}[s, k]$. Therefore every honest party eventually receives a valid $\mathsf{Lock}^{\mathsf{vc},s}_{n-t_r}$ and sends $(t_s + 1)$-threshold signature on $(\mathtt{fastvote}, s, k, vc)$. Eventually, every honest party receives t_s+1 valid sign signatures on $(\mathtt{fastvote}, s, k, vc)$, and outputs $\mathsf{Unique}^{\mathsf{vc},s}_{n-t_r} = \mathsf{Lock}^{\mathsf{vc},s}_{n-t_r}$.

Case 2: $T = n - t_s$**.** We consider asynchronous and synchronous networks.

Asynchronous network. Since $n > 2t_s + t_a$, at least $t_s + 1$ honest parties hold a valid $\mathsf{Lock}^{\mathsf{vc},s}_{n-t_s}$. By Lemma 5.6, no conflicting lock can exist. Hence each such honest party sends a vote for vc in the form of a $(n - t_s)$-threshold partial signature on $(\mathtt{Vote}, s, k, \mathsf{vc})$ and forward their locks. Eventually, $n - t_s$ valid shares are collected, forming a valid certificate satisfying condition (ii), and every honest party outputs $\mathsf{Unique}^{\mathsf{vc},s}_{n-t_s}$.

Synchronous network. By Lemma 5.2, every honest party outputs a valid $\mathsf{Store}^{\mathsf{vc},s}$, $\mathsf{Lock}^{\mathsf{vc},s}_{n-t_s}$ within 3Δ from receiving an input, and the sender outputs $\mathsf{Done}^s_{n-t_s}$ within 4Δ from receiving an input. Since P_s is t-slow PPD-honest, by Lemma 5.8, no conflicting locks can exist. Thus, all honest parties send votes for vc. Thus at least $n - t_s$ valid shares on $(\mathtt{Vote}, s, k, \mathsf{vc})$ are eventually collected as there are at most t_s, and every honest party outputs $\mathsf{Unique}^{\mathsf{vc},s}_{n-t_s}$. □

Lemma 5.15 (Unique implies Uniqueness in Asynchrony). *Assume the network is asynchronous. Fix a sender* P_s*. There do not exist two valid certificates* $\mathsf{Unique}_T^{\mathsf{vc},s}$ *and* $\mathsf{Unique}_{T'}^{\mathsf{vc}',s}$ *with* $\mathsf{vc} \neq \mathsf{vc}'$ *and* $T, T' \in \{n - t_s, n - t_r\}$.

Proof. A valid $\mathsf{Unique}_T^{\mathsf{vc},s}$ certificate consists of $t_s + 1$ signature shares on a vote message for $(\mathtt{fastvote}, s, k, \mathsf{vc})$ or $n - t_s$ signature shares for $(\mathtt{Vote}, s, k, \mathsf{vc})$, depending on T). An honest party sends $(\mathtt{fastvote}, s, k, \mathsf{vc})$ only after receiving a valid $\mathsf{Lock}^{\mathsf{vc},s}_{n-t_r}$ together with $\mathtt{Finish}_{n-t_r}$. It sends $(\mathtt{Vote}, s, k, \mathsf{vc})$ only after receiving a valid $\mathsf{Lock}^{\mathsf{vc},s}_{n-t_s}$ together with $\mathtt{Finish}_{n-t_s}$, and only if it has not received a valid lock-certificate for any conflicting value $\mathsf{vc}' \neq \mathsf{vc}$. Suppose for contradiction that two valid certificates $\mathsf{Unique}_T^{\mathsf{vc},s}$ and $\mathsf{Unique}_{T'}^{\mathsf{vc}',s}$ exist with $\mathsf{vc} \neq \mathsf{vc}'$. Each certificate consists of at least $n - t_s > t_s + 1$ signature shares, and since at most t_a parties are corrupted, at least one honest party contributes to the formation of the $\mathsf{Unique}_T^{\mathsf{vc},s}$ certificate. However, by Lemma 5.6, two conflicting valid lock-certificates cannot both exist in the asynchronous network. □

Lemma 5.16 (Unique implies Uniqueness in Synchrony). *Assume the network is synchronous. Fix a sender* P_s*. There do not exist two valid certificates* $\mathsf{Unique}_T^{\mathsf{vc},s}$ *and* $\mathsf{Unique}_{T'}^{\mathsf{vc}',s}$ *with* $\mathsf{vc} \neq \mathsf{vc}'$ *and* $T, T' \in \{n - t_s, n - t_r\}$.

Proof. Suppose for contradiction that there exist two valid certificates $\mathsf{Unique}_T^{\mathsf{vc},s}$ and $\mathsf{Unique}_{T'}^{\mathsf{vc}',s}$ with $\mathsf{vc} \neq \mathsf{vc}'$. By definition of ValidateUnique, a certificate $\mathsf{Unique}_T^{\mathsf{vc},s}$ is valid only if it is a $(t_s + 1)$-threshold signature on the message $(\mathtt{fastvote}, s, k, \mathsf{vc})$ when $T = n - t_r$, or $(n - t_s)$-threshold signature on the message $(\mathtt{Vote}, s, k, \mathsf{vc})$ when $T = n - t_s$. Thus, validity of each certificate implies that at least $n - t_s > t_s + 1 > 1$ honest party produced signature share on the corresponding message. We consider all possible combinations of T and T'.

Case 1: $T = T' = n - t_r$. In this case, both certificates are threshold signatures on $(\mathtt{fastvote}, s, k, \mathsf{vc})$ and $(\mathtt{fastvote}, s, k, \mathsf{vc}')$, respectively. An honest party sends $(\mathtt{fastvote}, s, k, \mathsf{vc})$ only after receiving a valid $(\mathsf{Lock}^{\mathsf{vc},s}_{n-t_r}, \mathtt{Finish}_{n-t_r})$ certificate. By Lemma 5.7, two conflicting lock-certificates $\mathsf{Lock}^{\mathsf{vc},s}_{n-t_r}$ and $\mathsf{Lock}^{\mathsf{vc}',s}_{n-t_r}$ cannot both be valid. Hence, at most one of vc and vc′ can trigger honest parties to send $\mathtt{fastvote}$ shares, contradicting the existence of two distinct $\mathsf{Unique}^{\cdot,s}_{n-t_r}$ certificates.

