BBCA-CHAIN: Low Latency, High Throughput BFT Consensus on a DAG

Dahlia Malkhi, Chrysoula Stathakopoulou, and Maofan Yin

Chainlink Labs

Abstract. This paper presents a partially synchronous BFT consensus protocol powered by BBCA, a lightly modified Byzantine Consistent Broadcast (BCB) primitive. BBCA provides a Complete-Adopt semantic through an added probing interface to allow either aborting the broadcast by correct nodes or exclusively, adopting the message consistently in case of a potential delivery. It does not introduce any extra type of messages or communication cost to BCB.

BBCA is harnessed into BBCA-CHAIN to make direct commits on a chained backbone of a causally ordered graph of blocks, without any additional voting blocks or artificial layering. With the help of Complete-Adopt, the additional knowledge gained from the underlying BCB completely removes the voting latency in popular DAG-based protocols. At the same time, causal ordering allows nodes to propose blocks in parallel and achieve high throughput. BBCA-CHAIN thus closes up the gap between protocols built by consistent broadcasts (e.g., Bullshark) to those without such an abstraction (e.g., PBFT/HotStuff), emphasizing their shared fundamental principles.

Using a Bracha-style BCB as an example, we fully specify BBCA-CHAIN with simplicity, serving as a solid basis for high-performance replication systems (and blockchains).

1 Introduction

A consensus protocol allows a network of nodes to consistently commit a sequence of values (referred to as blocks). Usually, practical solutions assume a partially synchronous network that may suffer from temporary outages or periods of congestion, where protocols can tolerate Byzantine faulty nodes exhibiting arbitrary behavior so long has a quorum of two-thirds are correct.

After decades, this line of research has reached the known communication complexity lower bound that is linear [43, 29] for steady operations and upon reconfiguration. However, these solutions operate in a sequential manner, extending the committed prefix one proposed block at a time. Hence, throughput is limited by the (underutilized) proposer's computational power, as it has to wait for a round-trip to collect a quorum to move on.

To fully utilize the idle time in which a proposer waits for the next sequential progress, recent works [19, 25, 14, 37, 30] harness Lamport's causal communication to allow blocks to be injected to the network in parallel to finalize their commit ordering. Each block references one or more previous blocks and the protocol relies on causal ordering, so that when nodes deliver the block, they have already delivered the blocks in its entire ancestry on a Direct Acyclic Graph (DAG). Then, the consensus logic "rides" on the DAG, so that nodes interpret it locally for a sequentially committed backbone (the chain made from blocks with crowns in Figure 1). Each backbone block also commits other non-backbone blocks it transitively references. Systems based on such a design have demonstrated significant throughput gains.

However, existing DAG-riding solutions come with a relatively complicated logic and suffer from several times higher latency. For example, a solution like Bullshark [37] takes a minimum of 4 chained blocks to commit one block, as illustrated on the left of Figure 2: a regular block at the head of the chain needs to first be delivered into the DAG; secondly it must be followed and referenced by a leader block which is restricted to odd layers of the DAG, hence may take two more blocks; third, the proposal must be referenced by a quorum of vote blocks that need to be on a subsequent, even layer dedicated to voting. Each block in this procedure has to be delivered by a Byzantine Consistent Broadcast (BCB) instance to guarantee a consistent view at all nodes. Internally, BCB already consists of multiple rounds of messages,

This paper introduces BBCA-CHAIN, a new DAG-based consensus solution that enables the same high throughput with much lower latency. BBCA-CHAIN substantially simplifies the overall consensus logic: it commits a backbone block via a single BBCA broadcast, a variant of BCB, without requiring vote blocks or waiting to broadcast into the artificial layers. Apart from using fewer steps than existing protocols, it also places fewer constraints on how the DAG could be grown.

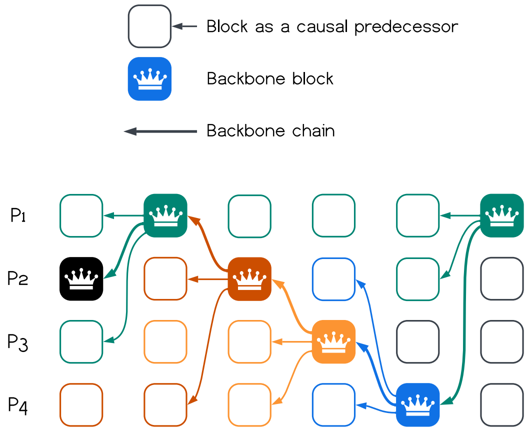

Fig. 1. A sequenced backbone on a DAG (thick arrows and solid blocks) and referenced blocks that commit with it (thin arrows and hollow blocks).

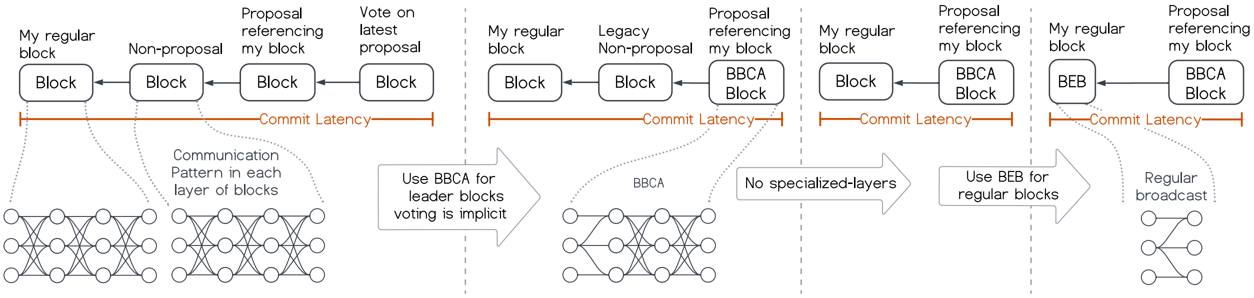

Fig. 2. Reducing Bullshark commit latency (left) via three mechanisms.

