Jolteon and Ditto: Network-Adaptive Efficient Consensus with Asynchronous Fallback

Rati Gelashvili[1], Lefteris Kokoris-Kogias[1,2], Alberto Sonnino[1], Alexander Spiegelman[1], and Zhuolun Xiang[1,3]

[1] Novi Research
{gelash, asonnino, sashaspiegelman}@fb.com
[2] IST Austria
ekokoris@ist.ac.at
[3] University of Illinois at Urbana-Champaign
xiangzl@illinois.edu

Abstract. Existing committee-based Byzantine state machine replication (SMR) protocols, typically deployed in production blockchains, face a clear trade-off: (1) they either achieve linear communication cost in the steady state, but sacrifice liveness during periods of asynchrony, or (2) they are robust (progress with probability one) but pay quadratic communication cost. We believe this trade-off is unwarranted since existing linear protocols still have asymptotic quadratic cost in the worst case. We design `Ditto`, a Byzantine SMR protocol that enjoys the best of both worlds: optimal communication on and off the steady state (linear and quadratic, respectively) and progress guarantee under asynchrony and DDoS attacks. We achieve this by replacing the view-synchronization of partially synchronous protocols with an asynchronous fallback mechanism at no extra asymptotic cost. Specifically, we start from HotStuff, a state-of-the-art linear protocol, and gradually build `Ditto`. As a separate contribution and an intermediate step, we design a 2-chain version of HotStuff, `Jolteon`, which leverages a quadratic view-change mechanism to reduce the latency of the standard 3-chain HotStuff. We implement and experimentally evaluate all our systems to prove that breaking the robustness-efficiency trade-off is in the realm of practicality.

1 Introduction

The popularity of blockchain protocols generated a surge in researching how to increase the efficiency and robustness of consensus protocols used for agreement. On the efficiency front, the focus has been on decreasing the communication complexity in the steady state, first to quasilinear [18] and ultimately to linear [31, 14]. These protocols work in the eventually synchronous model and require a leader to aggregate proofs. However, handling leader failures or unexpected network delays requires quadratic communication and if the network is asynchronous, there is no liveness guarantee. On the robustness side, recent protocols [4, 24, 29] make progress by having each replica act as the leader and decide on a leader retroactively. This requires quadratic communication even under good network conditions when the adversary is strongly adaptive [2].

	Comm. Complexity	Rounds	Liveness
HotStuff [31]	sync $O(n)$	7	not live if async
VABA [4]	$O(n^2)$	E(16.5)	always live
`Jolteon`	sync $O(n)$	5	not live if async
`Ditto`	sync $O(n)$, async $O(n^2)$	sync 5, async E(10.5)	always live

Table 1: Theoretical comparison of our protocol implementations. For HotStuff and our protocols, sync $O(n)$ assumes honest leaders. Communication complexity measures the cost per committed block. Rounds measure the block-commit latency. $E(r)$ means r rounds in expectation.

We believe that in practice, we need the best of both worlds. An efficient steady state is beneficial for any production system, but for blockchains to support important (e.g. financial) infrastructure, robustness against asynchrony is also key. First, unpredictable network delays are a common condition when running in a geo-distributed network environment, e.g. over the Internet. Second, the possibility of targeted DDoS attacks on the leaders of leader-based protocols motivates the leaderless nature of the asynchronous solutions. Thus, we ask:

Are there efficient blockchain systems that have a linear steady state and are robust against asynchrony?

This is an important question, posed as early as [20], and studied from a theoretical prospective [26, 28], but the existing blockchain systems still forfeit robustness for efficiency [31, 14, 18]. In this paper, we answer the above question with the first practical system, `Ditto`, tailor-made to directly apply to the prominent HotStuff/DiemBFT [31, 30] family of protocols.

Contribution (i): `Jolteon`. As an intermediate step to build `Ditto`, we first present `Jolteon`, a protocol that's a hybrid of HotStuff [31] and classical PBFT [8]. `Jolteon` abandons the linear view-change [31] of HotStuff/DiemBFT, since the protocol anyway has a quadratic pacemaker [30]. In particular, `Jolteon` preserves the linearity of HotStuff under good network conditions while reducing the steady state block-commit latency by 30% using a 2-chain commit rule. This decrease in latency comes at the cost of a quadratic view-change. As the pacemaker is already quadratic, as expected, this does not affect performance in our experiments.

Contribution (ii): `Ditto`. We design `Ditto`, which combines the optimistic (good network conditions) linear steady state with pessimistic (worst-case network conditions) liveness guarantees with no extra asymptotic communication cost. `Ditto` is based on the key observation that when there is asynchrony or failures, the protocols with linear steady state still pay the quadratic cost, same as state-of-the-art asynchronous protocols (e.g., VABA [4] or Dumbo [24]) that provide significantly more robustness. Specifically, `Ditto` replaces the pacemaker of `Jolteon`/HotStuff/DiemBFT (a quadratic module that deals with view synchronization) with an asynchronous fallback. In other words, instead of synchronizing views that will anyway fail due to asynchrony or faults, we fall back to an asynchronous protocol that guarantees progress. Furthermore, `Ditto` switches be-

tween the steady state and the fallback *without overhead* (e.g. additional rounds), and maintains *pipelined/chained operations* similar to HotStuff.

Contribution (iii): Implementation and evaluation. Since `Jolteon` outperforms the original HotStuff in every scenario, we use it as the basis for `Ditto` design and implementation. As shown experimentally under good network conditions both `Jolteon` and `Ditto` outperform HotStuff. Importantly, `Ditto`'s performance is almost identical to `Jolteon` in this good case whereas during attack `Ditto` performs almost identically to our optimized implementation of VABA [4]. In addation, the throughput of `Ditto` is 50% better than VABA in the steady state and much better than HotStuff and `Jolteon` under faulty (dead) leaders (30-50% better) or network instability (they drop to 0). Finally, after discussions with the Diem Engineering team, they deployed `Jolteon` and are currently considering the use of `Ditto` if attacks or instability of the network happen too often.

2 Preliminaries

We consider a permissioned system that consists of an adversary and n replicas numbered $1, 2, \ldots, n$, where each replica has a public key certified by a public-key infrastructure (PKI). The replicas have all-to-all reliable and authenticated communication channels controlled by the adversary. We say a replica multicasts a message if it sends the message to all replicas. We consider an adversary that can adaptively corrupt up to f replicas, referred as *Byzantine*. The rest of the replicas are called *honest*. The adversary controls the message delivery times, but we assume messages among honest replicas are eventually delivered.

An execution of a protocol is *synchronous* if all message delays between honest replicas are bounded by Δ; is *asynchronous* if they are unbounded; and is *partially synchronous* if there is a global stabilization time (GST) after which they are bounded by Δ [11]. Without loss of generality, we let $n = 3f + 1$ where f denotes the assumed upper bound on the number of Byzantine faults, which is the optimal worst-case resilience bound for asynchrony, partial synchrony [11], or asynchronous protocols with fast synchronous path [5].

Cryptographic primitives and assumptions. We assume standard digital signature and public-key infrastructure (PKI), and use $\langle m \rangle_i$ to denote a message m signed by replica i. For simplicity, we assume every message in the protocol is signed by its sender. We also assume an adaptively secure threshold signature scheme such as [21], where a set of signature shares for message m from t (the threshold) distinct replicas can be combined into one threshold signature of the same length for m. We use $\{m\}_i$ to denote a threshold signature share of a message m signed by replica i. We also assume a collision-resistant cryptographic hash function $H(\cdot)$ that can map an input of arbitrary size to an output of fixed size. Any deterministic agreement protocol cannot tolerate even a single fault under asynchrony due to FLP [12]. Our asynchronous fallback protocol generates distributed randomness using the adaptively secure common coin of [22]: the

generated randomness of a view is the hash of the unique threshold signature (of threshold $f + 1$) on the view number.

For simplicity of presentation, we assume the above cryptographic schemes are ideal and a trusted dealer equips replicas with these cryptographic materials. The dealer assumption can be lifted using any asynchronous distributed key generation protocol such as [19, 3, 9, 10].

BFT SMR. A Byzantine fault-tolerant state machine replication protocol [1] commits client transactions as a log akin to a single non-faulty server, and provides the following two guarantees:

- Safety. Honest replicas commit same transactions at the same log position.
- Liveness. Each transaction is eventually committed by all honest replicas.

We assume that each client transaction will be repeatedly proposed by honest replicas until it is committed [4]. In other words, any transaction will not be censored by Byzantine leaders forever. For most of the paper, we omit the client from the discussion and focus on replicas. SMR protocols usually implement many instances of single-shot Byzantine agreement, but there are various approaches for ordering. We focus on the chaining approach, used in HotStuff [31] and DiemBFT [30], in which each proposal references the previous one and each commit commits the entire prefix of the chain.

Terminology. We present some terminologies used throughout the paper.

- *Round Number and View Number.* The protocol proceeds in rounds and views and each replica keeps track of the current round number and view number, both initialized as 0. Each view can have several rounds and it is incremented by 1 after each asynchronous fallback. Each round r has a designated leader L_r that proposes a new block (defined below) of transactions in round r.
- *Block Format.* A block is formatted as $B = [id, qc, r, v, txn]$ where qc is the quorum certificate (defined below) of B's parent block in the chain, r is the round number of B, v is the view number of B, txn is the digest of a batch of new transactions, and $id = H(qc, r, v, txn)$ is the unique hash digest of qc, r, v, txn. We will use $B.x$ to denote the element x of B.
- *Quorum Certificate.* A *quorum certificate (QC)* of some block B is a threshold signature of a message that includes $B.id, B.r, B.v$, produced by combining the signature shares $\{B.id, B.r, B.v\}$ from a quorum of $n-f = 2f+1$ replicas. We say a block is *certified* if there exists a QC for the block. Blocks are chained by QCs to form a blockchain, or block-tree if there are forks. The round and view numbers of QC for block B are denoted by $QC.r$ and $QC.v$, which equals $B.r$ and $B.v$, respectively. A QC or a block of view number v and round number r has rank $rank = (v, r)$, and QCs or blocks are compared lexicographically by their rank (i.e. first by the view number, then by the round number). We use qc_{high} to denote the highest ranked quorum certificate.
- *Timeout Certificate.* A timeout message of round r by a replica contains the replica's threshold signature share on r, and its qc_{high}. A timeout certificate

[4] For example, clients can send their transactions to all replicas, and the leader can propose transactions that are not yet included in the blockchain, in the order that they are submitted. With rotating leaders of HotStuff/DiemBFT and random leader election of the asynchronous fallback, the assumption can be guaranteed.

Let L_r be the leader of round r. Each replica keeps the highest voted round r_{vote}, the highest locked round r_{lock} [a], current round number r_{cur}, and the highest quorum certificate qc_{high} (the current view number v_{cur} is not used and remains 0 throughout). Replicas initialize $r_{vote} = 0$, $r_{lock} = 0$, $r_{cur} = 1$, qc_{high} as the QC of the genesis block of round 0, and enter round 1.