Case 2: $T = T' = n - t_s$. In this case, both certificates are threshold signatures on $(\texttt{Vote}, s, k, \text{vc})$ and $(\texttt{Vote}, s, k, \text{vc}')$. An honest party sends $(\texttt{Vote}, s, k, \text{vc})$ only after receiving a valid $(\texttt{Lock}^{\text{vc},s}_{n-t_s}, \texttt{Finish}_{n-t_s})$ and provided that within 2Δ it does not observe a conflicting lock-certificate for P_s. We argue that two conflicting certificates $\texttt{Unique}^{\text{vc},s}_{n-t_s}$ and $\texttt{Unique}^{\text{vc}',s}_{n-t_s}$ with $\text{vc} \neq \text{vc}'$ cannot both be valid. Suppose toward contradiction that both are valid. Then there exist sets Q, Q' of size at least $t_s + 1$ such that parties in Q signed $(\texttt{Vote}, s, k, \text{vc})$ and parties in Q' signed $(\texttt{Vote}, s, k, \text{vc}')$. Since at most t_s parties are corrupted, each of Q and Q' contains an honest party. Let P_i be the first honest party that sends $(\texttt{Vote}, s, k, \text{vc})$, and let τ_i be the time it first receives a valid certificate $(\texttt{Lock}^{\text{vc},s}_{n-t_s}, \texttt{Finish}_{n-t_s})$. By protocol, P_i sends its vote at time $\tau_i + 2\Delta$, provided it has not observed a conflicting lock-certificate during the interval $[\tau_i, \tau_i + 2\Delta]$. By synchrony, every honest party receives $\texttt{Lock}^{\text{vc},s}_{n-t_s}$ by time $\tau_i + \Delta$. Consider now any honest party P_j that contributes a signature share to $(\texttt{Vote}, s, k, \text{vc}')$. Let τ_j be the time P_j first receives $\texttt{Lock}^{\text{vc}',s}_{n-t_s}$. Since P_i is the first honest voter overall, we must have $\tau_j + 2\Delta \geq \tau_i + 2\Delta$, and hence $\tau_j \geq \tau_i$. But then P_j receives $\texttt{Lock}^{\text{vc},s}_{n-t_s}$ no later than $\tau_i + \Delta \leq \tau_j + \Delta$. Thus P_j observes the conflicting lock strictly before its own waiting period of 2Δ expires, and therefore it does not send a vote for vc'.

Case 3: $\{T, T'\} = \{n - t_s, n - t_r\}$. Wlog, assume $T = n - t_r$ and $T' = n - t_s$, and suppose toward contradiction that both $\texttt{Unique}^{\text{vc},s}_{n-t_r}$ and $\texttt{Unique}^{\text{vc}',s}_{n-t_s}$ are valid with $\text{vc} \neq \text{vc}'$. The validity of $\texttt{Unique}^{\text{vc},s}_{n-t_r}$ implies that at least $t_s + 1$ parties produced signature shares on $(\texttt{fastvote}, s, k, \text{vc})$, which can occur only after observing a valid $\texttt{Lock}^{\text{vc},s}_{n-t_r}$ certificate. By synchrony, this lock-certificate becomes known to all honest parties within Δ time. Once an honest party has observed $\texttt{Lock}^{\text{vc},s}_{n-t_r}$, it will not subsequently issue a vote for a conflicting value. Thus, the argument of Case 2 applies here as well: after $\texttt{Lock}^{\text{vc},s}_{n-t_r}$ becomes globally visible, honest parties cannot contribute $t_s + 1$ shares toward $(\texttt{Vote}, s, k, \text{vc}')$ for $\text{vc}' \neq \text{vc}$. Hence $\texttt{Unique}^{\text{vc}',s}_{n-t_s}$ cannot be formed. The symmetric argument applies if $\texttt{Unique}^{\text{vc}',s}_{n-t_s}$ forms first. Therefore, two such certificates cannot coexist.

□

Lemma 5.17 (Unique Implies Dispersal). *Let t denote the maximum corruption. Assume some party output $\texttt{Unique}^{\text{vc},s}_{T}$ in $\Pi_{\text{DC}}[s]$. Then, then $T - t$ forever honest parties P_i received valid store certificates $(\texttt{store}, s_i, \pi, vc)$.*

Proof. By definition of ValidateUnique, a certificate $\texttt{Unique}^{\text{vc},s}_{T}$ is valid only if it is a $(t_s + 1)$-threshold signature on the message $(\texttt{fastvote}, s, k, \text{vc})$ when $T = n - t_r$, or $(n - t_s)$-threshold signature on the message $(\texttt{Vote}, s, k, \text{vc})$ when $T = n - t_s$. An honest party sends such share if it sees $\texttt{Lock}^{\text{vc},s}_{n-t_r}$ for the former and $\texttt{Lock}^{\text{vc},s}_{n-t_s}$ for the latter. Thus, the claim follows immediately from Lemma 5.4. □

Lemma 5.18. *If all honest parties invoke $\text{RC}[s, k]$ with $\texttt{Reconstruct}^{vc,s,k}$ as input,then $\text{RC}[s, k]$ returns the same value to all honest parties. Moreover, if P_s (the sender) is honest, then $\text{RC}[s, k]$ returns an externally validated value.*

Proof. By definition of $\texttt{Reconstruct}^{vc,s,k}$ there exist T and vc such that the accompanying proof is $\pi = \texttt{Unique}^{\text{vc},s}_{T}$ and $\text{ValidateUnique}(s, k, \pi) = 1$. From Lemma 5.16 and 5.15, there is only one valid $\texttt{Unique}^{\text{vc},s}_{T}$ that could exist. By Lemma 5.17, it follows that at least $T - t$ forever honest parties received valid store certificates $(\texttt{store}, s_i, \pi_i, \text{vc})$. Once parties invoke $\Pi_{\text{RC}}[s, k]$ and multicast their store shares, every honest party receives minimum $n - 2t_s$ as there could be a most t_s under any network condition and T is at least $n - t_s$ by definition, which are enough to reconstruct. Finally, if P_s is honest, then the value dispersed in $\Pi_{\text{PD}}(\rho)[s]$ is externally validated by assumption, and the reconstruction follows from the earlier argument. Therefore, $\Pi_{\text{RC}}[s]$ returns an externally validated value. □

Lemma 5.19. *The triple* $(\Pi_{\text{PPD}}(n-2t_s), \Pi_{\text{DC}}, \Pi_{\text{RC}}(n-2t_s))$ *is a secure PPDB protocol according to Definition 2.2 when the validity predicates are as defined in Definition A.1. It satisfies* $(n-t_r-t_s, t_s)$*-Two-Step-Locking in synchrony and* $(n-t_s-t_a, t_a)$*-Two-Step-Locking in asynchrony.*