BBCA-Chain achieves these improvements via three mechanisms depicted in Figure 2 from left to right:

Instead of encoding the voting process by the DAG layers, it removes vote blocks altogether, because BBCA addresses voting internally. More specifically, by peeking inside BCB, we observe that it is already a multi-phase process similar to what other consensus protocols (such as PBFT [8]) do in their main logic. We add a shim on top of BCB to create BBCA (Byzantine Broadcast with Complete-Adopt), a new primitive that supports probing the local state of the broadcast protocol. The key new feature is a Complete-Adopt interface: if a broadcast is completed by any non-faulty node running the BBCA protocol, then a quorum probing via BBCA-probe will adopt it. Said differently, if a quorum probes and gets a BBCA-NOADOPT response, then it is guaranteed that no completion of the BBCA broadcast will ever occur to any (non-faulty) node. Through the Complete-Adopt interface from BBCA, one can extract this useful invariant to directly drive the committed backbone. As probing is done locally with a simple check of some protocol state variable, it does not incur any additional communication.

As no vote blocks are needed, BBCA-Chain does not need special layering, which further reduces the latency. As depicted in the middle of Figure 2, a block can be immediately referenced by a leader block and become committed when the leader block is completed by BBCA-broadcast. Finally, only leader blocks need to use BBCA. All other blocks can be broadcast with a "Best-Effort Broadcast" that simply sends the same message to all nodes via the inter-node channels (denoted as "BEB" block on the right of Figure 2).

The table below summarizes the latency in terms of consecutive network-trips of BBCA-Chain compared with state-of-art DAG solutions like Bullshark. The implementation of the BBCA broadcast is assumed to use an all-to-all protocol, since there is only a single BBCA invocation per view (see more details on broadcasts in the body of the paper).

	BBCA-Chain	Bullshark
Broadcast primitives	BEB (1 trip) / BBCA (3 trips)	BCB (3 trips)
leader block	1× BBCA	2× BCB
non-leader block	1× BEB + 1× BBCA	4× BCB

Table 1. Latency to commit a block on a DAG.

The Benefits of Causal Ordering. BBCA-Chain is an interesting mid-point between non-DAG and DAG-based consensus solutions, that enjoys the best of both worlds: the same latency as in those non-DAG protocols, the high throughput from a better utilization of processing power while waiting for backbone's progress, and the simplicity of checking a DAG to know the commitment progress.

In steady state, the way the sequenced backbone is formed in BBCA-Chain resembles traditional non-DAG consensus protocols. The difference is that in the latter, the network is underutilized as the protocol only disseminates blocks on the linear backbone, whose rate of progress is bounded by a quorum round-trip time, whereas DAG-based protocols can keep injecting non-backbone blocks, which carry useful payload, without waiting. Furthermore, the leader has to broadcast a (potentially large) block combining all the requests. Previous studies [38] have demonstrated that a single leader becomes a throughput bottleneck, even if it broadcasts blocks which only include references to transactions. BBCA-Chain leverages causal communication to allow nodes to inject blocks in parallel. Then the leader sends a (smaller) block that simply references other blocks. In addition, BBCA-Chain benefits from being able to parallelize the validation of blocks, offloading the computational burden from the backbone.

At the same time, the way BBCA-Chain handles failures borrows the simplified logic of DAG-based consensus. When leaders fail, the causal history allows nodes to accept leader proposals without reasoning about (un)locking. Compared to existing DAG-based protocols, BBCA-Chain further simplifies the solution by requiring nodes only to inject new-view blocks into the DAG, foregoing vote blocks.

Foregoing block-votes on the DAG has two additional far-reaching benefits.

1. It avoids the extra latency introduced in previous DAG-based solutions. Rather, BBCA-Chain makes use of acknowledgment messages which are in any case sent inside the BBCA broadcast protocol as votes.
2. Whereas votes must be non-equivocating, blocks do not need to. Because we got rid of vote blocks, there is no need to use BCB for non-leader blocks. Instead, regular blocks can be uniquely ordered via the backbone utilizing only causality. This change allows further reduction in latency.

2 Technical Overview

Consensus. In the celebrated State Machine Replication (SMR) approach, a network of validator nodes form consensus on a growing totally ordered log (or "chain") of transactions. The focus in this paper is on practical settings in which during periods of stability, transmission delays on the network are upper bounded, but occasionally the system may suffer unstable periods. This is known as the partial synchrony model [17]. The strongest resilience against Byzantine faults which is possible in this setting is f-tolerance, where $f < n/3$ for a network of n validators.

Parallel Leader Proposals. Concensus protocols require a leader node to propose the next transaction to be added to the chain and broadcast it to all validator nodes. Then, validators run a voting protocol to commit the leader proposal. In practice, to amortize the computation and communication per voting decision, leaders bundle multiple transactions in blocks in their proposals. However, this mechanism alone is not enough to ensure high throughput, as the larger the set of validators, the longer it takes for the leader to broadcast the proposal. During this time, the rest of nodes remain idle. To enable throughput scalability, we borrow from recent consensus protocols (e.g. [39, 23, 14, 37]) and allow parallel leader proposals.

Causal Broadcast. A Direct Acyclic Graph (DAG) captures a transport substrate that guarantees causal ordering of block delivery. Whenever a node is ready to broadcast a block, the block contains references to some locally delivered blocks. The consensus logic operates above the transport level, such that blocks are handled (received and generated) in causal order.

The Backbone. The general approach we take for DAG-based consensus is forming a totally ordered backbone of leader blocks which we therefore call backbone blocks (Figure 1). When a backbone block becomes committed, every leader block in its causal ancestry becomes committed as well. By walking from the earliest uncommitted leader block in the backbone towards the latest committed leader block, one can also commit into a total order those blocks outside the backbone but causally referenced by the backbone blocks, according to any predefined, deterministic DAG traversal order. Forming consensus on the backbone itself "rides" on the DAG: any validator node can independently and deterministically compute a commit structure to determine the order in which blocks become committed to the backbone by processing the DAG, without additional communication or historical states outside the DAG. In previous solutions, deducing the commit structure is somewhat complex, as the DAG consists of interleaving voting and non-voting block layers. As we shall see below, the commit structure in BBCA-CHAIN is very simple: backbone blocks become committed by themselves, without the need of voting block layers, utilizing the invariants from Complete-Adopt of BBCA.