Steady State Protocol for Replica i

- **Propose.** Upon entering round r, the leader L_r multicasts a signed block $B = [id, qc_{high}, r, v_{cur}, txn]$.
- **Vote.** Upon receiving the first valid block $B = [id, qc, r, v, txn]$ from L_r in round r, execute *Advance Round*, *Lock*, and then *Commit* (defined below). If $r = r_{cur}$, $v = v_{cur}$, $r > r_{vote}$ and $qc.r \geq r_{lock}$, vote for B by sending the threshold signature share $\{id, r, v\}_i$ to L_{r+1}, and update $r_{vote} \leftarrow r$.
- **Lock.** (2-chain lock rule) Upon observing any valid qc [b], let qc' be the QC contained in the block certified by qc (i.e., qc' is the parent of qc), the replica updates $r_{lock} \leftarrow \max(r_{lock}, qc'.r)$, and $qc_{high} \leftarrow \max(qc_{high}, qc)$.
- **Commit.** (3-chain commit rule) If there exist three adjacent certified blocks B, B', B'' in the chain with consecutive round numbers, i.e., $B''.r = B'.r+1 = B.r + 2$, the replica commits B and all its ancestors.

Pacemaker Protocol for Replica i

- **Advance Round.** The replica updates current round $r_{cur} \leftarrow \max(r_{cur}, r)$ iff
  - the replica receives or forms a round-$(r-1)$ quorum certificate qc, or
  - the replica receives or forms a round-$(r-1)$ timeout certificate tc.
- **Timer and Timeout.**
  - Upon entering round r, the replica sends the round-$(r-1)$ TC to L_r if it has the TC, and resets its timer to count down for a predefined time interval (timeout τ).
  - When the timer expires, the replica stops voting for round r_{cur} and multicasts a signed `timeout` message $\langle \{r_{cur}\}_i, qc_{high} \rangle$ where $\{r_{cur}\}_i$ is a threshold signature share.
  - Upon receiving a valid `timeout` message or TC, execute *Advance Round*, *Lock*, and then *Commit*.
  - Upon receiving 2f+1 timeouts, form a TC.

[a] Corresponds to the *preferred round* in [30].

[b] May be formed from votes (by a leader) or contained in a proposal or a timeout messsage.

Fig. 1: DiemBFT in our terminology.

(TC) is formed by a quorum of $n - f = 2f + 1$ timeout messages, containing a threshold signature on a round number r produced by combining $2f + 1$ signature shares $\{r\}$ from the `timeout` messages, and $2f + 1$ qc_{high}'s. A valid TC should only contain qc_{high} with round numbers $< TC.r$, and this will be checked implicitly when a replica receives a TC.

We say a message (block, QC or TC) is valid, if it follows the definition and is properly signed.

Performance metrics. We consider *communication complexity* per committed block. For the theoretical analysis, we consider the standard latency metric called

block-commit latency, i.e., the number of rounds for all honest replicas to commit a block since it is proposed by an honest leader (under synchrony and honest leaders). For the empirical analysis, we measure the *end-to-end latency*, i.e., the time to commit a transaction since it is sent by a client.

Description of DiemBFT. The DiemBFT protocol (or LibraBFT) [30] is a production version of HotStuff [31] with a synchronizer implementation (Pacemaker). We describe the full protocol of DiemBFT in Figure 1, and give a brief description below.

There are two components of DiemBFT, a *Steady State protocol* that makes progress when the round leader is honest, and a *Pacemaker protocol* that advances round numbers either due to the lack of progress or due to the current round being completed.

– *Propose.* The leader L_r, upon entering round r, proposes a block B that extends a block certified by the highest QC it knows about.

When receiving the first valid round-r block from L_r, any replica tries to advance its current round number, and update its highest QC. It also checks for commit, updates its locked round and votes for the block according to the following rules.

– *3-chain commit.* A block can be committed if it is the first block among 3 adjacent certified blocks with consecutive round numbers.
– *2-chain lock.* For any two adjacent certified blocks observed, update the locked round number to be the highest of the first block's round number.
– *Voting.* Replica votes for a block if the block has round and view number same as the replica, and round number higher than last voted block's round, and contains a QC of round no less than replica's locked round number. The replica votes for B by sending a threshold signature share to the next leader.

Then, when the next leader L_{r+1} receives $2f + 1$ such votes, it forms a QC of round r, enters round $r + 1$, proposes the block for that round, and the above process is repeated.

– *Quadratic view-synchronization.* When the timer of some round r expires before entering round $r + 1$, the replica stops voting for round r and multicast a timeout containing a threshold signature share for r and its highest QC.

When any replica receives $2f + 1$ such timeout messages, it forms a TC of round r, enters round $r + 1$ and sends the TC to the (next) leader L_{r+1}. When any replica receives a timeout or a TC, it tries to advance its current round number given the high-QCs (in the timeout or TC) or the TC, updates its highest locked round and its highest QC given the high-QCs, and checks if any block can be committed using the same rules above.

3 `Jolteon` Design

In this section, we describe how we turn DiemBFT into `Jolteon` – a 2-chain version of DiemBFT (commit via 2-chain rule). We present the full protocol of `Jolteon` in Figure 2, and highlight the intuition and major changes compared to DiemBFT below. As mentioned previously, the quadratic cost of view-synchronization in leader-based consensus protocols, due to faulty leaders or

Replicas keep the same variables as DiemBFT in Figure 1.
Steady State Protocol for Replica i
Changes from DiemBFT in Figure 1 are marked in blue.
- **Propose.** Upon entering round r, the leader L_r multicasts a signed block $B = [id, qc_{high}, tc, r, v_{cur}, txn]$, where $tc = tc_{r-1}$ if L_r enters round r by receiving a round-$(r-1)$ tc_{r-1}, and $tc = \perp$ otherwise.
- **Vote.** Upon receiving the first valid block $B = [id, qc, tc, r, v, txn]$ from L_r, execute *Advance Round*, *Lock*, and then *Commit* (defined below). If $r = r_{cur}$, $v = v_{cur}$, $r > r_{vote}$ and ((1) $r = qc.r + 1$, or (2) $r = tc.r + 1$ and $qc.r \geq \max\{qc_{high}.r \mid qc_{high} \in tc\}$), vote for B by sending the threshold signature share $\{id, r, v\}_i$ to L_{r+1}, and update $r_{vote} \leftarrow r$.
- **Lock.** (1-chain lock rule) Upon seeing a valid qc (formed by votes or contained in proposal or timeouts), the replica updates $qc_{high} \leftarrow \max(qc_{high}, qc)$.
- **Commit.** (2-chain commit rule) If there exists two adjacent certified blocks B, B' in the chain with consecutive round numbers, i.e., $B'.r = B.r + 1$, the replica commits B and all its ancestors.

Pacemaker Protocol for Replica i
Identical to DiemBFT in Figure 1.

Fig. 2: `Jolteon`.

asynchronous periods, is inherent. While the linearity of HotStuff's view-change is a theoretical milestone, its practical importance is limited by this anyway quadratic cost of synchronization after bad views.

With this insight in mind, `Jolteon` uses a quadratic view-change protocol that allows a linear 2-chain commit rule in the steady state (see the **Commit** rule in Figure 2). The idea is inspired by PBFT [8] with each leader proving the safety of its proposal. In the steady state each block extends the block from the previous round and providing the QC of the parent is enough to prove safety, hence the steady state protocol remains linear. However, after a bad round caused by asynchrony or a bad leader, proving the safety of extending an older QC requires the leader to prove that nothing more recent than the block of that QC is committed. To prove this, the leader uses the TC formed for view-changing the bad round. Recall that a TC for round r contains $2f+1$ replicas' qc_{high} sent in timeout messages for round r. The leader attaches the TC to its proposal in round $r+1$ and extends the highest QC among the QCs in the TC.

When a replica gets a proposal B, it first tries to advance its round number, then updates its qc_{high} with $B.qc$ and checks the 2-chain commit rule for a possible commit. Then, before voting, it verifies that at least one of the following two conditions is satisfied:

- $B.r = B.qc.r + 1$ or;
- $B.r = B.tc.r + 1$ and
  $B.qc.r \geq \max\{qc_{high}.r \mid qc_{high} \in B.tc\}$

In other words, either B contains the QC for the block of the previous round; or it contains at least the highest QC among the $2f + 1$ QCs in the attached TC, which was formed to view-change the previous round.

	Latency	Steady state	View-change	View-synchronization
DiemBFT	7 messages	linear	linear	quadratic
`Jolteon`	5 messages	linear	quadratic	quadratic

Table 2: Theoretical comparison between DiemBFT and `Jolteon`.

Safety intuition. If the first condition is satisfied then B directly extends the block from the previous round. Since at most one QC can be formed in a round, this means that no forks are possible, and voting for B is safe.

The second condition is more subtle. Note that by the 2-chain commit rule, if a block B' is committed, then there exists a certified block B'' s.t. $B'.round+1 = B''.round$. That is, at least $f+1$ honest replicas vote to form the QC for B'' and thus set their qc_{high} to be the QC for block B' ($qc_{B'}$). By quorum intersection and since replicas never decrease their qc_{high}, any future (higher round) TC contains a qc_{high} that is at least as high as $qc_{B'}$. The second condition then guarantees that honest replicas only vote for proposals that extend the committed block B'. Due to lack of space, the full proof is given in Appendix A.

Efficiency. Table 2 compares the efficiency of DiemBFT and `Jolteon` from a theoretical point of view. Both protocols have linear communication complexity per round and per decision in steady state (under synchrony and honest leaders), due to the leader-to-all communication pattern and the threshold signature scheme[5]. The complexity of the Pacemaker to synchronize views (view-synchronization in Table 2), for both protocols, under asynchrony or failures is quadratic due to the all-to-all timeout messages. The complexity of proposing a block after a bad round that requires synchronization (view-change communication in Table 2) is linear for DiemBFT and quadratic for `Jolteon`. This is because in DiemBFT such a proposal only includes qc_{high}, whereas in `Jolteon` it includes a TC containing $2f+1$ qc_{high}. The block-commit latency *under synchrony and honest leaders* is 7Δ and 5Δ for DiemBFT and `Jolteon`, respectively, due to the 3-chain (2-chain) commit. Each 1-chain requires two rounds (leader proposing and replicas voting), plus the new leader multicast the last QC of the chain that allows replicas to learn and commit the new block.

Limitations. During periods of asynchrony, or when facing DDoS attacks on the leaders, both protocols have *no liveness guarantees* – the leaders' blocks cannot be received on time. As a result, replicas keep multicasting timeout messages and advancing round numbers without certifying or committing any blocks. This is fundamentally unavoidable [28]: communication complexity of any deterministic partially synchronous Byzantine agreement protocol is unbounded before GST.

Fortunately, in the next section, we show that it is possible to boost the liveness guarantee of DiemBFT and `Jolteon`, by replacing the view-synchronization mechanism with a fallback protocol that guarantees progress even under asynchrony. Furthermore, the asynchronous fallback can be efficient. The protocol we propose in the next section has quadratic communication cost for fallback, which is the cost DiemBFT and `Jolteon` pay to synchronize views.

[5] The implementation of DiemBFT does not use threshold signatures, but for the theoretical comparison here we consider a version of DiemBFT that does.

4 Ditto Design

To strengthen the liveness guarantees of existing partially synchronous BFT protocols such as DiemBFT [30] and Jolteon, we propose an Asynchronous Fallback protocol. It has quadratic communication complexity (same as Jolteon view-change and the Pacemaker of DiemBFT) and always makes progress even under asynchrony or DDoS attacks on the leader. We call the composition of Jolteon with Asynchronous Fallback Ditto. Ditto has linear communication cost for the synchronous path, quadratic cost for the asynchronous path, and preserves liveness robustly in asynchronous network conditions. The steady state protocol (sync. path) is presented in Figure 4, and the asynchronous fallback protocol (async. path) is presented in Figure 5. The proofs for Ditto can be found in Appendix B.