Proof.
- PPD-*Termination* It is easy to realize that the protocol outputs $\texttt{Finish}_{n-t_s}$ in $O(\max(\Delta, \delta))$ time to all parties, since it only consists of a constant number of sequential steps in which messages from $n-t_s$ parties are collected in the parallel Π_{PD} instances. The only reason one of the at least $n-t_s$ honest parties would not participate in a Π_{PD} instance, is if it has abandoned the protocol, which it only does for threshold t_s after seeing a forwardable certificate $\texttt{Ready}_{n-t_s}$, which also prompts it to send its contribution to the $\texttt{Finish}_{n-t_s}$ certificate. Since a party needs to wait for both some actual network delays δ and timeouts of order Δ, the time for the slow path is $O(\max(\Delta, \delta))$. The same reasoning applies to the fast path, except that it only goes through if less than t_r parties are corrupted, but in only $O(\delta)$ time.
- DC-*Termination* It is simple to verify that if some honest party has either of the locks for the sender P_s, then these propagate to all parties and pass the equivocation checks which only require participation from at most $n-t_s$ parties, forming reconstruct certificate. If some honest party holds a $\texttt{Lock}^{vc,s}_{n-t_r}$ and a $\texttt{Finish}_{n-t_r}$ certificate, then the check is completed in $O(\delta)$ time since these propagate at network speed (unless something else causes parties to terminate the protocol before this happens). On the other hand, if no honest parties have any locks, then it is similarly easy to verify that sufficiently many parties send a "no lock" message that it prompts a skip-certificate. If a $\texttt{Finish}_{n-t_r}$ is in the view of some honest party and less than t_r parties are corrupted, then a skip-certificate is formed in $O(\delta)$ time (unless some other certificate appears before it).
- RC-*Termination* Follows from Lemma 5.18 as Π_{RC} is the same as in BumbleBee.
- RC-*Agreement* Follows from Lemma 5.18.
- (h, c)*-Two-Step-Locking* We first argue about the c component, specifically we bound it to be the actual number of corruptions in the execution, f. If only a $\texttt{Finish}_T$ certificate exists, then the honest parties who signed it, did so after abandoning the T threshold path in all Π_{PD} instances. This implies for threshold $n-t_r$, that the at most t_r parties who did not sign it, cannot together with even up to t_s corrupted parties form a $\texttt{Lock}^{vc,s}_{n-t_r}$ on a vc that was not shared and signed by the adversary prior to $\texttt{Finish}_{n-t_r}$ existing. Similarly, for threshold $n-t_s$, the same holds if there are only t_a corruptions, as the at most t_s parties who did not sign a $\texttt{Finish}_{n-t_s}$ certificate cannot with only t_a parties form a new $\texttt{Lock}^{vc,s}_{n-t_s}$. If on the other hand, the network is synchronous and up to t_s parties could be corrupted, we are instead saved by the fact that honest parties are only signing $\texttt{Ready}_{n-t_s}$ after 5Δ, at which point all parties that are not yet corrupted have sent responded to each others original shared message. Thus, not a single honest signature would be available towards a conflicting lock. While in both cases, we only argued that no conflicting lock could exist for the same threshold T as the first $\texttt{Finish}_T$ certificate, this is sufficient because honest parties do not vote for a $\texttt{Lock}^{vc,s}_{n-t_s}$ in Π_{DC}, unless they have the corresponding $\texttt{Finish}_T$. Thus, the adversary is locked into its choice of at most f corrupted parties until a second finish certificate appears. This gives us the worst case guarantees that in asynchrony $c = t_a$ and in synchrony $c = t_s$.
  We then move on to the h component, where we have to show that there is a set H of at least h parties that cannot be skipped without a second Finish certificate appearing. We consider the synchronous and asynchronous cases separately.
  In asynchrony: Lemma 5.9 says that Fast PPD-Done parties are not skippable in asynchrony.

In other words, as soon as a valid $\texttt{Finish}_{n-t_r}$ exists in the view of an honest party, there are at least $n - t_r - t_a$ parties who can never be skipped in iterations that have not yet been invoked by an honest party. Thus, if a $\texttt{Finish}_{n-t_r}$ appears in the view of an honest party before a $\texttt{Finish}_{n-t_s}$ does, we get at least $n - t_r - t_a > n/4$ parties who were honest while dispersing their message can never be skipped, even if later a $\texttt{Finish}_{n-t_s}$ shows up. Lemma 5.10 says that Slow Done can only be skipped if a Fast Finish show up, i.e., that a valid $\text{Skip}^{s,k}$ certificate cannot exist in the view of an honest party before a $\texttt{Finish}_{n-t_r}$ does. This gives us at least $n - t_s - t_a > n/3$ honest parties who are committed until another $\texttt{Finish}$ certificate is in the view of an honest party. *In synchrony:* If a valid $\texttt{Finish}_{n-t_s}$ exists in the view of some honest party at time τ, then all honest parties were slow PPD-done before τ as shown in Lemma 5.3. This by Lemma 5.12 means that they cannot be skipped, unless a $\texttt{Finish}_{n-t_r}$ certificate appears in the view of an honest party. On the other hand, if a valid $\texttt{Finish}_{n-t_r}$ is the first to appear, then we have $n - t_s - t_r$ fast PPD-done and we rely on Lemma 5.11 to say that they can only be skipped if a $\texttt{Finish}_{n-t_s}$ appears. After certificates for both thresholds exist, there are similarly $n - t_s - t_r$ parties that are both fast and slow PPD-done. This is because out of any $n - t_r$ parties, $n - t_r - t_s$ are "forever honest", which means that they must be slow PPD-done when a $\texttt{Finish}_{n-t_s}$ appears (again using Lemma 5.3). Now using Lemma 5.13, we get that these parties are never skipped. In conclusion, in asynchrony we have $(n - t_s - t_a, t_a)$ while in synchrony we get $(n - t_s - t_r, t_s)$. □

Theorem 5.20. *[BumbleBee Protocol] Let $n > 2t_s + t_a$ and $n > t_s + 2t_r$. Let f denote the actual number of corruptions. Then, BumbleBee protocol (Fig. 6) is secure against strongly adaptive Byzantine adversaries. In particular, it is synchronously secure whenever $f \leq t_s$, asynchronously secure whenever $f \leq t_a$, and responsive whenever $f \leq t_r$. The expected communication complexity is $O(n^2\kappa + \frac{n^2}{n-2t_s}\ell)$, where ℓ is the input size and κ is the computational security parameter. In particular, when $n - 2t_s = \Theta(n)$, this becomes $O(n^2\kappa + n\ell)$. The expected running time is $O(\delta)$ whenever $f \leq t_r$, and $O(\Delta)$ otherwise.*

Proof. We instantiate the Π_{VBA} using [EKLN26], which provides expected communication $O(n^2\kappa)$, constant time $O(\Delta)$, and synchronous security under $f \leq t_s$, responsiveness in the form of expected $O(\delta)$ time under $f \leq t_r$, and asynchronous security under $f < t_a$. We instantiate the common coin using [CKS00] which is secure and responsive under $f < t_s$. The PPDB is instantiated with the triple $(\Pi_{\text{PPD}}(n - 2t_s), \Pi_{\text{DC+}}, \Pi_{\text{RC}}(n - 2t_s))$ from Lemma 5.19. Thus, the Π_{PPD} protocol produces a $\texttt{Finish}$ certificate within time $O(\Delta)$, and additionally produces a $\texttt{Finish}_{n-t_r}$ certificate within time $O(\delta)$ whenever $f \leq t_r$.

We first argue termination and complexity. In every iteration, a leader is chosen using the common coin, and this leader election is invoked by an honest party only after some $\texttt{Finish}_T$ certificate exists in its view. Because we have shown the PPDB protocol has worst case $(\min(n - t_s - t_a, n - t_r - t_s) > n/4, t_s)$-Two-Step-Locking across the different scenarios, we get that in each iteration, there is a constant probability that the chosen leader P_s is committed in the H set. Fix an iteration in which some honest party invokes the leader election. By definition of (h, c)-Two-Step-Locking, at that point there exist two sets H and C with $|H| \geq h$ and $|C| \leq c$ such that the following holds. If the elected leader belongs to H, then either its value can be successfully reconstructed and output (We will call this a "lucky iteration") or it can be skipped via $\text{Skip}^{s,k}$ certificate, which that the adversary could do only to a single honest leader. From Lemmas 5.10 to 5.12, we know that for P_s to be skipped by a $\text{Skip}^{s,k}$ certificate, a new $\texttt{Finish}_{T'}$ for $T' \neq T$ must appear. More precisely, using the notation of Definition 2.2, if we let l_τ denote the number of thresholds for which a $\texttt{Finish}_T$

certificate exists in the view of an honest party at time τ, then electing a leader in H either leads to successful reconstruction or causes l_τ to increase. Since $l_\tau \in 1, 2$ and is monotone, this increase can happen at most once over the entire execution. Note, that (h, c)-Two-Step-Locking property is proven in Lemma 5.19. Thus, that the protocol will terminate in the second lucky iteration. Since the leader is sampled uniformly at random, each iteration is good with probability at least h/n. Since $h = \Omega(n)$, the expected number of iterations until two good iterations occur is constant. Each iteration requires expected communication $O(n^2\kappa + \frac{n^2}{n-2t_s}\ell)$, since the Π_{VBA} contributes $O(n^2\kappa)$ and reconstruction of an ℓ-bit value from $n - 2t_s$ coded fragments costs $O(\frac{n^2}{n-2t_s}\ell)$. Moreover, each iteration takes time $O(\Delta)$ in general and $O(\delta)$ whenever $f \leq t_r$. This yields the claimed expected communication complexity and running time.