Separating Data Availability. Borrowing from various prevailing systems, we assume that a separate data availability layer may be used for disseminating data blobs. First, parallel workers disseminate transactions and generate certificates of availability. Then, bundles of certificates are assembled into blocks which are totally ordered in the consensus layer. This way, when a leader proposes a block containing a bundle of certificates, validators trust that their data can be retrieved. As demonstrated in prior work [14] [42], offloading data dissemination to worker nodes helps a system to effectively scale out. Note that this mechanism alone is not enough to ensure high throughput; as it has been demonstrated [38] the throughput capacity of a consensus protocol, which does not support parallelized proposals, drops when the number of validator nodes increases, even if they only orders references to data. However, it can be combined with parallel leader proposals for improving performance and scalability. The data dissemination and data availability work is left out of scope in this paper and we note that it can be highly parallelized, and is not considered in the critical path of consensus or counted in latency analysis.

In the rest of this section, we first introduce the BBCA broadcast primitive, and then we explain how BBCA is used in BBCA-CHAIN to solve consensus on a DAG.

2.1 BBCA

The BBCA primitive is an abortable variation of BCB. Similar to BCB, BBCA has a dedicated sender which BBCA-broadcasts messages and no two correct nodes BBCA-complete different messages for the same BBCA instance.

Informally, inside BBCA, $f+1$ correct nodes must become locked on a unique message m before BBCA-complete is possible at any node. BBCA adds an interactive Complete-Adopt interface to facilitate view changes in partial synchrony which allows nodes to actively probe at any time regardless of their progress in BCB. Probing stops a node from completing the delivery, with one of two possible return values: BBCA-ADOPT for some locked message m or BBCA-NOADOPT, with the following guarantee:

The Complete-Adopt Invariant. If any correct node BBCA-COMPLETE$(bid, m, cert_m)$, then at least $f+1$ correct nodes return $\langle$BBCA-ADOPT$, bid, m, cert_m\rangle$ upon probing. In other words, if $f+1$ probes by correct nodes return $\langle$BBCA-NOADOPT$, bid\rangle$, BBCA-COMPLETE$(bid, m, cert_m)$ never occurs at any correct node.

We implement BBCA via an all-to-all regime à la Bracha broadcast [3]. In this form, BBCA takes 3 network trips. To further improve scalability, all-to-all communication may take place over a peer-to-peer gossip transport. Furthermore, acknowledgements may be batched and aggregated.

The table below summarizes the number of network trips in BBCA compared with other forms of broadcast. In all forms, the underlying communication may be carried either over all-to-all authenticated channels, or in a linear regime relayed through a sender who aggregates signatures.

Broadcast scheme	all-to-all	linear
Best Effort Broadcast (BEB)	1	1
Byzantine Consistent Broadcast (BCB)	2	3
BBCA	3	5

Table 2. Network trips in various broadcast schemes.

2.2 BBCA-Chain

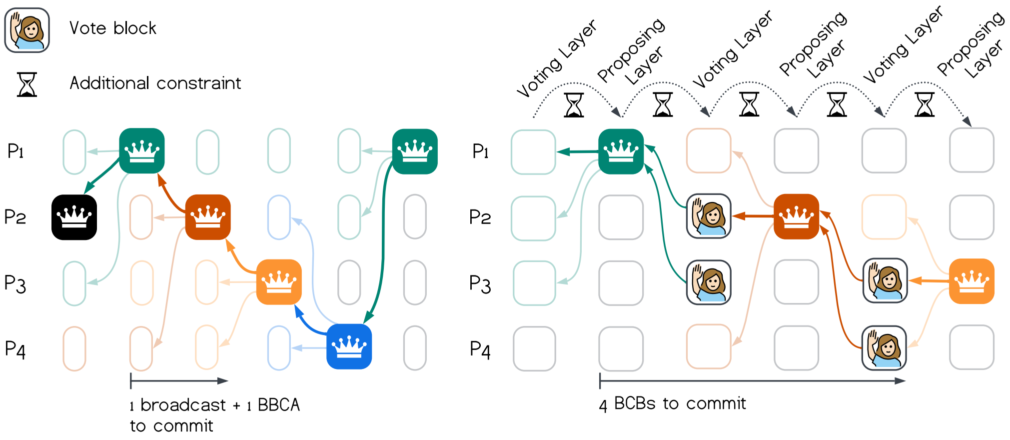

Fig. 3. Decision making latency in BBCA-Chain (left) compared with Bullshark (right).

BBCA-Chain is a view-by-view protocol. Each view reaches a consensus decision in a single BBCA step. BBCA-Chain forms a DAG of two types of blocks: backbone blocks and new-view blocks. Although there is no artificial layering, they can be logically grouped by views (as shown on the left of Figure 3). Both types of blocks may carry transaction payload.

A transition to view v occurs either upon BBCA-complete of the backbone block of view $v-1$, or when the new-view timer of view $v-1$ expires. In the second case, a node invokes BBCA-PROBE locally, and then embeds the return value in its new-view block. A new-view block must include one of the following three items: a BBCA-completed backbone block from view $v-1$, an adopted backbone block from probing, or a signed BBCA-NOADOPT result together with the highest completed block locally known. Before entering view v, each node other than the leader broadcasts a new-view block, and all nodes start their new-view timers. The leader of view v BBCA-broadcasts a backbone block. The block refers to the completed or adopted backbone block of view $v-1$, if available[*], otherwise it references $2f+1$ blocks with signed BBCA-NOADOPT.