Protocol intuition. Our solution consists of a steady state protocol, which is similar to that of Jolteon, and an asynchronous fallback protocol, which replaces the view-change of Jolteon. An illustration of our protocol is shown in Figure 3. The idea behind our fallback protocol is that, after entering the fallback, all replicas will act as leaders to build their fallback chains. Once enough fallback chains grow to a certain height, a random leader election occurs to select one fallback chain, allowing the replicas to return to steady state and continue with the chosen chain. It can be shown that with constant probability, a fallback chain with enough blocks is selected, such that at least one new block on this fallback chain is committed by all replicas.

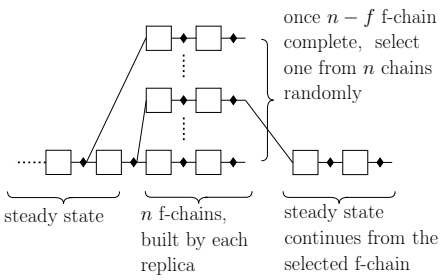


Fig. 3: The protocol intuition.

Since this protocol has two paths, a synchronous fast path and an asynchronous fallback path, it is critical to ensure safety and liveness when the protocol switches from one path to another. On a high level, our protocol ensures safety by always following commit and voting rules from Jolteon. While in the fallback path, the fallback chain selected by leader election is very similar to a steady state chain, hence we let all replicas update their local states with respect to the fallback chain when exiting the fallback, as if Jolteon had made progress. As for liveness, our protocol guarantees that either the sync path (same as Jolteon) makes progress, or enough replicas timeout the synchronous path and enter the fallback, and the fallback always finishes.

Additional terminology for Ditto.

– *Fallback-block and Fallback-chain.* For the fallback, we define another type of block named *fallback-block (f-block)*, denoted as $\overline{B}$. In contrast, the steady state block is called the *regular block*. An f-block $\overline{B}$ adds two additional fields to a regular block B, formatted as $\overline{B} = [B, height, proposer]$ where $height \in \{1, 2\}$ is the position of the f-block in the fallback-chain and *proposer* is the

replica that proposes the block. We will use $\overline{B}_{h,i}$ to denote a height-h f-block proposed by replica i. A fallback-chain (f-chain) consists of f-blocks.

- *Fallback-QC.* A fallback quorum certificate (f-QC) $\overline{qc}$ for an f-block $\overline{B}_{h,i}$ is a threshold signature for the message $(\overline{B}.id, \overline{B}.r, \overline{B}.v, h, i)$, produced by combining the signature shares $\{\overline{B}.id, \overline{B}.r, \overline{B}.v, h, i\}$ from a quorum of replicas ($n - f = 2f + 1$ replicas). An f-block is certified if there exists an f-QC for the f-block. f-QCs or f-blocks are first ranked by view numbers and then by round numbers. In contrast, the QC of regular blocks is called *regular QC*.
- *Fallback-TC.* A fallback timeout certificate (f-TC) $\overline{tc}$ is a threshold signature for a view number v, produced by combining the signature shares $\{v\}$ from a quorum of replicas ($n - f = 2f + 1$ replicas). f-TCs are ranked by views.
- *Leader Election and Coin-QC.* We use the adaptively secure common coin [22] for leader election. For any given view, each replica signs the view number with threshold signature as the coin share. Then any $f + 1$ valid coin shares of the same view from distinct replicas can form a unique threshold signature (called coin-QC or qc_{coin}) on the view number. The hash of the unique threshold signature above is used as randomness to elect leader L among n replicas with probability $1/n$. The probability of the adversary to predict the outcome of the election is at most $1/n + negl(k)$.
- *Endorsed Fallback-QC and Endorsed Fallback-block.* Once a replica has a qc_{coin} of view v that elects replica L as the fallback-chain leader, we say any f-QC of view v by replica L is endorsed (by qc_{coin}), and the f-block certified by the f-QC is also endorsed (by qc_{coin}). Any endorsed f-QC is handled as a regular QC in any steps of the protocol such as *Lock, Commit, Advance Round*. An endorsed f-QC ranks higher than any regular QC with the same view number. As cryptographic evidence of endorsement, the first block in a new view includes the coin-QC of the previous view.

Description of Steady State. The steady state protocol is given in Figure 4. Compared to `Jolteon`, the leader no longer provides TC in the block as the proof of view-synchronization. Instead, it attaches the coin-QC formed by the fallback of the previous view, which proves the fallback already finishes and replicas should enter the new view. Each replica additionally keeps a boolean value `async` to record if it is in the fallback, during which the replica will not vote regular blocks. The 1-chain lock rule and 2-chain commit rule still apply, but the two blocks in the 2-chain can be certified regular block or endorsed fallback-block from the same view.

Description of Asynchronous Fallback. Now we give a brief description of the Asynchronous Fallback protocol (Figure 5), which replaces the Pacemaker protocol in the `Jolteon` protocol (Figure 2). Just like in `Jolteon`, when the timer expires, the replica tries to initiate the fallback (the equivalent of view-change) by broadcasting a timeout message containing the highest QC and a signature share of the current view number. When receiving or forming a fallback-TC from $2f+1$ timeout messages, the replica enters the fallback path: It updates its current view number v_{cur}, initializes the voted round number $\bar{r}_{vote}[j] = 0$ and the voted height number $\bar{h}_{vote}[j] = 0$ for each replica j. Finally, the replica starts

Let L_r be the leader of round r. Each replica keeps the highest voted round r_{vote}, current round number r_{cur}, the highest quorum certificate qc_{high}, and a boolean value `async`. Replicas initialize $r_{vote} = 0$, $r_{cur} = 1$, qc_{high} as the QC of the genesis block of round 0, `async` $= false$ and enter round 1.

Steady State Protocol for Replica i

Changes from `Jolteon` in Figure 2 are marked in blue.

- **Propose.** Upon entering round r, the leader L_r multicasts a block $B = [id, qc_{high}, qc_{coin}, r, v_{cur}, txn]$, where qc_{coin} is the coin-QC of view v_{cur-1} if B is the first proposal in view v_{cur}, otherwise $qc_{coin} = \perp$.
- **Vote.** Upon receiving the first valid proposal $B = [id, qc, qc_{coin}, r, v, txn]$ from L_r, execute *Exit Fallback if* $qc_{coin} \neq \perp$; otherwise execute *Advance Round*, *Lock*, and then *Commit*. If $r = r_{cur}$, $v = v_{cur}$, $r > r_{vote}$, $r = qc.r+1$, $qc.rank \geq qc_{high}.rank$, and `async` $= false$, vote for B by sending the threshold signature share $\{id, r, v\}_i$ to L_{r+1}, and update $r_{vote} \leftarrow r$.
- **Lock.** (1-chain lock rule) Upon seeing a valid qc (formed by votes or contained in proposal or timeouts), the replica updates $qc_{high} \leftarrow \max(qc_{high}, qc)$.
- **Commit.** (2-chain commit rule) If there exists two adjacent blocks B, B' with the *same view number* in the chain, each can be a certified block or an endorsed fallback-block, the replica commits B and all its ancestors.

Round Synchronization for Replica i

- **Advance Round.** Upon receiving a valid qc, the replica updates its current round $r_{cur} \leftarrow \max(r_{cur}, qc.r + 1)$.
- **Timer and Timeout.** Upon entering a new round or a new view, the replica resets its timer to τ. When the timer expires, the replica updates `async` $\leftarrow$ *true*, and multicasts a `timeout` message $\langle \{v_{cur}\}_i, qc_{high} \rangle_i$ where $\{v_{cur}\}_i$ is a threshold signature share. Upon receiving a valid `timeout` message, execute *Advance Round*, *Lock*, and then *Commit*.

Fig. 4: Steady State of `Ditto`

building its fallback-chain by broadcasting the f-TC and proposing the first f-block which extends the qc_{high}, has height 1, round number $qc_{high}.r + 1$, and view number v_{cur}. Any f-block (irrelevant of height) gets verified by all replicas who vote on it (updating their voted round and height number for the fallback) by sending signature shares back to the proposer of the f-block. Replicas build their fallback-chain *not by necessarily extending their own chain but by adopting the first certified block of matching height they received* (in v_{cur}). This boosting strategy guarantees that no honest replica's chain is left behind in the middle of the fallback, hence at least $2f + 1$ chains will reach height-2 and their leader-replica will broadcast a height-2 f-QC.

Finally, when the replica receives $2f + 1$ height-2 f-QCs, it knows that $2f + 1$ f-chains are complete and starts the leader election by releasing a coin share for the current view number. When $f + 1$ shares are released the leader of the view is determined through the formation of a coin-QC qc_{coin}. The fallback is then terminated, and the replica updates `async` $= false$ to exit the fallback and enters the next view, acting as if the chain of the elected leader is the only known chain. Looking at this chain the replica updates all relevant variables and

During a fallback of view v, replicas record all the f-QCs of view v by any replica j, and keeps a voted round number $\bar{r}_{vote}[j]$ and a voted height number $\bar{h}_{vote}[j]$.

Async. Fallback Protocol for Replica i

- **Enter Fallback.** Upon receiving or forming an f-TC $\overline{tc}$ of view $v \geq v_{cur}$, update `async` $\leftarrow true$, $v_{cur} \leftarrow v$, $\bar{r}_{vote}[j] \leftarrow 0, \bar{h}_{vote}[j] \leftarrow 0$ for $\forall j \in [n]$, and multicast $\overline{tc}$ and a height-1 f-block $\overline{B}_{1,i} = [id, qc_{high}, qc_{high}.r + 1, v_{cur}, txn, 1, i]$.
- **Fallback Vote.** Upon receiving an f-block $\overline{B}_{h,j}$ from replica j, if $h > \bar{h}_{vote}[j]$, `async` $= true$, and
  - if $h = 1$ and $\overline{B}_{h,j} = [id, qc, r, v, txn, 1, j]$ such that $qc.rank \geq qc_{high}.rank$, $r = qc.r + 1$, and $v = v_{cur}$; or
  - if $h = 2$, and $\overline{B}_{h,j} = [id, \overline{qc}, r, v, txn, 2, j]$ such that $\overline{qc}$ is a valid f-QC, $v = v_{cur}$, $r = \overline{qc}.r + 1$, $r > \bar{r}_{vote}[j]$ and $\overline{qc}.height = 1$,

  set $\bar{r}_{vote}[j] \leftarrow r, \bar{h}_{vote}[j] \leftarrow h$ and send $\{id, r, v, h, j\}_i$ back to replica j.
- **Fallback Propose.** Upon the first height-h f-block $\overline{B}_{h,j}$ (*proposed by any replica* j) is certified by some f-QC $\overline{qc}$ and `async` $= true$,
  - if $h = 1$, replica i multicasts $\overline{B}_{2,i} = [id, \overline{qc}, \overline{B}_{1,j}.r + 1, v, txn, 2, i]$.
  - if $h = 2$, replica i signs and multicasts $\overline{qc}$;
- **Leader Election.** Upon receiving $n - f$ distinct signed height-2 f-QCs of view v_{cur} and `async` $= true$, sign and multicast a coin share for view v_{cur}.
- **Exit Fallback.** Upon receiving or forming a coin-QC qc_{coin} (consisting of $f + 1$ coin shares) of view $\geq v_{cur}$ for the first time, multicast qc_{coin}. Let replica L be the fallback-chain leader elected by qc_{coin}. If `async` $= true$, update $r_{vote} \leftarrow \bar{r}_{vote}[L]$. Update `async` $\leftarrow false$, $v_{cur} \leftarrow qc_{coin}.v + 1$. Execute *Advance Round*, *Lock*, and then *Commit*.