It remains to prove External Validity, Agreement, and Quality.

External Validity. A party outputs a value only after reconstructing it and checking that it satisfies the predicate Pred. Hence every output value satisfies Pred.

Agreement. In every iteration, all honest parties invoke the same binary Π_{VBA} instance, and thus all honest parties obtain the same binary output for that iteration. If the binary output is 0, all honest parties move to the next iteration. If the binary output is 1, they reconstruct the value corresponding to the elected leader. By agreement of the reconstruction protocol Π_{RC}, all honest parties that reconstruct obtain the same value. Therefore, no two honest parties can output different values.

Quality. As argued above, the protocol terminates in the second good iteration. Therefore, for the protocol to terminate with an output value corresponding to a good leader, it suffices that the first two iterations both elect leaders from H. Since each iteration elects a leader from H with probability at least $\frac{h}{h+c}$, it follows the protocol has quality at least $(\frac{h}{h+c})^2$. In synchrony, the PPDB satisfies $(h, c) = (n - t_s - t_r,\ t_s)$-Two-Step-Locking. Hence the protocol achieves quality at least $\left(\frac{n-t_s-t_r}{n-t_r}\right)^2$. Using the assumption $n - t_s - t_r > n/4$, we obtain $\left(\frac{n-t_s-t_r}{n-t_r}\right)^2 \geq \left(\frac{n/4}{n}\right)^2 = \frac{1}{16}$. In asynchrony, the PPDB satisfies $(h, c) = (n - t_s - t_a,\ t_a)$-Two-Step-Locking. Hence the protocol achieves quality at least $\left(\frac{n-t_s-t_a}{n}\right)^2$. Using the assumption $n - t_s - t_a > n/3$, we obtain $\left(\frac{n-t_s-t_a}{n-t_s}\right)^2 \geq \left(\frac{n/3}{n}\right)^2 = \frac{1}{9}$. □

6 BUMBLEBEE+ SECURITY ANALYSIS

Lemma 6.1 (Full Honest Never Skipped). *Assume the network is synchronous, $k \leq \lambda$, and P_s is an τ'-slow honest and τ-fast honest, and every honest party invokes $\Pi_{\mathrm{DC+}}[s][k]$ at time $\tau'' > \max(\tau, \tau')$, then (1) every honest party either outputs a valid $\mathtt{Unique}^{\mathrm{vc},s,k}_{n-t_r}$ or a valid $\mathsf{Redisperse}^{\mathrm{vc},s,k}$, and (2) $\mathsf{TimeoutCert}^{s,k}$ does not exist in any party's view.*

Proof. Observe that every honest party invokes $\Pi_{\mathrm{DC+}}[s, k]$ after time τ''. By the construction of $\Pi_{\mathrm{DC}} + [s, k]$, each party first invokes $\Pi_{\mathrm{DC}}[s, k]$ as the first step. Therefore, every honest party obtains either a $\mathsf{Skip}^{s,k}$ certificate or a $\mathtt{Unique}^{\mathrm{vc},s,k}_{n-t_r}$ certificate from the output of $\Pi_{\mathrm{DC}}[s, k]$. Since every honest party invokes $\Pi_{\mathrm{DC+}}[s, k]$ after time τ'' and P_s is τ'-slow honest such that $\tau' < \tau''$, Lemma 5.13 implies that no valid $\mathsf{Skip}^{s,k}$ can exist in this execution. Hence each honest party outputs either a valid $\mathtt{Unique}^{\mathrm{vc},s,k}_{n-t_r}$ or a valid $\mathtt{Unique}^{\mathrm{vc},s,k}_{n-t_s}$ for some commitment vc. If some honest party outputs $\mathtt{Unique}^{\mathrm{vc},s,k}_{n-t_r}$, then this certificate is forwarded, and by synchrony every honest party eventually receives and outputs a valid $\mathtt{Unique}^{\mathrm{vc},s,k}_{n-t_r}$, proving (1). Otherwise, all honest parties output $\mathtt{Unique}^{\mathrm{vc},s,k}_{n-t_s}$. Since P_s is honest and τ-slow and all honest parties invoke $\Pi_{\mathrm{DC+}}[s, k]$ after τ, Lemma 5.2 implies that each honest party has already received a valid $(\mathtt{store}, s_j, \pi_j, \mathrm{vc})$ for the

same vc. Thus each honest party multicasts its valid $(\texttt{store}, s_i, \pi_i, \text{vc})$. By synchrony, within Δ every honest party receives at least $n - t_s$ such shares, reconstructs the same message m satisfying $\text{vc} = \text{VC.VCom}(\text{Enc}(m, \rho))$, and multicasts $(\texttt{Redispersed}, s, k, \text{vc})$. Again by synchrony, within another Δ every honest party receives at least $t_s + 1$ such messages and constructs $\text{Redisperse}^{\text{vc},s,k}$, proving (1). Finally, in this execution every honest party completes reconstruction and redisperse within a bounded synchronous window (at most 2Δ after receiving $\texttt{Unique}^{\text{vc},}_{n-t_s}s$ as honest parties receive $\texttt{Unique}^{\text{vc},s}_{n-t_s}$ at most Δ apart). Therefore no honest party sends a timeout message. Since at most t_s parties are corrupted and $\text{TimeoutCert}^{s,k}$ requires $t_s + 1$ signature shares, such a certificate cannot exist, proving (2). □

Lemma 6.2 (Redisperse Implies Reconstruction). *If a valid* $\text{Redisperse}^{\text{vc},s,k}$ *exists in an execution of* $\Pi_{\text{DC+}}[s,k]$ *for some* vc*, then every honest party eventually holds a valid* $(\texttt{store}, s_j, \pi_j, \text{vc})$ *for the same* vc.