The left graph of Figure 3 shows the fast, commit path, when the leader has BBCA-completed the backbone block of the previous view. Figure 4 illustrates the adopt path, with two different outcomes from nodes' probing results. The key observation is that the BBCA Complete-Adopt scheme guarantees that the fast and adopt paths are consistent.

BBCA allows backbone blocks to become committed by themselves, without any additional votes or DAG layering. This single-broadcast commit decision accomplishes a substantial reduction in latency to reach consensus, and also significantly simplifies the logic.

Non-backbone blocks are broadcast in parallel using a Best-Effort Broadcast where a node just directly sends a block to all nodes. These blocks become committed by being causally referenced by some blocks on the backbone, similar to

---

[*] As a technical matter, note that the code in Figure 6 encapsulates these references in a new-view block which the leader can embed inside a backbone block.

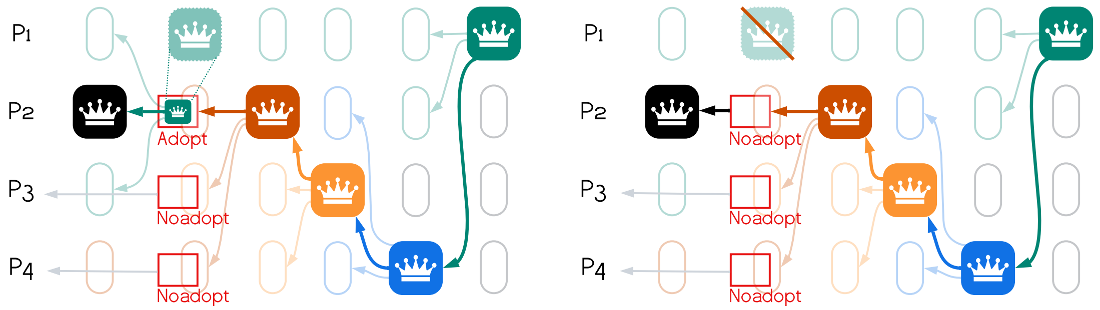

Fig. 4. When a view timer expires, each node needs to include a BBCA-probe result inside its new-view block. The left DAG depicts an adopted block, and some correct node may have just BBCA-completed it. The right DAG shows $2f + 1$ noadopt responses, proving that no block can be completed in that view.

"uncle blocks" in Ethereum. Since the local DAG of different nodes may evolve at different pace, the Complete-Adopt feature of BBCA preserves the safety of committed backbone blocks. BBCA-completed backbone blocks suffice to uniquely determine which non-backbone blocks to include in the total ordering.

It is worth noting that previous solutions like Bullshark need non-backbone blocks to vote in the consensus protocol, hence to prevent equivocation, these blocks are broadcast using BCB. Because BBCA-Chain gets rid of voting blocks altogether, there is no need to use BCB for non-backbone blocks, and this change allows further reduction in latency. While BCB is not required for correctness, it may be used for other reasons (e.g., to avoid keeping equivocating blocks in the DAG). Even if BCB is used for non-backbone blocks, BBCA-Chain exhibits an overall substantially lower latency than existing DAG-based protocols.

Overall, BBCA-Chain accomplishes a substantial reduction in latency over state-of-art DAG protocols. Figure 3 illustrates side-by-side the time to commit backbone blocks in BBCA-Chain (left) compared with a DAG-based solution like Bullshark (right). Hourglasses on the right side denote additional constraints posed on advancing layers by Bullshark.

3 The BBCA Primitive

BBCA-Chain invokes BBCA broadcast primitive, an abortable variant of BCB[†] for multiple times. Each BBCA instance has a unique identifier bid to be totally isolated in its progress. Similar to BCB, each instance of BBCA broadcast has a dedicated node as the sender who can invoke BBCA-broadcast(bid, m), to broadcast message m for instance bid. Eventually, the node may BBCA-complete$(bid, m, cert_m)$. We say for short that the node BBCA-completes m and $cert_m$ is a certificate to convince any node that m has been completed. A node may also abort a BBCA instance and probe its state at any time by invoking BBCA-probe(bid). This invocation returns in two possible ways. The BBCA instance returns $\langle$BBCA-adopt$, bid, m, cert_m\rangle$ for some message m with a certificate $cert_m$, to certify that some correct node might complete m and no correct node could complete a message $m' \neq m$. Otherwise, BBCA returns $\langle$BBCA-noadopt$, bid\rangle$. In the former case, we say in short that the node BBCA-adopts m, otherwise, we say that it does not BBCA-adopt any message. BBCA maintains the following properties:

Validity. If a correct sender BBCA-broadcasts a message m for instance bid, and no correct node probes bid, eventually every correct node BBCA-completes m.

Consistency. If a correct node either BBCA-adopts or BBCA-completes a message m for instance bid, and some other correct node BBCA-completes message m' for bid, then $m = m'$.

Complete-Adopt. If a message m is ever BBCA-completed by some correct node for instance bid, then at least $f + 1$ correct nodes get $\langle$BBCA-adopt$, bid, m, cert_m\rangle$ for bid, if they invoke BBCA-probe(bid). That is, if $f + 1$ correct nodes get $\langle$BBCA-noadopt$, bid\rangle$, then no correct node ever BBCA-completes any message for instance bid.

[†] Pronounced 'Bab-Ka', and loosely stands for Byzantine Abortable Broadcast with Complete-Adopt.

Integrity. If a correct node BBCA-adopts or BBCA-completes a message m for instance bid and the sender p is correct, then p has previously invoked BBCA-BROADCAST(bid, m).

BBCA is intialized with an external validity predicate $\mathcal{P}$, such that a correct node only BBCA-completes m if $\mathcal{P}(m)$ is true.