Fig. 5: Asynchronous Fallback of `Ditto`

commits any blocks that have 2-chain support. Given that we waited for $2f + 1$ long-enough chains, with 2/3 probability the replicas will commit a block.

5 Implementation

We implement `Jolteon`, `Ditto`, and 2-chain VABA on top of a high-performance open-source implementation of HotStuff [6] [31]. We selected this implementation because it implements a Pacemaker [31], contrary to the implementation used in the original HotStuff paper[7]. Additionally, it provides well-documented benchmarking scripts to measure performance in various conditions, and it is close to a production system (it provides real networking, cryptography, and persistent storage). It is implemented in Rust, uses Tokio for asynchronous networking, ed25519-dalek for elliptic curve based signatures, and data-structures are persisted using RocksDB. It uses TCP to achieve reliable point-to-point channels, necessary to correctly implement the distributed system abstractions. We additionally use `threshold_crypto` to implement random coins, Our implementations are between 5,000 and 7,000 LOC, and a further 2,000 LOC of unit tests. We are open sourcing our implementations of `Jolteon`, and `Ditto` and 2-chain

[6] `https://github.com/asonnino/hotstuff/tree/3-chain`
[7] `https://github.com/hot-stuff/libhotstuff`

VABA We are also open sourcing all AWS orchestration scripts, benchmarking scripts, and measurements data to enable reproducible results [8].

2-chain VABA: As a by-product of `Ditto`, we improve the block-commit latency of VABA [4] from expected 16.5 rounds to expected 10.5 rounds, through chaining and adopting the 2-chain commit rule. We refer to the improved version as 2-chain VABA and the analysis can be found in Appendix C. The 2-chain VABA implementation is obtained by disabling the synchronous path of `Ditto`.

`Ditto` **with exponential backoff:** To improve the latency performance of `Ditto` under long periods of asynchrony or leader attacks, we adopt an exponential backoff mechanism for the asynchronous fallback as follows. We say a replica executes the asynchronous fallback consecutively x times if it only waits for the timer to expire for the first fallback, and skips waiting for the timer and immediately sends timeout for the rest $x-1$ fallbacks. Initially, replicas only execute asynchronous fallback consecutively $x = 1$ time. However, if a replica, within the timeout, does not receive from the steady state round-leader immediately after the fallback, it will multiply x by a constant factor (5 in our experiments); otherwise, the replica resets $x = 1$. For instance, during long periods of asynchrony or leader attacks, the number of consecutively executed fallbacks would be exponentially increasing $(1, 5, 25, ...)$; while during periods of synchrony and honest leaders, the number of consecutively executed fallbacks is always 1.

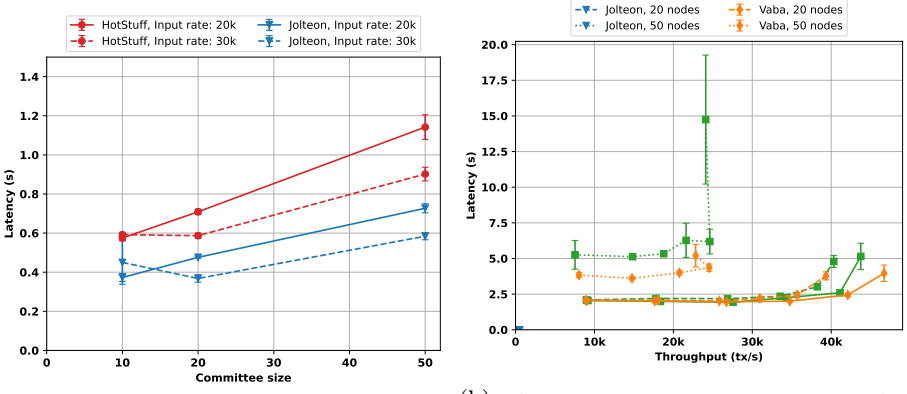
6 Evaluation

We evaluate the throughput and latency of our implementations through experiments on Amazon Web Services (AWS). We particularly aim to demonstrate (i) that `Jolteon` achieves the theoretically lower block-commit latency than 3-chain DiemBFT under no contention and (ii) that the theoretically larger message size during view-change does not impose a heavier burden, making `Jolteon` no slower than 3-chain DiemBFT under faults (when the view-change happens frequently). Additionally we aim to show that `Ditto` adapts to the network condition, meaning that (iii) it behaves similarly to `Jolteon` when the network is synchronous (with and without faults) and (iv) close to our faster version of VABA (2-chain) when the adversary adaptively compromises the leader.

We deploy a testbed on AWS, using m5.8xlarge instances across 5 different AWS regions: N. Virginia (us-east-1), N. California (us-west-1), Sydney (apsoutheast-2), Stockholm (eu-north-1), and Tokyo (ap-northeast1). They provide 10Gbps of bandwidth, 32 virtual CPUs (16 physical core) on a 2.5GHz, Intel Xeon Platinum 8175, and 128GB memory and run Ubuntu server 20.04. This type of machines are well in the price range of commodity servers and fairly common for prototype testbeds of distributed systems.

We measure throughput and end-to-end latency as the performance metrics. Throughput is computed as the average number of committed transactions per

[8] `https://github.com/asonnino/hotstuff/tree/async`



(a) Comparative block-commit latency for 3-chain DiemBFT (HotStuff) and `Jolteon`. WAN measurements with 10, 20, or 50 replicas. No replica faults, 500KB mempool batch size and 512B transaction size.

(b) Comparative throughput-latency performance for 3-chain DiemBFT (HotStuff), `Jolteon`, `Ditto`, and 2-chain VABA. WAN measurements with 10, 20, or 50 replicas. No replica faults, 500KB mempool batch size, and 512B transaction size. Leader constantly under DoS attack.

Fig. 6: Evaluations of `Jolteon` and `Ditto`.

second, and end-to-end latency measures the average time to commit a transaction from the moment it is submitted by the client. Compared with the block-commit latency in our theoretical analysis, end-to-end latency also includes the queuing delay of the transaction when the clients' input rate is high which helps identify the capacity limit of our system.

In all our experiments, the transaction size is set to be 512 bytes and the mempool batch size is set to be 500KB. We deploy one benchmark client per node submitting transactions at a fixed rate for a duration of 5 minutes (to ensure we report steady state performance). We set the timeout to be 5 seconds for experiments with 10 and 20 nodes, and 10 seconds for 50 nodes, so that the timeout is large enough for not triggering the pacemaker of `Jolteon` and fallback of `Ditto`. In the following sections, each measurement in the graphs is the average of 3 runs, and the error bars represent the standard deviation.

To find the peak performance of our system, we keep increasing the transaction submission rate of the clients until the capacity of the system is saturated. As a result, the latency-throughput measurements in the figures share similar patterns: the throughput of the system first increases with stable latency (dominated by the network delay) before the saturation point; then the throughput stops increasing and the latency increases significantly (due to high queuing delay) as the transaction submission rate exceeds the system's maximum capacity.

Due to space limit, more evaluations can be found in Appendix D.

6.1 Evaluation of `Jolteon`

In this section, we compare `Jolteon` with our baseline 3-chain DiemBFT implementation in two experiments. First in Figure 6a we run both protocol with a varying system size (10, 20, 50 nodes). In order to remove any noise from the mempool, this graph does not show the end-to-end latency for clients but the

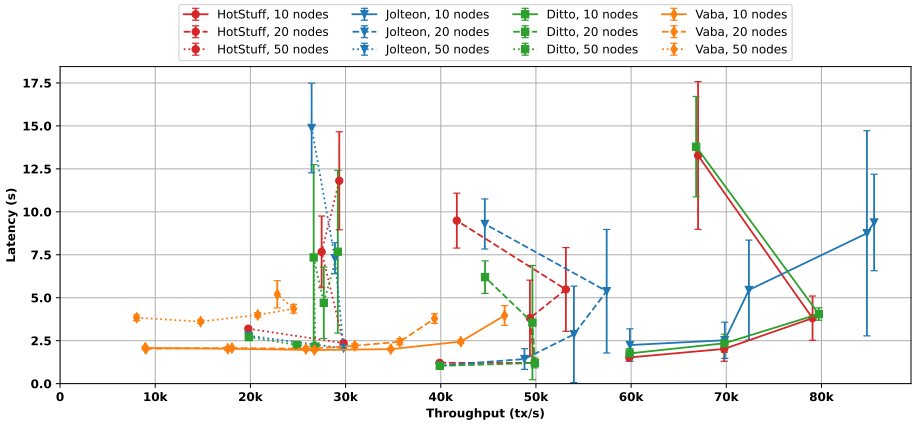


Fig. 7: Comparative throughput-latency performance for 3-chain DiemBFT (HotStuff), `Jolteon`, `Ditto`, and 2-chain VABA WAN measurements with 10, 20, or 50 replicas. No replica faults, 500KB mempool batch size and 512B transaction size.

time it takes for a block to be committed. As the Figure illustrated `Jolteon` consistently outperforms 3-chain DiemBFT by about $200 - 300$ms of latency which is around one round-trip across the world and both systems scale similarly. In Figure 7 this effect is less visible due to the noise of the mempool (end-to-end latency of around 2 secs), but `Jolteon` is still slightly faster than 3-chain DiemBFT in most experiments.

Finally, in Figure 8 (Appendix D) we run both protocols with 20 nodes and crashed 0, 1, or 3 nodes at the beginning of the experiment. This forces frequent view-changes due to the leader rotation that 3-chain DiemBFT uses in order to provide fairness. This is enables us to see the overall impact of the more costly view-change slowing down `Jolteon`. As we can see, `Jolteon` again outperforms 3-chain DiemBFT in most settings due to the 2-chain commit enabling more frequent commits under faults. As a result, we can conclude that there is little reason to pay the extra round-trip of 3-chain DiemBFT in order to have a theoretically linear view-change.

6.2 Evaluation of `Ditto`

Synchronous and fault-free executions. When all replicas are fault-free and the network is synchronous, we compare the performances of the four protocol implementations in Figure 7. As we can observe from the figure, the synchronous path performance of `Ditto` is very close to that of `Jolteon`, when the quadratic asynchronous fallback of `Ditto` and the quadratic pacemaker of `Jolteon` is not triggered. On the other hand, the performance of 2-chain VABA is worse than `Jolteon` and `Ditto` in this setting, due to its quadratic communication pattern – instead of every replica receiving the block metadata and synchronizing the transaction payload with only one leader per round in `Jolteon` and `Ditto`, in VABA every replica will receive and synchronize with $O(n)$ leaders per round.