Proof. A valid $\text{Redisperse}^{\text{vc},s,k}$ is a t_s+1 threshold signature on the message $(\texttt{Redispersed}, s, k, \text{vc})$. Since at most t_s parties are corrupted, at least one honest party P_i must have produced a valid signature share on $(\texttt{Redispersed}, s, k, \text{vc})$. By the protocol, an honest party P_i sends such a share only after receiving sufficiently many valid shares of the form $(\texttt{store}, s_j, \pi_j, \text{vc})$ that verify with respect to the same vc appearing in a valid $\texttt{Unique}^{\text{vc},s,k}_{n-t_s}$, and after reconstructing a candidate message m from these shares such that $\text{vc} = \text{VC.VCom}(\text{Enc}(m, \rho))$. Upon successful reconstruction, P_i re-encodes m as $\text{Enc}(m, \rho)$ and multicasts $(\texttt{store}, s_j, \pi_j, \text{vc})$ to every party P_j before issuing its signature share on $(\texttt{Redispersed}, s, k, \text{vc})$. □

Lemma 6.3 (No Slow Unique under Responsive Condition). *Assume that at most* $f \le t_r$ *parties are corrupted, and assume some party* P_s *is* τ*-fast honest. Assume a valid* $\texttt{Finish}_{n-t_r}$ *certificate exists in the view of some party, and every party invokes* $\Pi_{\text{DC+}}[s,k]$ *at time* $\tau' > \tau$*. Then, no* $\texttt{Unique}^{\text{vc},s,k}_{n-t_s}$ *can be output from* $\Pi_{\text{DC}}[s,k]$ *by any party.*

Proof. We assume that there are at most t_r malicious parties. Since P_s is τ-fast PPD-Honest, there are at most t_r honest parties who have not contributed to its $\texttt{Done}^{s}_{n-t_s}$ certificate. Since all the honest parties who did contribute to the certificate already had its high threshold lock: $\texttt{Lock}^{\text{vc},s}_{n-t_r}$, there are at most t_r honest parties without this lock at time τ. By assumption, every party invokes $\Pi_{\text{DC+}}[s,k]$ at time $\tau' > \tau$. According to the protocol code, a party that has $\texttt{Lock}^{\text{vc},s}_{n-t_r}$ does not send a signature share on $(\texttt{Vote}, s, k, vc)$, which is needed to construct $\texttt{Unique}^{\text{vc},s,k}_{n-t_s}$. Thus, at most t_r honest parties may send $(\texttt{Vote}, s, k, vc)$. A valid $\texttt{Unique}^{\text{vc},s,k}_{n-t_s}$ is $n - t_s$ threshold signature on $(\texttt{Vote}, s, k, vc)$. Since there are at most t_r corrupted parties, the combined number of parties who may send such a signature share is $2t_r < n - t_s$. Thus, no party can create a valid $\texttt{Unique}^{\text{vc},s,k}_{n-t_s}$ certificate.

□

Lemma 6.4 (Only Fast Unique under Responsive Condition). *Assume that at most* $f \le t_r$ *parties are corrupted and some party* P_s *is* τ*-fast PPD honest. Assume every honest party starts* $\Pi_{\text{DC+}}[s,k]$ *after time* τ *where* $\texttt{Finish}_{n-t_r}$ *exists in the view of some party. Then,* $\texttt{Unique}^{\text{vc},s,k}_{n-t_r}$ *is the only valid possible certificate than any party outputs.*

Proof. The only possible outputs of $\Pi_{\text{DC+}}[s,k]$ are $\texttt{Unique}^{\text{vc},s,k}_{n-t_r}$, $\text{Redisperse}^{\text{vc},s,k}$, $\text{Skip}^{s,k}$, or $\text{TimeoutCert}^{s,k}$. In the first step of $\Pi_{\text{DC+}}[s,k]$, every party invokes $\Pi_{\text{DC}}[s,k]$. Since P_s is τ-fast honest and all honest parties invoke $\Pi_{\text{DC+}}[s,k]$ after time τ, Lemma 5.9 implies that no valid skip-certificate $\text{Skip}^{s,k}$ can be produced in $\Pi_{\text{DC}}[s,k]$, hence no party can output $\text{Skip}^{s,k}$. Similarly, Lemma 6.3 implies that no valid $\texttt{Unique}^{\text{vc},s,k}_{n-t_s}$ can be produced in $\Pi_{\text{DC}}[s,k]$, hence no party can

output it. It is left to argue $\mathsf{TimeoutCert}^{s,k}$. For timeout, recall that $\mathsf{TimeoutCert}^{s,k}$ is a $t_s + 1$ threshold signature on a timeout message. In $\Pi_{\mathsf{DC+}}[s,k]$, an honest party issues a timeout-share for (s,k) only if after obtaining a valid $\mathtt{Unique}^{\mathsf{vc},s,k}_{n-t_s}$ and does not receive the $(\mathtt{store},\cdot,\cdot,\mathsf{vc})$ message. From Lemma 6.3, no valid $\mathtt{Unique}^{\mathsf{vc},s,k}_{n-t_s}$ can be produced in $\Pi_{\mathsf{DC}}[s,k]$. Therefore no honest party sends a timeout signature share, and since at most t_s parties are corrupted, no adversary can collect $t_s + 1$ timeout shares to form $\mathsf{TimeoutCert}^{s,k}$. Hence $\mathsf{TimeoutCert}^{s,k}$ cannot exist. □

Lemma 6.5. *The triple* $(\Pi_{\mathsf{PPD}}(n-t_s-t_r), \Pi_{\mathsf{DC+}}, \Pi_{\mathsf{RC}}(n-t_s-t_r))$ *is a secure PPDB protocol according to Definition 2.2 when the validity predicates are as defined in Definition A.1. It satisfies* $(n-t_r-t_s, t_s)$*-Two-Step-Locking in synchrony while in asynchrony, it satisfies* $(n-t_s-t_a, t_a)$*-Weak-Two-Step-Locking in the first* λ *iterations, then* $(n-t_s-t_a, t_a)$*-Two-Step-Locking in the remaining iterations.*

Proof.

- PPD-*Termination* Follows from the proof of Lemma 5.19. The only difference between the protocols is the number of symbols needed to reconstruct the message, which only affects the liveness of the reconstruction in the Π_{RC} protocol.
- DC-*Termination* The $\Pi_{\mathsf{DC+}}$ protocol first invokes Π_{DC} which we showed DC-termination in Lemma 5.19; each honest party outputs some valid $\mathsf{Skip}^{s,k}$ certificate (noLock-certificate) or $\mathtt{Reconstruct}^{s,k}$ (unique-certificate) certificate. In the first λ iterations, $\Pi_{\mathsf{DC+}}$ will not directly output a $\mathtt{Unique}^{vc,s,k}_{n-t_s}$ certificate, but instead attempt to reconstruct the message and establish a certificate ($\mathsf{Redisperse}^{vc,s,k}$) that this succeeded or failed; alternatively receive a skip-certificate ($\mathsf{NoLock}^{s,k}$) from Π_{DC}. So, if a single honest party has a skip-certificate that is an output from Π_{DC}, it terminates $\Pi_{\mathsf{DC+}}$. Otherwise, all honest parties have a $\mathtt{Unique}^{vc,s,k}_{n-t_s}$ output and will attempt to reconstruct the message. Either all of them reach the timeout without having reconstructed and send a (t_s+1)-threshold partial signature on $\mathtt{Timeout}$, establishing a $\mathsf{TimeoutCert}^{s,k}$ for all parties as there are at least $n - t_s > t_s$ honest parties. Alternatively, one of them succeeds, and reconstructs a message which passes the consistency check. It will then send each party P_j their share of the message with a proof that belongs to the j^{th} position in the vc in $\mathtt{Unique}^{vc,s,k}_{n-t_s}$. This prompts each honest party P_j to multicast their authenticated share, thus allowing all honest parties to reconstruct a consistent message. Following this, either all of the at least $t_s + 1$ honest parties establish a $\mathsf{Redisperse}^{vc,s,k}$ using a $t_s + 1$ threshold signature, or at least one of them has in the meantime received a different certificate that allows giving output. In the latter case, that party forwarded the certificate to all honest parties.
- RC-*Termination* Follows from Lemma 5.18, and Lemma 6.2.
- RC-*Agreement* Follows from Lemma 5.18.
- (h,c)*-(Weak-)Two-Step-Locking.* We first argue that the bound on the c parameter from Lemma 5.19 still holds; that is in asynchrony $c = t_a$ and in synchrony $c = t_s$. The proof there only relies on the inability of the adversary to create a valid $\mathtt{Reconstruct}^{vc,s,k}$ certificate (unique certificate) for a sender outside the bounded corruption set. In $\Pi_{\mathsf{DC+}}$, the only deviation from Π_{DC} is that a $\mathtt{Unique}^{vc,s,k}_{n-t_s}$ certificate is no longer by itself considered a valid $\mathtt{Reconstruct}^{vc,s,k}$ certificate; instead, reconstruction must additionally pass the reconstruction check. This strictly strengthens the conditions for reconstruction. Hence, any reconstruction attack that was impossible in Lemma 5.19 remains impossible here, and therefore the same bound on c continues to hold. We now argue the h parameter. We split into two cases: $k \leq \lambda$ and $k > \lambda$: Fix some iteration $k \leq \lambda$. We consider the asynchronous and synchronous cases separately.