The pseudocode in Figure 5 implements BBCA by slightly modifying the Bracha broadcast protocol [3]. Upon sender's invocation of BBCA-BROADCAST, it broadcasts $\langle$INIT, $bid, m\rangle$. For a node p, upon receiving $\langle$INIT, $bid, m\rangle$ from the sender of instance bid, it checks that validity condition $\mathcal{P}(m)$ holds. If m is valid, p broadcasts $\langle$ECHO, $bid, m\rangle_{\sigma_p}$, where σ_p is the signature of p on $\langle$ECHO, $bid, m\rangle$, and becomes initialized, such that it will no longer be initialized again.

Upon receiving an ECHO message from $2f+1$ distinct nodes for bid and m with valid signatures, p becomes ready, unless p has already probed instance bid. Then it broadcasts a signed message $\langle$READY, $bid, m\rangle_{\sigma_p}$. The $2f+1$ signatures on $\langle$ECHO, $bid, m\rangle$ certify a node has sent a READY message for m.

Upon receiving a valid READY message from $2f+1$ distinct nodes for bid and m, and, BBCA-completes m in instance bid. The $2f+1$ signatures on $\langle$READY, $bid, m\rangle$ certify a node has completed m.

When probed in instance bid, if p has already become ready for m, it returns $\langle$BBCA-ADOPT, $bid, m, cert_m\rangle$ where $cert_m$ is a collection of $2f+1$ signed ECHO messages. Otherwise, it returns $\langle$BBCA-NOADOPT, $bid\rangle$.

```
Init:
    // $bid$ is a unique identifier for a BBCA-broadcast instance, it contains $bid.sender$ indicating the designated sender of the instance
    $bbcaState[bid].pendingMsgs \leftarrow \{\}$
    // $pendingMsgs[m].nEcho \leftarrow \{\}$ and
    // $pendingMsgs[m].nReady \leftarrow \{\}$
    // when message $m$ is first seen.
    $\forall p \in N, bbcaState[bid].receivedEcho[p] \leftarrow$ FALSE
    $\forall p \in N, bbcaState[bid].receivedReady[p] \leftarrow$ FALSE
    $bbcaState[bid].ready \leftarrow$ FALSE
    $bbcaState[bid].echo \leftarrow$ FALSE
    $bbcaState[bid].abort \leftarrow$ FALSE

interface BBCA-BROADCAST$(bid, m)$ :
    BECOMESINITIALIZED$(bid, m)$
    BROADCAST$(\langle$INIT, $bid, m\rangle)$

interface BBCA-PROBE$(bid)$ :
    let $bstate = bbcaState[bid]$
    let $mstates = bstate.pendingMsgs$
    if $\exists m : |mstates[m].nEcho| \geq 2f+1$ then
        return $\langle$BBCA-ADOPT, $bid, m, mstates[m].nEcho\rangle$
    else
        $bstate.abort \leftarrow$ TRUE
        return $\langle$BBCA-NOADOPT, $bid\rangle$
    end if

procedure BECOMESINITIALIZED$(bid, m)$ is
    $bbcaState[bid].echo \leftarrow$ TRUE
    BROADCAST$(\langle$ECHO, $bid, m\rangle_{\sigma_p})$

procedure BECOMESREADY$(bid, m)$ is
    if $!bbcaState[bid].abort$ then
        $bbcaState[bid].ready \leftarrow$ TRUE
        BROADCAST$(\langle$READY, $bid, m\rangle)$
    end if

upon receiving $\langle$INIT, $bid, m \neq \bot\rangle$ from $q$ do
    let $bstate = bbcaState[bid]$
    if $\mathcal{P}(m) \wedge q = bid.sender \wedge (!bstate.echo)$ then
        BECOMESINITIALIZED$(bid, m)$
    end if

upon receiving $\langle$ECHO, $bid, m \neq \bot\rangle_{\sigma_q}$ from $q$ do
    let $bstate = bbcaState[bid]$
    if $\mathcal{P}(m) \wedge (!bstate.receivedEcho[q])$ then
        $bstate.receivedEcho[q] \leftarrow$ TRUE
        let $mstate = bstate.pendingMsgs[m]$
        $mstate.nEcho \leftarrow mstate.nEcho \cup \{\sigma_q\}$
        if $!bstate.ready \wedge |mstate.nEcho| = 2f+1$ then
            BECOMESREADY$(bid, m)$
        end if
    end if

upon receiving $\langle$READY, $bid, m \neq \bot\rangle_{\sigma_q}$ from $q$ do
    let $bstate = bbcaState[bid]$
    if $\mathcal{P}(m) \wedge (!bstate.receivedReady[q])$ then
        $bstate.receivedReady[q] \leftarrow$ TRUE
        let $mstate = bstate.pendingMsgs[m]$
        $mstate.nReady \leftarrow mstate.nReady \cup \{\sigma_q\}$
        if ~~$!bstate.ready \wedge |mstate.nReady| = f+1$~~ then
            ~~BECOMESREADY$(bid, m)$~~
        end if
        if $|mstate.nReady| = 2f+1$ then
            BBCA-COMPLETE$(bid, m, mstate.nReady)$
        end if
    end if
```

Fig. 5. BBCA-broadcast primitive with a validity predicate $\mathcal{P}$ based on a all-to-all Bracha broadcast protocol. The blue text shows the simple tweak that is added to the vanilla Bracha to enable the Complete-or-Adopt scheme, whereas red text shows the part that is removed.

For completeness, we mention that another approach for implementation is a linear regime à la Cachin et al. [7]. In a linear form, BBCA takes 5 network trips, while the probing logic is similar (return the adopted m or promise not to sign the second acknowledgement):

1. The designated sender broadcasts the block to all nodes.
2. After the validation of the block, nodes respond with a signed acknowledgement over the block's digest.
3. The sender broadcasts an aggregated/concatenated proof of $2f + 1$ signatures.
4. Nodes respond to the proof from the sender with a signed acknowledgement over the proof.
5. The sender broadcasts an aggregate/concatenated proof of signatures, and nodes BBCA-complete it upon receipt.