Attacks on the leaders. Figure 6b presents the measurement results. When the eventual synchrony assumption does not hold, either due to DDoS attacks

on the leaders or adversarial delays on the leaders' messages, 3-chain DiemBFT and `Jolteon` will have no liveness, i.e., the throughput of the system is always 0. The reason is that whenever a replica becomes the leader for some round, its proposal message is delayed and all other replicas will timeout for that round. On the other hand, `Ditto` and 2-chain VABA are robust against such adversarial delays and can make progress under asynchrony. The performance of the 2-chain VABA protocol implementation is not affected much by delaying a certain replica's proposal. Therefore, we use it as a baseline to compare with our `Ditto` protocol implementation. Our results, confirm our theoretical assumption as the asynchronous fallback performance of `Ditto` is very close to that of 2-chain VABA under 10 or 20 nodes, and slightly worse than 2-chain VABA under 50 nodes. This extra latency cost is due to the few timeouts that are triggered during the exponential back-off.

Take away. To conclude, there is little reason not to use `Ditto` as our experiments confirm our theoretical bounds. `Ditto` adapts to the network behavior and achieves almost optimal performance. The only system that sometimes outperforms `Ditto` is 2-chain VABA during intermittent periods of asynchrony as it does not pay the timeout cost of `Ditto` when deciding how to adapt. This, however, comes at a significant cost when the network is good and in our opinion legitimizes the superiority of `Ditto` when run over the Internet.

7 Related Work

Eventually synchronous BFT. BFT SMR has been studied extensively in the literature. A sequence of efforts [8, 6, 7, 14, 31, 18] have been made to reduce the communication cost of the BFT SMR protocols, with the state-of-the-art being HotStuff [31] that has $O(n)$ cost for decisions, a 3-chain commit latency under synchrony and honest leaders, and $O(n^2)$ cost for view-synchronization. `Jolteon` presents another step forward from HotStuff as we realize the co-design of the pacemaker with the commit rules enables removing one round without sacrificing the linear steady state. Two concurrent theoretical works propose a 2-chain variation of the HotStuff as well [16, 27]. However, the work of Rambaud et al. [27] relies on impractical cryptographic primitives to preserve a linear view-change (assuming still a quadratic pacemaker) whereas neither protocol provides a comprehensive evaluation to showcase that the extra view-change costs (which also applies in [16]) does not cause significant overheads. Most importantly, both protocols fail to realize the full power of 2-chain protocols missing the fact the view-change can become robust and DDoS resilient.

Asynchronous BFT. Several recent proposals focus on improving the communication complexity and latency, including HoneyBadgerBFT [25], VABA [4], Dumbo-BFT [15], Dumbo-MVBA [24], ACE [29], Aleph [13], and DAG-Rider [17]. The state-of-the-art protocols for asynchronous SMR have $O(n^2)$ cost per decision [29], or amortized $O(n)$ cost per decision after transaction batching [15, 24, 13, 17]. As mentioned in Section 5, our design and implementation separate

transaction dissemination (mempool) from the critical path of consensus to fairly evaluate the consensus protocols.

BFT with optimistic and fallback paths. To the best of our knowledge, [20] is the first asynchronous BFT protocol with an efficient steady state. Their asynchronous path has $O(n^3)$ communication cost while their steady state has $O(n^2)$ cost per decision, which was later extended [26] to an amortized $O(n)$. A recent paper [28] further improved the communication complexity of asynchronous path to $O(n^2)$ and the cost of the steady state to $O(n)$. The latency of these protocols is not optimized, e.g. latency of the protocol in [28] is $O(n)$. Moreover, these papers are theoretical in nature and far from the realm of practicality. Finally, a concurrent work named the Bolt-Dumbo Transformer (BDT) [23], proposes a BFT SMR protocol with both synchronous and asynchronous paths and provides implementation and evaluation. BDT takes the straightforward solution of composing three separate consensus protocols as black boxes. Every round starts with 1) a partially synchronous protocol (HotStuff), times-out the leader and runs 2) an Asynchronous Binary Agreement in order to move on and run 3) a fully asynchronous consensus protocol [15] as a fallback. Although BDT achieves asymptotically optimal communication cost for both paths this is simply inherited by the already known to be optimal back boxes. On the theoretical side, their design is beneficial since it provides a generally composable framework, but this generality comes at a hefty practical cost. BDT has a latency cost of 7 rounds (vs 5 of `Ditto`) at the fast path and of 45 rounds (vs 10.5 of `Ditto`) at the fallback, making it questionably practical. Finally, not opening the black-boxes stopped BDT from reducing the latency of HotStuff although it also has a quadratic view-change.

8 Conclusion

We first design a 2-chain version of HotStuff, named `Jolteon`, which leverages a quadratic view-change mechanism to reduce the latency of the standard 3-chain HotStuff. We then present `Ditto`, a practical byzantine SMR protocol that enjoys the best of both worlds: optimal communication on and off the steady state (linear and quadratic, respectively) and progress guarantees under the worst case asynchrony and DDoS attacks. We implement and experimentally evaluate all our systems to validate our theoretical analysis.

Acknowledgments

We thank our shepherd Aniket Kate and the anonymous reviewers at FC 2022 for their helpful feedback. This work is supported by the Novi team at Facebook. We also thank the Novi Research and Engineering teams for valuable feedback, and in particular Mathieu Baudet, Andrey Chursin, George Danezis, Zekun Li, and Dahlia Malkhi for discussions that shaped this work.

Bibliography

[1] I. Abraham, D. Malkhi, Kartik Nayak, Ling Ren, and Maofan Yin. Sync hotstuff: Simple and practical synchronous state machine replication. In *2020 IEEE Symposium on Security and Privacy (S&P)*, pages 106–118, 2020.

[2] Ittai Abraham, TH Hubert Chan, Danny Dolev, Kartik Nayak, Rafael Pass, Ling Ren, and Elaine Shi. Communication complexity of byzantine agreement, revisited. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing (PODC)*, pages 317–326, 2019.

[3] Ittai Abraham, Philipp Jovanovic, Mary Maller, Sarah Meiklejohn, Gilad Stern, and Alin Tomescu. Reaching consensus for asynchronous distributed key generation. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing (PODC)*, page 363–373, 2021.

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

[5] Erica Blum, Jonathan Katz, and Julian Loss. Network-agnostic state machine replication. *arXiv preprint arXiv:2002.03437*, 2020.

[6] Ethan Buchman, Jae Kwon, and Zarko Milosevic. The latest gossip on bft consensus. *arXiv preprint arXiv:1807.04938*, 2018.

[7] Vitalik Buterin and Virgil Griffith. Casper the friendly finality gadget. *arXiv preprint arXiv:1710.09437*, 2017.

[8] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance. In *Proceedings of the third symposium on Operating Systems Design and Implementation (NSDI)*, pages 173–186, 1999.

[9] Sourav Das, Zhuolun Xiang, and Ling Ren. Asynchronous data dissemination and its applications. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 2705–2721, 2021.

[10] Sourav Das, Tom Yurek, Zhuolun Xiang, Andrew Miller, Lefteris Kokoris-Kogias, and Ling Ren. Practical asynchronous distributed key generation. *Cryptology ePrint Archive, Report 2021/1591*, 2021.

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

[12] Michael J Fischer, Nancy A Lynch, and Michael S Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM (JACM)*, 32(2):374–382, 1985.

[13] Adam Gągol, Damian Leśniak, Damian Straszak, and Michał Świętek. Aleph: Efficient atomic broadcast in asynchronous networks with byzantine nodes. In *Proceedings of the 1st ACM Conference on Advances in Financial Technologies (AFT)*, pages 214–228, 2019.

[14] Guy Golan Gueta, Ittai Abraham, Shelly Grossman, Dahlia Malkhi, Benny Pinkas, Michael Reiter, Dragos-Adrian Seredinschi, Orr Tamir, and Alin Tomescu. Sbft: a scalable and decentralized trust infrastructure. In *2019 49th Annual IEEE/IFIP international conference on dependable systems and networks (DSN)*, pages 568–580. IEEE, 2019.

[15] Bingyong Guo, Zhenliang Lu, Qiang Tang, Jing Xu, and Zhenfeng Zhang. Dumbo: Faster asynchronous bft protocols. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 803–818, 2020.

[16] Mohammad M Jalalzai, Jianyu Niu, Chen Feng, and Fangyu Gai. Fast-hotstuff: A fast and resilient hotstuff protocol. *arXiv preprint arXiv:2010.11454*, 2020.

[17] Idit Keidar, Eleftherios Kokoris-Kogias, Oded Naor, and Alexander Spiegelman. All you need is dag. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing (PODC)*, 2021.

[18] Eleftherios Kokoris Kogias, Philipp Jovanovic, Nicolas Gailly, Ismail Khoffi, Linus Gasser, and Bryan Ford. Enhancing bitcoin security and performance with strong consistency via collective signing. In *25th Usenix Security Symposium*, pages 279–296, 2016.

[19] Eleftherios Kokoris Kogias, Dahlia Malkhi, and Alexander Spiegelman. Asynchronous distributed key generation for computationally-secure randomness, consensus, and threshold signatures. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 1751–1767, 2020.

[20] Klaus Kursawe and Victor Shoup. Optimistic asynchronous atomic broadcast. In *International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 204–215. Springer, 2005.

[21] Benoît Libert, Marc Joye, and Moti Yung. Born and raised distributively: Fully distributed non-interactive adaptively-secure threshold signatures with short shares. *Theoretical Computer Science*, 645:1–24, 2016.

[22] Julian Loss and Tal Moran. Combining asynchronous and synchronous byzantine agreement: The best of both worlds. *Cryptology ePrint Archive, Report 2018/235*, 2018.

[23] Yuan Lu, Zhenliang Lu, and Qiang Tang. Bolt-dumbo transformer: Asynchronous consensus as fast as pipelined bft. *arXiv preprint arXiv:2103.09425*, 2021.

[24] Yuan Lu, Zhenliang Lu, Qiang Tang, and Guiling Wang. Dumbo-mvba: Optimal multi-valued validated asynchronous byzantine agreement, revisited. In *Proceedings of the 39th Symposium on Principles of Distributed Computing (PODC)*, pages 129–138, 2020.

[25] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. The honey badger of bft protocols. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 31–42, 2016.

[26] HariGovind V Ramasamy and Christian Cachin. Parsimonious asynchronous byzantine-fault-tolerant atomic broadcast. In *International Con-*

ference On Principles Of Distributed Systems (OPODIS), pages 88–102. Springer, 2005.
[27] Matthieu Rambaud. Malicious security comes for free in consensus with leaders. *IACR Cryptol. ePrint Arch.*, 2020:1480, 2020.
[28] Alexander Spiegelman. In search for an optimal authenticated byzantine agreement. In *35th International Symposium on Distributed Computing (DISC)*, 2021.
[29] Alexander Spiegelman and Arik Rinberg. Ace: Abstract consensus encapsulation for liveness boosting of state machine replication. In *23rd International Conference on Principles of Distributed Systems (OPODIS)*, 2020.
[30] The DiemBFT Team. State machine replication in the diem blockchain, 2021. `https://developers.diem.com/docs/technical-papers/state-machine-replication-paper/`.
[31] Maofan Yin, Dahlia Malkhi, Michael K Reiter, Guy Golan Gueta, and Ittai Abraham. Hotstuff: Bft consensus with linearity and responsiveness. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing (PODC)*, pages 347–356, 2019.

A Correctness of `Jolteon`

Safety We begin by formalizing some notation.