Asynchronous case. If only a $\texttt{Finish}_{n-t_s}$ certificate exists in the view of honest parties, then the weak definition imposes no non-skippability guarantee. Indeed, the adversary may always force the execution into the reconstruction-check phase and delay reconstruction for up to 2Δ, and cause all honest parties to output $\mathsf{TimeoutCert}^{s,k}$. Thus, $h = 0$ here.
Suppose instead that a $\texttt{Finish}_{n-t_r}$ certificate exists. Then, by Lemma 6.4, at least $n - t_r - t_a$ forever honest parties are Fast PPD-honest and therefore cannot be skipped in any later $\mathrm{DC}[s,k]$ instance that was not yet invoked by an honest party. This remains true even if a $\texttt{Finish}_{n-t_s}$ certificate later appears. Accordingly, the same $h = n - t_s - t_a$ guarantee as in the asynchronous analysis of Lemma 5.19 continues to hold.
Synchronous case. If a $\texttt{Finish}_{n-t_r}$ certificate exists and $f \leq t_r$, then Lemma 6.4 again implies that the execution progresses without timeouts and that the corresponding set of Fast PPD-honest $n - t_s - t_r$ parties cannot be skipped. Otherwise, we fall back to the same argument as in the synchronous analysis of Lemma 5.19. The only difference is that, after obtaining a $\texttt{Unique}^{vc,s,k}_{n-t_s}$ certificate, parties may enter the reconstruction. However, in synchrony for every Slow PPD-honest sender P_s, reconstruction succeeds within the allotted 2Δ time. Thus, the same set $|H| = n - t_s - t_r$ constructed in Lemma 5.19 remains non-skippable here as well. For iterations $k > \lambda$, the same reasoning as in Lemma 5.19 applies directly as for iterations $k > \lambda$, as the protocol falls back to BumbleBee construction. Note, by construction, every iteration $k \leq \lambda$ terminates with constant probability whenever the network is synchronous. Consequently, reaching iteration $\lambda + 1$ implies, except with negligible probability, that the execution is asynchronous. If the network is asynchronous, then reconstruction is guaranteed. If a $\texttt{Finish}_{n-t_s}$ certificate exists in the view of honest parties, then at least $n-t_s-t_a$ honest parties have consistent shares. Since the reconstruction threshold is $n-t_s-t_r$, and $n-t_s-t_a \geq n-t_s-t_r$, honest parties can always gather sufficiently many valid shares to reconstruct the value. Note, if $\texttt{Finish}_{n-t_r}$ exists instead than we have larger number of honest parties with shares as $n - t_r - t_a > n - t_s - t_a$.

□

Theorem 6.6. *[BumbleBee+ Protocol] Let $n > 2t_s + t_a$ and $n > t_s + 2t_r$. Let f denote the actual number of corruptions. Then, BumbleBee+ protocol (Fig. 8) is secure against strongly adaptive Byzantine adversaries. In particular, it is synchronously secure whenever $f \leq t_s$, asynchronously secure whenever $f \leq t_a$, and responsive whenever $f \leq t_r$. The expected communication complexity is $O(n^2\kappa + n\ell)$ in synchrony and $O(\lambda(n^2\kappa + n\ell))$ in asynchrony, where λ is the statistical security parameter. The expected running time is $O(\delta)$ whenever $f \leq t_r$, and $O(\Delta)$ otherwise.*

Proof. We instantiate the PPDB using the triple $(\Pi_{\mathrm{PPD}}(n - 2t_s), \Pi_{\mathrm{DC+}}, \Pi_{\mathrm{RC}}(n - 2t_s))$, which by Lemma 6.5 satisfies (h, c)-Weak-Two-Step-Locking. In particular, it guarantees $(n - t_s - t_r, t_s)$-Two-Step-Locking in synchrony and $(n - t_r - t_a, t_a)$-Weak-Two-Step-Locking whenever a $\texttt{Finish}_{n-t_r}$ certificate exists in the view of an honest party and $(0, t_a)$-Weak-Two-Step-Locking. Moreover, Π_{PPD} produces a $\texttt{Finish}_{n-t_r}$ certificate within time $O(\delta)$ whenever $f \leq t_r$.

We first argue termination and complexity. As in Theorem 5.20, in every iteration a leader is sampled uniformly at random using the common coin, and leader election is invoked only after some $\texttt{Finish}_T$ certificate exists in the view of an honest party. We first consider the synchronous case. By Lemma 6.5, the PPDB satisfies (h, c)-Two-Step Locking with $h = n - t_s - t_r > n/4$. Thus, in every iteration there is probability at least $h/n > 1/4$ that the elected leader belongs to the corresponding committed set H. As in the proof of Theorem 5.20, electing a leader in H either leads to successful reconstruction and termination, or causes the number of visible $\texttt{Finish}$ thresholds

to increase. Since there are only two thresholds, this increase can occur at most once during the execution. Consequently, the protocol terminates in the second good iteration.

Therefore, after the first possible skipped good iteration, every subsequent iteration terminates independently with probability at least $1/4$. Hence, the probability that the protocol has not terminated after λ iterations is at most $4^{1-\lambda}$, which is negligible for an appropriate choice of λ. Thus, except with negligible probability, the protocol terminates during the first λ iterations whenever the network is synchronous.

We now consider the asynchronous case. During the first λ iterations, Weak-Two-Step-Locking only provides guarantees if a $\texttt{Finish}_{n-t_r}$ certificate appears in the view of an honest party. In that case, by Lemma 6.4, at least $n - t_r - t_a$ honest parties cannot be skipped, and the same constant-probability progress argument from Theorem 5.20 applies.