Relationship to Other Broadcasts Bracha's "all-to-all" broadcast protocol is not only a BCB but also a Byzantine Reliable Broadcast (BRB), because it additionally guarantees reliable delivery of the broadcast message. Since BBCA is abortable, reliability is not guaranteed in presence of a faulty sender or network delays. However, when used in the construction of BBCA-CHAIN as we show later, reliability is gained for committed blocks through chaining: after seeing a BBCA-complete certificate for a block, later commits will causally reference the block itself or BBCA-adopt it.

Since we forego reliability for BBCA, we remove the extra trip shown in red text from the original Bracha protocol (Figure 5). Requiring a weaker broadcast primitive also allows the alternative linear form to preserve linearity, otherwise all-to-all broadcast would be required on top "linear" BCB in order to guarantee reliability. Therefore, BBCA makes a weaker, more basic primitive than BRB and reduces latency/message-complexity.

4 The BBCA-CHAIN Protocol

4.1 Model

We assume a set of n nodes which communicate via point-to-point authenticated channels such that up to $f = \frac{n-1}{3}$ of them may fail arbitrarily. We assume partially synchronous communication such that network delays are unbounded until some unknown Global Stabilization Time (GST). After GST, network delays are bounded by constant Δ. But because GST is unknown, the safety of the protocol cannot rely on Δ.

4.2 Consensus

In a consensus protocol, each correct node may take some payload (e.g., transactions), seal it in a block, and send it to the network. A **commit** statement is triggered to sequence blocks in their final order, with the following properties:

Consistency. For the sequences of **committed** blocks by any two nodes, one has to be the prefix of another.
Chain growth. The sequence of **committed** blocks keeps growing for all correct nodes.
Censorship resistance. After GST, if a correct node sends a block b to the network, then b is eventually **committed**.

BBCA-CHAIN commits blocks in views and in each view v, one node acts as leader, evaluated by each node via a GETPROPOSER(v) function. For simplicity, throughout the paper we assume a round-robin rotation, though more elaborate and/or adaptive schemes may be used. Nodes enter a new view with a view synchronization protocol, which we discuss in more detail in Section 5. Each node, upon entering a new view, starts a view timer and starts participating in a new BBCA instance (identified as $bid = \langle \text{GETPROPOSER}(v), v \rangle$). Nodes other than the leader broadcast a new-view block. The leader, however, proposes a backbone block B_v by invoking BBCA-BROADCAST(bid, B_v).

In the common case, a node completes view v and is ready to enter view $v + 1$ upon it BBCA-completes the backbone block of view v. The block that has completed gets marked as the final block for v and the protocol guarantees that every correct node eventually also commits this block. In this case, the new-view block for view $v + 1$ includes (1) a causal reference to the completed backbone block B_v, (2) the BBCA-complete certificate $cert_{B_v}$ proving that p completed B_v, (3) the view number v, and (optionally) a payload of pending transactions and causal references to help commit some additional previous blocks. For a node that is the leader of $v + 1$, instead of broadcasting this new-view block, it just embeds (or causally references to) it when it propose in view $v + 1$.

If at some node p the timer for a view v expires before a BBCA-complete of backbone block for view v, then p probes the BBCA instance of view v and finishes view v. This forces the BBCA instance to stop sending ready messages internally. If there exists a node that could BBCA-complete a backbone block B_v, then BBCA returns $\langle$BBCA-ADOPT, B_v, $cert_{B_v}\rangle$, for at least $f + 1$ correct nodes where $cert_{B_v}$ is the corresponding adopt certificate. In this case, similarly, the new-view block for view $v + 1$ includes (1) a causal reference to the adopted backbone block B_v, (2) the adopt certificate $cert_{B_v}$ proving that p adopted B_v (i.e., no correct node could complete any other block), (3) the view number v, and like the previous case, some optional payload and causal references.

For the rest of the nodes, BBCA may return $\langle$BBCA-NOADOPT$\rangle$. In this case the new-view block which p broadcasts includes (1) a causal reference to the backbone block $B = lastCompleted$ of the highest view which p has completed, (2) the corresponding complete certificate $cert_B$, proving that p completed B, (3) a signed tuple $\langle \text{NOADOPT}, v\rangle_{\sigma_p}$, together with the additional information as mentioned.

The backbone block for view $v + 1$ includes (1) a causal reference to 1 or $2f + 1$ new-view blocks, depending on how the leader enters the view, (2) the new view number $v + 1$, and like new-view blocks, possibly some additional payload. In the common case, the leader includes a causal reference to a new-view block which references the backbone block of view v which the leader of view v has committed. Notice that in this case the leader can embed its own new-view block, and, hence, does not have to wait for other new-view blocks to arrive. Otherwise, the leader waits for new-view blocks of other nodes to justify what happened in view v. If the leader receives a new-view block B_n with a valid adopt certificate $cert_{B_v}$ for the backbone block B_v of view v, the leader includes a causal reference to B_n. Otherwise, the leader waits for and causally references $2f + 1$ new-view blocks from distinct nodes with a valid signature on the tuple $\langle \text{NOADOPT}, v\rangle$. These constitute a justification that no node could commit a backbone block in view v and thus it is safe to skip.

Each node maintains local state for finalized views, i.e., each view v for which the node has a valid complete certificate for a backbone block, a valid adopt certificate for a backbone block causally followed by a completed backbone block, or $2f + 1$ distinct valid signatures on $\langle \text{NOADOPT}, v\rangle$. Upon receiving a new-view block, including their own, with a causal reference to a backbone block B_v for view v and a valid complete certificate for B_v, a node p finalizes view v with B_v. Moreover, p recursively finalizes the non-finalized views in B_v's causal history as follows: a view v' is finalized with $B_{v'}$ if there exists a complete or adopt certificate for $B_{v'}$, otherwise v' is finalized with a special NO-OP value to indicate that view v' should simply be skipped.