- We call a block byzantine (honest) if it was proposed by a byzantine (honest) replica.
- We say that a block B is *certified* if a quorum certificate QC_B exists.
- $B_i \longleftarrow QC_i \longleftarrow B_{i+1}$ means that the block B_i is certified by the quorum certificate QC_i which is contained in the block B_{i+1}.
- $B_i \longleftarrow^* B_j$ means that the block B_j *extends* the block B_i. That is, there is exists a sequence $B_i \longleftarrow QC_i \longleftarrow B_{i+1} \longleftarrow QC_{i+1} \cdots \longleftarrow QC_{j-1} \longleftarrow B_j$

Definition 1 (Global direct-commit). *We say that a block B is* globally direct-committed *if $f+1$ honest replicas each successfully perform the **Vote** step on block B' proposal in round $B.r+1$, such that $B'.qc$ certifies B. These **Vote** calls invoke **Lock**, setting $qc_{high} \leftarrow B'.qc$, and return $f+1$ matching votes (that could be used to form a $QC_{B'}$ with f other matching votes).*

Lemma 1. *If an honest replica successfully performs the **Commit** step on block B then B is globally direct-committed.*

Proof. By the **Commit** condition, there exists a chain $B \longleftarrow QC_B \longleftarrow B' \longleftarrow QC_{B'}$ with $B'.r = B.r + 1$. The existence of $QC_{B'}$ implies that $f+1$ honest replicas did **Vote** for B'.

The next lemma follows from the voting rules and the definition of global direct commit.

Lemma 2. *If a block B is globally direct-committed then any higher-round TC contains qc_{high} of round at least $B.r$.*

Proof. By Definition 1, $f + 1$ honest replicas execute the **lock** step in round $B.r + 1$ and set qc_{high} to $B'.qc$ that certifies B (so $B'.qc.r = B.r$, B' is the block proposed in round $B.r + 1$). None of these honest replicas may have previously timed out in round $B.r + 1$, and timing out stops voting in a round (but the replicas voted for B').

Since qc_{high} is never decreased, a timeout message prepared by any of the above $f + 1$ honest replicas in rounds $> B.r$ contains a high qc of round at least $B.r$. By quorum intersection, timeout messages used to prepare the TC in any round $> B.r$ contain a message from one of these honest replicas, completing the argument.

Due to quorum intersection, we have

Observation 1 *If a block is certified in a round, no other block can gather $f + 1$ honest votes in the same round. Hence, at most one block is certified in each round.*

We can now prove the key lemma

Lemma 3. *For every certified block B' s.t. $B'.r \geq B.r$ such that B is globally direct-committed, $B \longleftarrow^* B'$.*

Proof. By Observation 1, $B \longleftarrow^* B'$ for every B' s.t. $B'.r = B.r$.

We now prove the lemma by induction on the round numbers $r' > B.r$.

Base case: Let $r' = B.r + 1$. B is globally direct-committed, so by Definition 1, there are $f + 1$ honest replicas that prepare votes in round $r' = B.r + 1$ on some block B_{r+1} such that $B \longleftarrow QC_B \longleftarrow B_{r+1}$. By Observation 1, only B_{r+1} can be certified in round r'.

Step: We assume the Lemma holds up to round $r' - 1 > B.r$ and prove that it also holds for r'. If no block is certified at round r', then the induction step holds vacuously. Otherwise, let B' be a block certified in round r' and let $QC_{B'}$ be its certificate. B is globally direct-committed, so by Definition 1, there are $f + 1$ honest replicas that have locked high qc in round $B.r + 1$. One of these replicas, v, must also have prepared a vote that is included in $QC_{B'}$ (as QC formation requires $2f + 1$ votes and there are $3f + 1$ total replicas).

Let $B'' \longleftarrow QC_{B''} \longleftarrow B'$ and denote $r'' = B''.r = QC_{B''}.r$. There are two cases to consider, $r'' \geq r$ and $r'' < r$. In the first case, by the induction assumption for round r'', $B \longleftarrow^* B''$ and we are done.

In the second case, $r'' < r < r'$ (the right inequality is by the induction step), i.e., the rounds for B'' and B' are not consecutive. Hence, B' must contain a TC for round $r' - 1$. By Lemma 2, this TC contains a qc_{high} with round $\geq r$.

Consider a successful call by an honest replica to vote for B'. The only way to satisfy the predicate to vote is to satisfy (2), which implies $B''.r \geq qc_{high}.r \geq B.r$, which is a contradiction to $r'' < r$.

As a corollary of Lemma 3 and the fact that every globally direct-committed block is certified, we have

Theorem 1. *For every two globally direct-committed blocks B, B', either $B \longleftarrow^* B'$ or $B' \longleftarrow^* B$.*

Let's call a successful invocation of the **Commit** step by a replica a local direct-commit. For every locally committed block, there is a locally direct-committed block that extends it, and due to Lemma 1, also a globally direct-committed block that extends it. Each globally committed block defines a unique prefix to the genesis block, so Theorem 1 applies to all committed blocks. Hence all honest replicas commit the same block at each position in the blockchain.

Furthermore, since all honest replicas commit the transactions in one block following the same order, honest replicas do not commit different transactions at the same log position.

Liveness

Lemma 4. *When an honest replica in round $< r$ receives a proposal for round r from another honest replica, it enters round r.*

Proof. Recall that a well-formed proposal sent by an honest replica contains either a TC or QC of round $r - 1$. When an honest replica receives such a proposal message, it will advance the round and enter round r.

Lemma 5. *If the round timeouts and message delays between honest replicas are finite, then all honest replicas keep entering increasing rounds.*

Proof. Suppose all honest replicas are in round r or above, and let v be an honest replica in round r.

We first prove that some honest replica enters round $r + 1$. If all $2f + 1$ honest replicas time out in round r, then v will eventually receive $2f + 1$ timeout messages, form a TC and enter round $r + 1$. Otherwise, at least one honest replica, v' – not having sent a timeout message for round r – enters round $r + 1$. For this, v' must have observed qc of round r and updated its qc_{high} accordingly.

Since qc_{high} is never decreased and included in timeout messages, if v' times out in any round $> r$, then its timeout message will trigger v to enter a round higher than r. Otherwise, v' must observe a QC in all rounds $> r$. In this case, an honest leader sends a proposal in some round $> r$. That proposal will eventually be delivered to v, triggering it to enter a higher round by Lemma 4.

In an eventually synchronous setting, the system becomes synchronous after the the global stabilisation time (GST). We assume a known upper bound Δ on message transmission delays among honest replicas (practically, a back-off mechanism can be used to estimate Δ) and let 4Δ be the local timeout threshold for all honest replicas in all rounds.

Rounds are consecutive, advanced by quorum or timeout certificates, and honest replicas wait for proposals in each round. We first show that honest replicas that receive a proposal without the round timer expiring accept the proposal, allowing the quorum of honest replicas to drive the system progress.

Lemma 6. *Let r be a round such that no QC has yet been formed for it and in which no honest replica has timed out. When an honest replica v receives a proposal of an honest leader of round r, v will vote for the proposal.*

Proof. The predicate in the **Vote** step for a proposal with block B in round r checks that (1) round numbers are monotonically increasing, and (2a) either the block extends the QC of the previous round ($B.qc.r + 1 = r$), or (2b) the round of extended qc ($B.qc.r$) isn't less than the maximum high qc round in the TC of the previous round.

For (1), by assumption none of the $2f + 1$ honest replicas have timed out, so no TC could have been formed for round r. Also by assumption, no QC has been formed. Hence, no honest replica may have entered or voted in a round larger than r. Round r has an honest leader, so when a honest replica executes **Vote** step the round r proposal, it does so for the first time and with the largest voting round.

For (2), we consider two cases. If $B.tc = \perp$, then by well-formedness of honest leader's proposal, the $B.qc$ it extends must have round number $r-1$, rounds are consecutive, and condition (a) holds.

If $B.tc$ is not empty, then it is a TC for round $r-1$, formed based on $2f+1$ timeout messages. In this case, $B.qc.r \geq \max\{qc_{high}.r \mid qc_{high} \in B.tc\}$ predicate determines whether the replica votes for the proposal. Since the leader is honest, $B.qc$ is the qc_{high} of the leader when the proposal was generated. The predicate holds as the honest leader updates the qc_{high} to have round at least as large as the qc_{high} of each timeout messages it receives (separately or within a forwarded TC).

We now show a strong synchronization for rounds with honest leaders.

Lemma 7. *Let r be a round after GST with an honest leader. Within a time period of 2Δ from the first honest replica entering round r, all honest replicas receive the proposal from the honest leader.*

Proof. When the first honest replica enters round r, if it is not the leader, it must have formed a TC for round $r-1$. Let v be the honest leader of round r. Since honest replicas forward TC to the leader of the next round, v will receive the TC and advance to round r within Δ time of the first honest replica entering round r.

Upon entering round r, v multicasts a proposal, which is delivered within Δ time to all honest replicas.

Liveness follows from the following

Theorem 2. *Let r be a round after GST. Every honest replica eventually locally commits some block B with $B.r > r$.*

Proof. Since the leaders are determined by round-robin and the number of byzantine replicas is bounded by f, we can find round $r' > r$ such that rounds $r', r'+1, r'+2$ all have honest leaders.

By Lemma 5 honest nodes enter increasing rounds indefinitely. Due to Lemma 7, all honest replicas receive round r' proposal with block B from the leader within 2Δ time of starting their round r' timer (by Lemma 4 triggering ones that haven't yet entered round r' to do so). By Lemma 6, honest replicas accept the proposal and vote for it. Within time Δ their votes are delivered to the leader of round $r'+1$, who forms a QC extending B and sends a proposal with a block $B_{r'+1}$. This proposal will be received by honest replicas within another Δ time, by Lemma 4 triggering them to enter round $r'+1$ before the local timer of 4Δ for round r' expires.

By Lemma 6, every honest replica accepts the proposal, prepares a vote and sends it to round $r'+2$ leader, who is also honest. At this point, since $f+1$ honest replicas voted for B, by Definition 1, B is globally direct-committed.

Continuing the argument, the honest leader of round $r'+2$ receives the votes to form the round $r'+1$ QC after at most 5Δ time of the first honest replica entering round r'. It then prepares and sends round $r'+2$ proposal that extends $QC_{B_{r'+1}}$. After at most another Δ time all honest replicas receive this proposal, leading them to enter round $r'+2$ (before local timer for round $r'+1$ expires[9]) and locally direct-commit B.

Since we assume that each client transaction will be repeatedly proposed by honest replicas until it is committed (see Section 2), eventually each client transaction will be committed by all honest replicas.

B Correctness of `Ditto`

Recall that we say a fallback-block $\overline{B}$ of view v by replica i is endorsed, if $\overline{B}$ is certified and there exists a coin-QC of view v that elects replica i as the leader.

We say a replica is in the Steady State of view v, if it has `async` $= false$ and $v_{cur} = v$; otherwise if `async` $= true$ and $v_{cur} = v$, we say the replica is in the asynchronous fallback of view v.

We use $\mathcal{C}(B)$ to denote the set of replicas who provide votes for the QC or f-QC for B.

If a block or f-block B is an ancestor of another block or f-block B' due to a chain of QCs or endorsed f-QCs, we say B' extends B. A block or f-block also extends itself.

Lemma 8. *Let B, B' both be endorsed f-blocks of the same view, or both be certified blocks of the same view. If the round number of B equals the round number of B', then $B = B'$.*

Proof. Suppose on the contrary that $B \neq B'$. Let r be the round number of B, B'.