If the protocol reaches iteration $\lambda + 1$, then except with negligible probability the execution is asynchronous. From this point onward, the protocol falls back to the original BumbleBee analysis, and therefore regains constant probability of termination in every iteration. The only additional observation needed is that reconstruction succeeds for every value corresponding to a $\texttt{Unique}^{vc,s,k}_{n-t_s}$ certificate. Indeed, in asynchrony there are at most t_a corruptions, so at least $n - t_s - t_a$ honest parties hold consistent shares for the value. Since reconstruction requires only $n - t_s - t_r$ shares, and $n - t_s - t_a \geq n - t_s - t_r$, honest parties can always gather sufficiently many valid shares to reconstruct.

It remains to argue running time. As in Theorem 5.20, every iteration takes time $O(\Delta)$ in general. Moreover, whenever $f \leq t_r$, the protocol obtains a $\texttt{Finish}_{n-t_r}$ certificate within time $O(\delta)$. After this point, Lemma 6.4 guarantees that a constant fraction of honest parties, namely $n - t_r - t_a$, cannot be skipped. Thus, the same responsiveness argument as in Theorem 5.20 shows that the expected running time is $O(\delta)$ whenever $f \leq t_r$.

For communication complexity, the synchronous case follows exactly as in Theorem 5.20, yielding expected communication $O(n^2\kappa + n\ell)$ since the expected number of iterations is constant. In contrast, in asynchrony the protocol may require up to $\lambda = \Theta(\lambda)$ iterations before switching to the asynchronous fallback behavior. Since each iteration costs $O(n^2\kappa + n\ell)$ communication, the overall expected communication complexity in asynchrony is $O(\lambda(n^2\kappa + n\ell))$.

It remains to prove External Validity, Agreement, and Quality.

External Validity. A party outputs a value only after reconstructing it and checking that it satisfies the predicate Pred. Hence every output value satisfies Pred.

Agreement. The agreement proof is identical to that of Theorem 5.20. All honest parties invoke the same binary agreement instance in every iteration and therefore obtain the same binary output. Whenever reconstruction occurs, agreement of Π_{RC} guarantees that all honest parties reconstruct the same value. Thus, no two honest parties can output different values.

Quality. In synchrony, the proof follows exactly as in Theorem 5.20, since the PPDB satisfies $(n - t_s - t_r, t_s)$-Two-Step-Locking. Hence the protocol achieves quality at least $\left(\frac{n-t_s-t_r}{n-t_r}\right)^2$, which is at least $1/16$ since $n - t_s - t_r > n/4$.

In contrast, in asynchrony the protocol only satisfies Weak-Two-Step-Locking. The adversary may therefore simulate an execution in which reconstruction checks fail throughout the first λ iterations whenever the elected leader is honest. It can continue this strategy until the common coin selects a corrupted leader, at which point reconstruction succeeds for the adversarial value. Thus, except with negligible probability, the adversary can force the protocol to terminate with an input originating from a corrupted party. Accordingly, the protocol provides only negligible quality in the asynchronous setting. □

REFERENCES

[ABKL22] Andreea B. Alexandru, Erica Blum, Jonathan Katz, and Julian Loss. State machine replication under changing network conditions. In Shweta Agrawal and Dongdai Lin, editors, *ASIACRYPT 2022, Part I*, volume 13791 of *LNCS*, pages 681–710, Taipei, Taiwan, December 5–9, 2022. Springer, Cham, Switzerland.

[ACC22] Ananya Appan, Anirudh Chandramouli, and Ashish Choudhury. Perfectly-secure synchronous MPC with asynchronous fallback guarantees. In Alessia Milani and Philipp Woelfel, editors, *41st ACM PODC*, pages 92–102, Salerno, Italy, July 25–29, 2022. ACM.

[AMN+19] Ittai Abraham, Dahlia Malkhi, Kartik Nayak, Ling Ren, and Maofan Yin. Sync HotStuff: Simple and practical synchronous state machine replication. Cryptology ePrint Archive, Paper 2019/270, 2019.

[AMS19] Ittai Abraham, Dahlia Malkhi, and Alexander Spiegelman. Asymptotically optimal validated asynchronous byzantine agreement. In *Proc. of the 2019 ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC)*, pages 337–346, 2019.

[BCLZL23] Renas Bacho, Daniel Collins, Chen-Da Liu-Zhang, and Julian Loss. Network-agnostic security comes (almost) for free in DKG and MPC. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part I*, volume 14081 of *LNCS*, pages 71–106, Santa Barbara, CA, USA, August 20–24, 2023. Springer, Cham, Switzerland.

[BKL19] Erica Blum, Jonathan Katz, and Julian Loss. Synchronous consensus with optimal asynchronous fallback guarantees. In Dennis Hofheinz and Alon Rosen, editors, *TCC 2019, Part I*, volume 11891 of *LNCS*, pages 131–150, Nuremberg, Germany, December 1–5, 2019. Springer, Cham, Switzerland.

[BKL21] Erica Blum, Jonathan Katz, and Julian Loss. Tardigrade: An atomic broadcast protocol for arbitrary network conditions. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part II*, volume 13091 of *LNCS*, pages 547–572, Singapore, December 6–10, 2021. Springer, Cham, Switzerland.

[Buc16] Ethan Buchman. Tendermint: Byzantine fault tolerance in the age of blockchains. Technical report, Master's Thesis, University of Guelph, 2016.

[BZL20] Erica Blum, Chen-Da Liu Zhang, and Julian Loss. Always have a backup plan: Fully secure synchronous MPC with asynchronous fallback. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part II*, volume 12171 of *LNCS*, pages 707–731, Santa Barbara, CA, USA, August 17–21, 2020. Springer, Cham, Switzerland.

[CF13] Dario Catalano and Dario Fiore. Vector commitments and their applications. In *International Conference on Theory and Practice of Public Key Cryptography*, 2013.

[CKPS01] Christian Cachin, Klaus Kursawe, Frank Petzold, and Victor Shoup. Secure and efficient asynchronous broadcast protocols. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 524–541, Santa Barbara, CA, USA, August 19–23, 2001. Springer Berlin Heidelberg, Germany.

[CKS00] Christian Cachin, Klaus Kursawe, and Victor Shoup. Random oracles in constantipole: practical asynchronous byzantine agreement using cryptography (extended abstract). In Gil Neiger, editor, *19th ACM PODC*, pages 123–132, Portland, OR, USA, July 16–19, 2000. ACM.

[CL99] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance. In *Proceedings of the 3rd Symposium on Operating Systems Design and Implementation (OSDI)*, pages 173–186, 1999.

[CR93] Ran Canetti and Tal Rabin. Fast asynchronous byzantine agreement with optimal resilience. In *25th ACM STOC*, pages 42–51, San Diego, CA, USA, May 16–18, 1993. ACM Press.

[EKLN26] Fatima Elsheimy, Simon Holmgaard Kamp, Julian Loss, and Jesper Buus Nielsen. Optimal best-of-both-worlds consensus. Cryptology ePrint Archive, Paper 2026/239, 2026.

[GLL+22] Yingzi Gao, Yuan Lu, Zhenliang Lu, Qiang Tang, Jing Xu, and Zhenfeng Zhang. Dumbo-NG: Fast asynchronous BFT consensus with throughput-oblivious latency. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022*, pages 1187–1201, Los Angeles, CA, USA, November 7–11, 2022. ACM Press.

[GLT+20] Bingyong Guo, Zhenliang Lu, Qiang Tang, Jing Xu, and Zhenfeng Zhang. Dumbo: Faster asynchronous BFT protocols. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020*, pages 803–818, Virtual Event, USA, November 9–13, 2020. ACM Press.