Additionally, the node maintains a pointer for the highest committed view $lastCommitted$. Upon finalizing some view v, p commits backbone blocks for all contiguous finalized views, starting from $lastCommitted + 1$, while skipping views finalized with NO-OP. This ensures that nodes commit backbone blocks in increasing view numbers with consistent committed prefixes. Along each backbone block, p commits to the total order of all remaining causally referenced blocks including new-view blocks according to some deterministic rule. Note that by the causality of broadcast, a node that commits the backbone block already has all the blocks referenced by the backbone block available.

The BBCA-CHAIN protocol is presented as pseudocode in Figure 6.

```
Init:
    $\forall v \in \mathbb{N}, newViewBlocks[v] \leftarrow \bot$
    // $newViewBlocks[v]$ holds the set of received new-view blocks in view $v$, $newViewBlocks[v][q]$ is the block from $q$
    $\forall v \in \mathbb{N}, finalized[v] \leftarrow \bot$
    // $finalized[v]$ holds the finalized block for view $v$, a continuously committed prefix is the committed chain.
    $lastCommitted \leftarrow 0$
    $lastCompleted \leftarrow \langle B_{genesis}, cert_0\rangle$
    $newViewBlocks[0] \leftarrow newViewBlock($
        $ref = B_{genesis}, completeCert = cert_0, view = 0)$
    $view \leftarrow 0$
    $viewTimer.init()$
    // start the view advancing loop
    ENTERVIEW$(1)$

procedure ENTERVIEW$(v)$ is
    $view \leftarrow v$
    $viewTimer.restart()$
    let $proposer = $ GETPROPOSER$(view)$
    initialize BBCA instance with $bid = \langle proposer, view\rangle$
    // Nothing for non-leader to do until some event received
    if $proposer \neq p$ then return
    let $C = newViewBlocks[view - 1]$
    let $complete = \exists q : C[q].completeCert \neq \bot \wedge$
                      $C[q].ref.view = view - 1$
    let $adopt = \exists q : C[q].adoptCert \neq \bot$
    let $noadopt = \exists Q : |Q| = 2f + 1 \wedge$
            $(\forall r \in Q : C[r].sig = \langle \text{NOADOPT}, view - 1\rangle_{\sigma_r})$
    wait until $complete \vee adopt \vee noadopt = $ TRUE
    if $complete$ then
        $prevView \leftarrow \langle completed = C[q]\rangle$
    else if $adopt$ then
        $prevView \leftarrow \langle adopted = C[q]\rangle$
    else
        $prevView \leftarrow \langle noadopted = \{C[r] : \forall r \in Q\}\rangle$
```

```
    end if
    $B \leftarrow backboneBlock(ref = prevView, view = view)$
    BBCA-BROADCAST($\langle sender = p, view = view \rangle, B$)

upon BBCA-COMPLETE($bid, B, cert_B$) do
    TRYCOMMIT($B$)
    if $B.view < view$ then return
    $lastCompleted \leftarrow \langle ref = B, cert = cert_B \rangle$
    let $B_n = newViewBlock($
        $ref = B, completeCert = cert_B, view = B.view)$
    if GETPROPOSER($B.view + 1$) $= p$ then
        // $p$ is the proposer for the next view, thus
        // ENTERVIEW will embed $B_n$ in the proposal.
        $newViewBlocks[B_n.view][p] \leftarrow B_n$
    else
        // Otherwise, broadcast the new-view block.
        BROADCAST($B_n$)
    end if
    ENTERVIEW($B.view + 1$)

upon $viewTimer.elapsed > T_{max}$ do
    if $\langle$BBCA-ADOPT$, bid, B, cert_B \rangle =$ BBCA-PROBE($bid$) then
        BROADCAST($newViewBlock($
                $ref = B, adoptCert = cert_B, view = view))$
    else
        $\langle B, cert_B \rangle \leftarrow lastCompleted$
        BROADCAST($newViewBlock($
                $ref = B, completeCert = cert_B,$
                $sig = \langle$NOADOPT$, view \rangle_{\sigma_p},$
                $view = view))$
    end if
    ENTERVIEW($view + 1$)

upon receiving newViewBlock $B_n$ from $q$ do
    if $newViewBlocks[B_n.view][q] = \bot$ then
        $newViewBlocks[B_n.view][q] \leftarrow B_n$
        if $B_n.completeCert \neq \bot$ then TRYCOMMIT($B_n.ref$)
    end if

procedure TRYCOMMIT($B$) is
    FINALIZE($B$)
    while $finalized[lastCommitted + 1] \neq \bot$ do
        $lastCommitted \leftarrow lastCommitted + 1$
        if $finalized[lastCommitted] \neq$ NO-OP then
            commit $finalized[lastCommitted]$
        end if
    end while

procedure FINALIZE($B$) is
    if $finalized[B.view] \neq \bot$ then return
    $finalized[B.view] \leftarrow B$
    if $B.ref.completed \neq \bot$ then
        FINALIZE($B.ref.completed.ref$)
    else if $B.ref.adopted \neq \bot$ then
        FINALIZE($B.ref.adopted.ref$)
    else
        $B_{\max} \leftarrow \arg\max_{B_n \in B.ref.noadopted}(B_n.ref.view)$
        for $i \in [B_{\max}.ref.view + 1, B.view)$ do
            $finalized[i] \leftarrow$ NO-OP
        end for
        FINALIZE($B_{\max}.ref$)
    end if

```

Fig. 6. BBCA-Chain operational logic for node p. Block validation is omitted in the code for clarity. $backboneBlock()$ and $newViewBlock()$ construct backbone and new-view blocks respectively. During the block construction, actual payload (e.g., transactions from the mempool) and causal references to the uncommitted frontier can be added to causally commit useful transactions according to the consistent ordering on the view-by-view backbone determined by **commit**.

5 Further Discussion

Simplifying further. We can change the protocol to avoid requiring nodes to broadcast new-view blocks. Adopting the principles of HotStuff-2 [29], nodes will send new-view messages directly to the leader of the next view. The view-synchronization would be handled using some protocol outside the DAG (see the related works in Section 8).