Suppose that B, B' both are certified blocks of the same view. Since $n = 3f+1$ and $|\mathcal{C}(B)| = |\mathcal{C}(B')| = 2f+1$, by quorum intersection, $\mathcal{C}(B) \cap \mathcal{C}(B')$ contains

[9] Analogous to round r'. Moreover, as the round $r'+1$ leader enters the round first, no replica spends more than 3Δ time in round $r'+1$.

at least one honest replica h who voted for both B and B'. Without loss of generality, suppose that h voted for B first. According to the **Vote** step, h updates its $r_{vote} = r$ and will only vote for blocks with round number $> r_{vote}$ in the same view. Therefore h will not vote for B', thus B' cannot be certified, contradiction.

Suppose that B, B' both are endorsed f-blocks of the same view. By quorum intersection, at least one honest replica h voted for both B and B'. Without loss of generality, suppose that h voted for B first. Since B, B' are of the same view with elected leader L, according to the **Fallback Vote** step, after voting for B of round r, h updates $\bar{r}_{vote}[L] = r$ and will only vote for blocks of round number $> \bar{r}_{vote}[L]$. Thus h will not vote for B' and B' cannot be certified, contradiction.

Lemma 9. *For any chain that consists of only certified blocks and endorsed f-blocks, the adjacent blocks in the chain have consecutive round numbers, and nondecreasing view numbers. Moreover, for blocks of the same view number, no endorsed f-block can be the parent of any certified regular block.*

Proof. Suppose on the contrary that there exist adjacent blocks B, B' of round number r, r' where B is the parent block of B', and $r' \neq r+1$. If B' is a certified block, according to the **Vote** step, honest replicas will not vote for B' since $r' \neq r+1$, and B' cannot be certified, contradiction. If B is an endorsed f-block, according to the **Fallback Vote** step, honest replicas will not vote for B' since $r' \neq r+1$, and B' cannot be certified, contradiction. Therefore, the blocks in the chain have consecutive round numbers.

Suppose on the contrary that there exist adjacent blocks B, B' of view number v, v' where B is the parent block of B', and $v' < v$. Since $n = 3f+1$ and $|\mathcal{C}(B)| = |\mathcal{C}(B')| = 2f+1$, by quorum intersection, $\mathcal{C}(B) \cap \mathcal{C}(B')$ contains at least one honest replica h who voted for both B and B'. According to the **Vote** and **Fallback Vote** steps, h has its $v_{cur} = v$ when voting for B. According to the protocol, h only updates its v_{cur} in steps **Enter Fallback** and **Exit Fallback**, and v_{cur} is nondecreasing. Hence, h will not vote for B' since $v' < v_{cur}$, contradiction. Therefore, the blocks in the chain have nondecreasing view numbers.

Suppose on the contrary that there exist adjacent blocks B, B' of the same view number v where B is the parent block of B', and B is an endorsed f-block and B' is a certified block. Since $n = 3f+1$ and $|\mathcal{C}(B)| = |\mathcal{C}(B')| = 2f+1$, by quorum intersection, $\mathcal{C}(B) \cap \mathcal{C}(B')$ contains at least one honest replica h who voted for both B and B'. According to the **Fallback Vote** step, h has $\texttt{async} = \mathit{true}$ when voting for B. According to the **Vote** step, h has $\texttt{async} = \mathit{false}$ when voting for B'. The only step for h to set its `async` to false is the **Exit Fallback** step, where h also updates its current view to $v+1$. Then, h will not vote for the view-v block B', contradiction. Therefore, for blocks of the same view number, no fallback-block can be the parent of any regular block.

Lemma 10. *Let B, B' both be endorsed f-blocks of the same view, then either B extends B' or B' extends B.*

Proof. Suppose on the contrary that B, B' do not extend one another, and they have height numbers h, h' respectively. Let L be the replica that proposes B, B'

and elected by the coin-QC. Without loss of generality, assume that $h \leq h'$. According to the **Fallback Vote** step, each honest replica votes for f-blocks of L with strictly increasing height numbers. Since by quorum intersection, there exists at least one honest replica h that voted for both B and B', thus we have $h < h'$. Since $h' > h$, and according to the **Fallback Vote** step, there must be another endorsed f-block B'' by L of height h that B' extends, and B, B'' do not extend one another. Similarly, by quorum intersection, at least one honest replica voted for both B and B'', which is impossible since each honest replica votes for f-blocks of L with strictly increasing height numbers. Therefore, B, B' extend one another.

Lemma 11. *If there exist two adjacent certified or endorsed blocks B_r, B_{r+1} in the chain with consecutive round numbers $r, r+1$ and the same view number v, then any certified or endorsed block of view number v that ranks no lower than B_r must extend B_r.*

Proof. Suppose on the contrary that there exists a block of view number v that ranks no lower than B_r and does not extend B_r. Let B be such a block with the smallest rank, then the parent block B' of B also does not extend B_r, but ranks no higher than B_r. By Lemma 9, there are 3 cases: (1) B_r, B_{r+1} are certified blocks, (2) B_r is a certified block and B_{r+1} is an endorsed blocks, and (3) B_r, B_{r+1} are endorsed blocks.

- Suppose B_r, B_{r+1} are certified blocks. By quorum intersection, $\mathcal{C}(B) \cap \mathcal{C}(B_{r+1})$ contains at least one honest replica h who voted for both B and B_{r+1}. According to the **Vote** and **Lock** step, when h votes for B_{r+1}, it updates qc_{high} to be the QC of B_r.
  - Suppose that B is a certified block, by Lemma 8, B must has round number $\geq r+2$ otherwise B will extend B_r. If the parent block B' of B has view number $< v$, then B' ranks lower than B_r. If B' has view number v, by Lemma 9 and 8, B' is a certified block and has round number $< r$, thus also ranks lower than B_r. According to the **Vote** step, h will not vote for B since B' has rank lower than its qc_{high}, contradiction.
  - Suppose that B is a endorsed block, let B_1 be the height-1 endorsed block of view v, and let B'_1 be the parent block of B_1, which ranks no higher than B_r. If B'_1 is a certified block, by the same argument above we have B'_1 ranks lower than B_r. If B'_1 is an endorsed block, since B'_1 ranks no higher than B_r its has view number $\leq v$. By definition an endorsed block ranks higher than a certified block if they have the same view number, thus B'_1 must have view number $< v$ and rank lower than B_r. By quorum intersection, at least one honest replica h' voted for both B_1 and B_{r+1}, and has qc_{high} to be the QC of B_r. However, according to the **Fallback Vote** step, h' will not vote for B_1 since B'_1 ranks lower than its qc_{high}, contradiction.
- Suppose B_r is a certified block and B_{r+1} is an endorsed blocks.
  - Suppose that B is a certified block, by Lemma 8, B must has round number $\geq r+1$ otherwise B will equal B_r. Since B, B_r does not extend

one another, by Lemma 9, there exists a round-r block $B'_r \neq B_r$ that B extends. By Lemma 9, B'_r has view number $\leq v$. If B'_r has view number v, by Lemma 8, $B_r = B'_r$, contradiction. If B'_r has view number $\leq v-1$, by quorum intersection, there exists at least one honest replica that voted for both B'_r and B_r. According to the steps **Vote, Fallback Vote**, after voting for B'_r, h sets its $r_{vote} = r$ and will not vote for regular blocks of round number $\leq r$. Hence, h will not vote for B_r of round number r, contradiction.

- Suppose that B is an endorsed block, by Lemma 10, B and B_{r+1} extend one another. By Lemma 9, no endorsed f-block can be the parent of any certified block of the same view, therefore B must extend B_r, contradiction.

– Suppose B_r, B_{r+1} are endorsed blocks.
  - Suppose that B is a certified block. Since B, B_r both have view number v, and by definition an endorsed block ranks higher than a certified block if they have the same view number, B ranks lower than B_r, contradiction.
  - Suppose that B is an endorsed block. By Lemma 10, B and B_r extend one another. Since B ranks no lower than B_r and endorsed f-blocks in the chain have consecutive round numbers, we have B extends B_r, contradiction.

Therefore, any certified or endorsed block of view number v that ranks no lower than B_r must extend B_r.

Lemma 12. *If a block B_r is committed by some honest replica due to a 2-chain starting from B_r, and another block $B'_{r'}$ is committed by some honest replica due to a 2-chain starting from $B'_{r'}$, then either B_r extends $B'_{r'}$ or $B'_{r'}$ extends B_r.*

Proof. Suppose on the contrary that $B_r, B'_{r'}$ are committed by honest replica, but they do not extend one another. Suppose there exist two adjacent certified or endorsed blocks B_r, B_{r+1} in the chain with consecutive round numbers $r, r+1$ and the same view number v. Suppose there also exist two adjacent certified or endorsed blocks $B_{r'}, B'_{r'+1}$ in the chain with consecutive round numbers $r', r'+1$ and the same view number v'. Without loss of generality, suppose that $v \leq v'$.

If $v = v'$, without loss of generality, further assume that $r \leq r'$. By Lemma 11, any certified or endorsed block of view number v that ranks no lower than B_r must extend B_r. Since $v = v'$ and $r \leq r'$, $B'_{r'}$ ranks no lower than B_r, and thus $B'_{r'}$ must extend B_r, contradiction.

If $v < v'$, by Lemma 9, there exist blocks that $B'_{r'}$ extends, have view numbers $> v$, and do not extend B_r. Let B be such a block with the smallest view number. Then the parent block B' of B also does not extend B_r, and has view number $\leq v$. By quorum intersection, at least one honest replica h voted for both B_{r+1} and B. According to the **Lock** step, after h voted for B_{r+1} in view v, it updates qc_{high} to be the QC of B_r. Then, when h votes for B, according to the steps **Vote, Fallback Vote**, the rank of the parent block B' must be $\geq qc_{high}.rank$, and thus no lower than the rank of B_r. By definition B' has view number $\leq v$,

hence B' must have view number v. By Lemma 11, B' of view number v and ranks no lower than B_r must extend B_r, contradiction.

Therefore, we have either B_r extends $B'_{r'}$, or B_r extends $B'_{r'}$.

Theorem 3 (Safety). *Honest replicas do not commit different transactions at the same log position.*

Proof. First we show that if blocks B and B' are committed at the same height in the blockchain by honest replicas, then $B = B'$. Suppose that B is committed due to a block B_l of round l being directly committed by a 2-chain, and B' is committed due to a block B_k of round k being directly committed by a 2-chain. By Lemma 12, either B_l extends B_k or B_l extends B_k, which implies that $B = B'$.

Since all honest replicas commit the transactions in one block following the same order, the theorem is true.

Lemma 13. *If all honest replicas enter the asynchronous fallback of view v by setting* `async` $= true$*, then eventually they all exit the fallback and set* `async` $= false$*. Moreover, with probability* $2/3$*, at least one honest replica commits a new block after exiting the fallback.*

Proof. Suppose that all honest replicas set `async` $= true$ and have $v_{cur} = v$. Let h denote the honest replica who has the highest ranked qc_{high} when entering the fallback.