[GLZW22] Diana Ghinea, Chen-Da Liu-Zhang, and Roger Wattenhofer. Optimal synchronous approximate agreement with asynchronous fallback. In Alessia Milani and Philipp Woelfel, editors, *41st ACM PODC*, pages 70–80, Salerno, Italy, July 25–29, 2022. ACM.

[KZG10] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 177–194, Singapore, December 5–9, 2010. Springer Berlin Heidelberg, Germany.

[LJY14] Benoît Libert, Marc Joye, and Moti Yung. Born and raised distributively: fully distributed non-interactive adaptively-secure threshold signatures with short shares. In *Proceedings of the 2014 ACM Symposium on Principles of Distributed Computing*, PODC '14, page 303–312, New York, NY, USA, 2014. Association for Computing Machinery.

[LLTW20] Yuan Lu, Zhenliang Lu, Qiang Tang, and Guiling Wang. Dumbo-MVBA: Optimal multi-valued validated asynchronous byzantine agreement, revisited. In Yuval Emek and Christian Cachin, editors, *39th ACM PODC*, pages 129–138, Virtual Event, Italy, August 3–7, 2020. ACM.

[PS17] Rafael Pass and Elaine Shi. Hybrid consensus: Efficient consensus in the permissionless model. In *DISC*, volume 91 of *LIPIcs*, pages 39:1–39:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

[PS18] Rafael Pass and Elaine Shi. Thunderella: Blockchains with optimistic instant confirmation. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 3–33, Tel Aviv, Israel, April 29 – May 3, 2018. Springer, Cham, Switzerland.

[RS60] Irving S. Reed and Gustave Solomon. Polynomial codes over certain finite fields. *Journal of the Society for Industrial and Applied Mathematics*, 8(2):300–304, 1960.

[SAN25] Nibesh Shrestha, Ittai Abraham, and Kartik Nayak. Communication and round efficient parallel broadcast protocols. In *FC*, volume 15751 of *Lecture Notes in Computer Science*, pages 327–344. Springer, 2025.

[Sho00] Victor Shoup. Practical threshold signatures. In *Proceedings of the 19th International Conference on Theory and Application of Cryptographic Techniques*, EUROCRYPT'00, page 207–220, Berlin, Heidelberg, 2000. Springer-Verlag.

[YMR+18] Maofan Yin, Dahlia Malkhi, Michael K. Reiter, Guy Golan Gueta, and Ittai Abraham. Hotstuff: Bft consensus in the lens of blockchain, 2018.

A ADDITIONAL DEFINITIONS

Definition A.1 (Validation Predicates). We list the precise predicates defining the validity of certificates used across our two PPDB protocols (PPDB and PPDB+) and corresponding MVBA protocol (BumbleBeeand BumbleBee+). Both protocols employ the same validity checks for certificates created in the dispersal phase PPD and for the uniqueness certificates used in the dispersal check DC.[1]

- $\mathsf{ValidateLock}(T, \mathsf{Lock}_T^{\mathsf{vc},s})$: Outputs 1 iff $\mathsf{TS.Verify}_T(\mathsf{PK}, (\mathtt{Stored}_T, \mathsf{vc}, s), \mathsf{Lock}_T^{\mathsf{vc},s}) = \text{accept}$.
- $\mathsf{ValidateDone}(T, \mathtt{Done}_T^s)$: Outputs 1 iff $\mathsf{TS.Verify}_T(\mathsf{PK}, (\mathtt{locked}_T, s), \mathtt{Done}_T^s) = \text{accept}$.
- $\mathsf{ValidateReady}(T, \mathtt{Ready}_T)$: Outputs 1 iff $\mathsf{TS.Verify}_{t_s+1}(\mathsf{PK}, (\mathtt{Ready}_T) = \text{accept}$.
- $\mathsf{ValidateFinish}(T, \mathtt{Finish}_T)$: Outputs 1 iff $\mathsf{TS.Verify}_T(\mathsf{PK}, (\mathtt{Finish}_T) = \text{accept}$.
- $\mathsf{ValidateUnique}(s, k, \mathtt{Unique}_T^{\mathsf{vc},s,k})$: Outputs 1 iff either
  (i) $T = n - t_r$ and $\mathsf{TS.Verify}_{t_s+1}(\mathsf{PK}, (\mathtt{fastvote}, s, k, vc), \mathtt{Unique}_T^{\mathsf{vc},s,k}) = \text{accept}$, , or
  (ii) $T = n - t_s$ and $\mathsf{TS.Verify}_{n-t_s}(\mathsf{PK}, (\mathtt{Vote}, s, k, \mathsf{vc}), \mathtt{Unique}_T^{\mathsf{vc},s,k}) = \text{accept}$.
- $\mathsf{ValidateNoLock}(s, k, \mathsf{NoLock}^{s,k})$: Outputs 1 iff $\mathsf{TS.Verify}_{n-t_s}(\mathsf{PK}, (\mathtt{Vote}, s, k, \perp), \mathsf{NoLock}^{s,k}) = \text{accept}$.

They differ in what constitutes a skip and a reconstruct certificate. In the vanilla BumbleBee protocol these are defined as follows:

- $\mathsf{ValidateSkip}(s, k, \pi)$: Outputs 1 iff $\mathsf{ValidateNoLock}(s, k, \pi)$ outputs 1.
- $\mathsf{ValidateReconstruct}(v, \pi_v)$: Outputs 1 iff $\mathsf{ValidateUnique}(s, k, \pi_v)$ outputs 1.

In BumbleBee+, the predicates are as above for $k > \lambda$, while for $k \leq \lambda$ they are defined as:

- $\mathsf{ValidateSkip}(s, k, \pi)$: Outputs 1 iff $\mathsf{TS.Verify}_{n-t_s}(\mathsf{PK}, (\mathtt{Vote}, s, k, \perp), \pi) = \text{accept}$ or $\mathsf{TS.Verify}_{t_s+1}(\mathsf{PK}, (\mathtt{Timeout}, s, k), \pi) = \text{accept}$
- $\mathsf{ValidateReconstruct}(s, k, \pi)$: Outputs 1 iff $\mathsf{TS.Verify}_{t_s+1}(\mathsf{PK}, (\mathtt{fastvote}, s, k, vc), \pi) = \text{accept}$ or $\mathsf{TS.Verify}_{t_s+1}(\mathsf{PK}, (\mathtt{Redispersed}, s, k, \mathsf{vc}), \pi) = \text{accept}$

Finally, the validity predicates used to parameterize the k^{th} VBA instance is, for both protocols, the one which is satisfied by a skip-certificate when the bit is 0 and a reconstruction certificate when the bit is 1.

[1]While the same signature checks are performed, they still achieve different guarantees as parties are prompted to sign those messages by different events in the two protocols.

- $\mathsf{ValidateIn}_k(v, \pi_v)$: Outputs 1 iff either
  (i) $v = 1$ and there exist T, vc such that $\mathsf{ValidateReconstruct}(s, k, \pi_v) = 1$, or
  (ii) $v = 0$ and $\pi_v = \mathsf{Skip}^{s,k}$, where $\mathsf{ValidateSkip}(s, k, \mathsf{Skip}^{s,k}) = 1$.