We first show that the height-1 fallback-block proposed by h will be voted by all honest replicas. Suppose on the contrary that some honest replica h' does not vote for the height-1 f-block proposed by h. According to the step **Enter Fallback, Fallback Vote**, the only possibility is that $qc.rank < qc_{high}.rank$, where qc is the QC contained in the height-1 f-block by h and qc_{high} is the highest QC of h' when voting for the f-block. However, this contradicts the assumption that h holds the highest ranked qc_{high}. Hence, all honest replicas will vote for the height-1 fallback-block proposed by h.

After the height-1 f-block is certified with $2f+1$ votes that forms an f-QC $\overline{qc}_1$, according to the **Fallback Propose** step, any replica can propose a height-2 f-block extending it. All honest replicas will vote for the height-2 f-block according to the **Fallback Vote** step, and thus it will be certified with $2f + 1$ votes. Then honest replicas will multicast the f-QC for its height-2 f-block.

Since any honest replica i can obtain a height-2 f-QC and multicast it, any honest replica will receive $2f+1$ valid height-2 f-QCs and then sign and multicast a coin share of view v in the **Leader Election** step. When the qc_{coin} of view v that elects some replica L is received or formed at any honest replica, at least one honest replica has received $2f + 1$ height-2 f-QCs before sending the coin share, which means that $2f + 1$ fallback-chains have 2 f-blocks certified. All honest replicas will receive the qc_{coin} eventually due to the forwarding. According to the **Exit Fallback** step, they will set `async` $= false$ and exit the fallback.

Since the coin-QC elects any replica to be the leader with probability $1/n$, and at least one honest replica receives $2f + 1$ height-2 f-QCs among all $3f + 1$

f-chains. With probability 2/3, the honest replica has the height-2 f-QC of the f-chain by the elected leader, thus have the 2 f-blocks endorsed which have the same view number and consecutive round numbers. Therefore, the honest replica can commit the height-1 f-block according to the **Commit** step.

Theorem 4 (Liveness). *Each client transaction is eventually committed by all honest replicas.*

Proof. First we show that all honest replicas keep committing new blocks with high probability. We prove by induction on the view numbers.

We first prove for the base case where all honest replicas start their protocol with view number $v = 0$. If all honest replicas eventually all enter the asynchronous fallback, by Lemma 13, they eventually all exit the fallback, and a new block is committed at least one honest replica with probability 2/3. According to the **Exit Fallback** step, all honest replicas enter view $v = 1$ after exiting the fallback. If at least one honest replica never set `async` $= true$, this implies that the sequence of QCs produced in view 0 is infinite. By Lemma 9, the QCs have consecutive round numbers, and thus all honest replicas keep committing new blocks after receiving these QCs.

Now assume the theorem is true for view $v = 0, ..., k-1$. Consider the case where all honest replicas enter the view $v = k$. By the same argument for the $v = 0$ base case, honest replicas either all enter the fallback and the next view with a new block committed with 2/3 probability, or keeps committing new blocks in view k. When the network is synchronous and the leaders are honest, the block proposed in the Steady State will always extend the highest QC, and thus voted by all honest replicas. Therefore, by induction, honest replicas keep committing new blocks with high probability.

Since we assume that each client transaction will be repeatedly proposed by honest replicas until it is committed (see Section 2), eventually each client transaction will be committed by all honest replicas.

Theorem 5 (Efficiency). *During the periods of synchrony with honest leaders, the amortized communication complexity per block decision is $O(n)$, and the block-commit latency is 5 rounds. During periods of asynchrony, the expected communication complexity per block decision is $O(n^2)$, and the expected block-commit latency is 10.5 rounds.*

Proof. When the network is synchronous and leaders are honest, no honest replica will multicast timeout messages. In every round, the designated leader multicast its proposal of size $O(1)$ (due to the use of threshold signatures for QC), and all honest replicas send the vote of size $O(1)$ to the next leader. Hence the communication cost is $O(n)$ per round and per block decision. For the block latency, since `Jolteon` adopts 2-chain commit and need one more round for all replicas to receive the 2-chain proof, the latency is $2 \times 2 + 1 = 5$ rounds.

When the network is asynchronous and honest replicas enter the asynchronous fallback, each honest replica in the fallback only broadcast $O(1)$ number of messages, and each message has size $O(1)$. Hence, each instance of the asynchronous

fallback has communication cost $O(n^2)$, and will commit a new block with probability 2/3. Therefore, the expected communication complexity per block decision is $O(n^2)$. The latency to finish one asynchronous fallback is 7 rounds, consisting of 1 round to exchange timeouts, $2 \times 2 = 4$ rounds to build height-1 and 2 f-blocks, 1 round to exchange height-2 f-QCs and 1 round to exchange randomness shares. Since the asynchronous fallback commits a new block with probability at least 2/3, the expected latency of the asynchronous path is $7 \times 1.5 = 10.5$ rounds.

C 2-chain VABA Details

The original VABA [4] is designed for single-shot value, runs in views, and each view consists of a leader nomination phase, a leader election phase and a view-change phase. The leader nomination phase uses n parallel instances of 3-chain HotStuff (with each replica as the stable leader) to broadcast and certify its value. This phase takes 4 RTTs or 8 rounds, because intuitively 3-chain commit needs 3 RTTs to commit a value and 1 extra RTT to ensure every replica receives the commit proofs. Then every replica exchanges the proofs showing the leader nomination phase has finished, and runs the leader election phase to decide which replica's value to commit. Finally, during view-change phase replicas exchange their local information about the elected leader's value, to ensure safety for entering the next view if no value is committed. With probability 2/3, each view of VABA will succeed and consensus is reached. Therefore, the latency of VABA is $(8 + 1 + 1 + 1) \times 1.5 = 16.5$ rounds in expectation.

VABA can be easily turned into a chained BFT SMR protocol, by having 3-chain HotStuff instances building chained blocks instead of just values during the leader nomination phase. As a result, the number of blocks to certify is 4 for each replica's VABA chain. However, as we shown in our asynchronous fallback protocol (Figure 5), the number of blocks can be reduced to just 2, by the following observations: (1) We adopt 2-chain commit instead of 3-chain commit, and thus immediately reducing the latency by 1 RTT; (2) We observe the one extra RTT during the leader nomination phase for exchanging commit proofs is unnecessary once we have chained blocks, as the information is embedded in the blockchain and replicas can learn the commit from the chain. Therefore, we are able to reduce the latency of each view by 4 rounds (2 RTTs), and since the expected number of views VABA needs to execute is 3/2, the reduced latency is $4 \times 3/2 = 6$ rounds.

In our implementation, we obtain 2-chain VABA protocol by simply disabling the synchronous path of `Ditto`. More specifically, 2-chain VABA is obtained by never allowing any leader-replica to propose when not in fallback, and setting the timeout τ to 0 in `Ditto` (Figure 4 and 5). Then, whenever the replica enters a new view, it broadcasting timeouts carrying its highest QC for the new view and enters the fallback. Therefore, the synchronous path is never executed, and the exchange of timeout messages acts as the view-change phase in the original VABA [4].

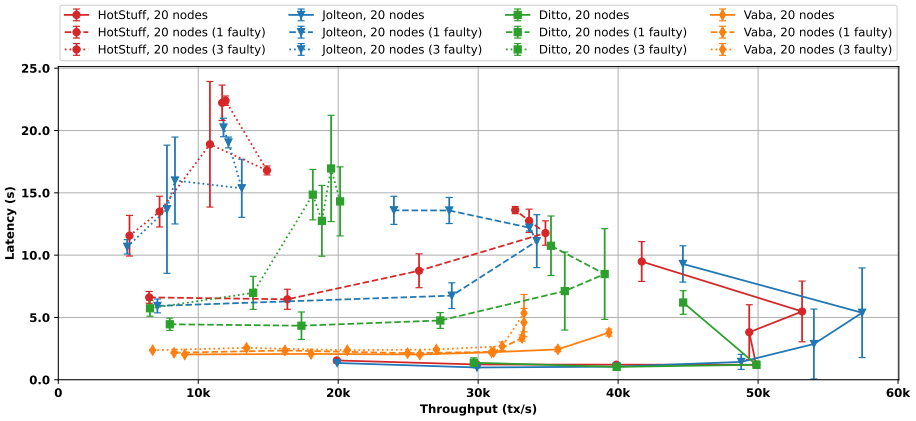


Fig. 8: Comparative throughput-latency performance for 3-chain DiemBFT (HotStuff), `Jolteon`, `Ditto`, and 2-chain VABA WAN measurements with 20 replicas. 0, 1, and 3 faults, 500KB mempool batch size and 512B transaction size.

D More Evaluations

Crash faults. In this experiment, we run the three protocol implementations with 20 nodes and crash 1 or 3 nodes from the beginning of the execution. The throughput-latency results are summarized in Figure 8. As we can observe from the figure, 2-chain VABA has the most robust performance under faults, where the latency only slightly increases before the throughput exceeds $30k$ tps. The reason is that by design 2-chain VABA can make progress as long as at least $2f+1$ nodes are honest, and crashing some of the nodes only add latency to the views when these nodes are elected as the leader of the views. The performance of 3-chain DiemBFT and `Jolteon` are more fragile under faults, where the peak performance degrades to $30k$ tps with about 10 seconds latency under 1 fault and $10k$ tps with about 15 seconds latency under 3 faults. The reason is that whenever a round-r leader is faulty, replicas need to wait for two timeouts, which is 10sec since the timeout is set to be 5sec, to enter round $r+1$ from round $r-1$. `Ditto` under faults performs better than `Jolteon` but worse than 2-chain VABA. Compared to 2-chain VABA, replicas in `Ditto` need to wait for a timeout of 5sec to enter the asynchronous fallback whenever the leader is faulty; compared to `Jolteon`, replicas only need to wait for one timeout (5sec) to enter the fallback, which makes progress efficiently even under faults (as the 2-chain VABA under faults suggests).

Crash faults and attacks on the leaders. Figure 9 presents the measurement results. The throughput of 3-chain DiemBFT and `Jolteon` is again 0 under leader attacks for the same reason mentioned above. 2-chain VABA makes progress even under leader attacks, and its performance is also robust against

1 or 3 node crashes. The performance of `Ditto` is very close to that of 2-chain VABA, since our implementation adopts the exponential backoff mechanism for the asynchronous fallback. One interesting artifact of the exponential backoff is that, compared to the case under just crash faults and no DDoS (Figure 8), `Ditto` has better performance under both crash faults and DDoS (Figure 9). The reason is that under long periods of leader attacks, the `Ditto` will skip waiting for the time to expire for most of the views, and directly send timeouts and enter the fallback.

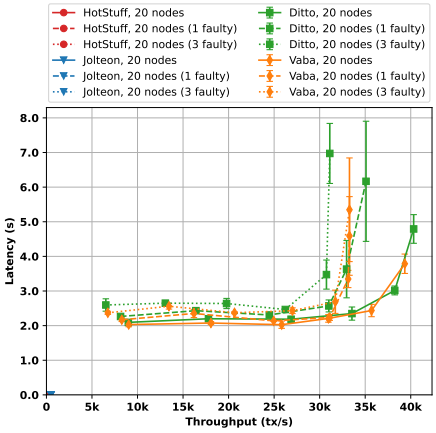


Fig. 9: Comparative throughput-latency performance for 3-chain DiemBFT (HotStuff), `Jolteon`, `Ditto`, and 2-chain VABA. WAN measurements with 20 replicas. 0, 1, and 3 faults, 500KB mempool batch size and 512B transaction size. Leader constantly under DoS attack.