Briefly, this modification works as follows. The leader of view v waits for the earliest of the following conditions in order to form a proposal:

- a commit certificate for B_{v-1} is received.
- an adopt certificate has been received.
- new-view messages on view $v - 1$ have been received from all nodes.
- the view $v - 1$ timer has expired, potentially with an additional time to allow one network trip.

Other nodes guard the safety of backbone proposals by comparing with their own highest certified backbone block, instead of looking at the proposal causal history. That is, the predicate $\mathcal{P}(B_v)$ nodes use before they accept a backbone block in BBCA is that the block references a backbone block (either as commit or an adopt certificate) at least as high as the node's highest adopt/commit certificate.

Briefly, after GST, in case the leader of view v does not obtain a certificate for a backbone block of view $v-1$, then it is guaranteed that no correct node holds a commit or adopt certificate for view $v-1$. Before GST, the leader might not receive in time the highest backbone block certificate in the system. In case there is any node with a higher backbone block certificate than proposed, then this node may indeed reject the leader proposal because it will not pass the $\mathcal{P}(B_v)$ predicate. This is ok, because safety is still ensured by BBCA, while liveness can only be guaranteed after GST.

Network utilization. As discussed in Section 4 backbone blocks and new-view blocks may include a payload, i.e., transactions or a bundle of certificates for available transactions, assuming a separate data dissemination and availability layer. However, BBCA-CHAIN design allows nodes to broadcast data blocks (with payload) independently in parallel, at any time, and orthogonally to view progression. Therefore, unlike protocols based on layered DAGs, nodes with high load and high bandwidth capacity can broadcast data blocks more frequently. Moreover, nodes can continue broadcasting data blocks, even when view progression is slow, in presence of a network partition or a slow/Byzantine leader. This allows nodes to stay busy continuously and truly saturate the network capacity.

View synchronization. In BFT protocols, view synchronization ensures that all correct nodes enter a new view within bounded time. There exist several recent advancements in literature which could be used by BBCA-CHAIN [9, 28]. However, we can enhance BBCA-CHAIN itself to implement view synchonization with few simple rules. In the common case, a node completes view v and enters view $v+1$ upon it BBCA-completes the backbone block of view v. Upon receiving a new-view block B_n for view $v+1$, which references a completed backbone block, the node completes view v and enters view $v+1$. Similarly, if B_n references an adopted backbone block, the node adopts the backbone block, and enters view $v+1$. Otherwise, a node that encounters the timeout probes BBCA and gets BBCA-NOADOPT, does not enter view $v+1$ unless it receives $2f+1$ new-view blocks. Additionally, a node that receives $f+1$ new-view blocks for view $v+1$ with a valid $\langle \text{NOADOPT}, v+1 \rangle$ signature, probes its BBCA instance for view v, completes view v and enters view $v+1$. In the last case, $f+1$ blocks are necessary to ensure that adversaries cannot force a correct node to complete a view.

6 BBCA Implementation Correctness

In this section we prove that the specification of BBCA in Algorithm 5 satisfies BBCA properties described in Section 3.

Lemma 1 (Validty). *If a correct sender BBCA-broadcasts a message m for instance bid, and no correct node probes bid, eventually every correct node BBCA-completes m.*

Proof. Because the sender is correct, all correct nodes broadcast an ECHO message for m. All correct nodes then receives at least $n-f=2f+1$ ECHO messages. As no correct node invokes BBCA-PROBE, all correct nodes will broadcast a READY message for m. Therefore, all correct nodes receive at least $2f+1$ READY messages and BBCA-complete m.

Lemma 2 (Consistency). *If a correct node either BBCA-adopts or BBCA-completes a message m for instance bid, and some other correct node BBCA-completes message m' for bid, then $m=m'$.*

Proof. For either BBCA-adopt or BBCA-complete case, a correct node p needs to receive at least $2f+1$ ECHO messages. Let's assume that some other node q BBCA-completes some message $m' \neq m$ in the instance of bid. Then q has also received at least $2f+1$ ECHO messages. Since $n=3f+1$, among $2(2f+1)=4f+2$ messages, there are $f+1$ sent by the same set of nodes to both p and q. Thus at least one correct node has sent to both p and q, leading to a contradiction that a correct node can only echo the same m.

Lemma 3 (Complete-Adopt). *If a message m is ever BBCA-completed by some correct node for instance bid, then at least $f+1$ correct nodes get $\langle \text{BBCA-ADOPT}, bid, m, cert_m \rangle$ for bid, if they invoke BBCA-PROBE(bid). That is, if $f+1$ correct nodes return $\langle \text{BBCA-NOADOPT}, bid \rangle$, then no correct node ever BBCA-completes any message for instance bid.*

Proof. If some correct node BBCA-completes a message m, then it has received a READY message from at least $2f+1$ nodes. Among them, at least $f+1$ are correct nodes, who have an adopt certificate (consists of $2f+1$ signed ECHO messages) for m. Therefore they will return BBCA-ADOPT upon probing.

For the second part of the lemma, assume that some correct node p BBCA-completes a message m. p has received $2f+1$ READY messages, of which $f+1$ are from the correct nodes. For the group of $f+1$ nodes which return BBCA-NOADOPT, there is at least one node that also has sent the aforementioned READY message, which is contradictory because a node cannot both send a ready message and returns BBCA-NOADOPT, in any order (in the pseudocode, probing will mark $abort \leftarrow$ TRUE, preventing the node from sending READY; and after sending READY, probing will always return the adopted message).

Lemma 4 (Integrity). *If a correct node BBCA-adopts or BBCA-completes a message m for instance bid and the sender p is correct, then p has previously invoked* BBCA-BROADCAST(bid, m).

Proof. Correct nodes only broadcast ECHO messages for an INIT message from the instance sender, so a completed message must have been initially sent in INIT message, sent by p upon the invocation of BBCA-BROADCAST.

7 BBCA-CHAIN Correctness

7.1 Safety

To prove consistency, we need to show that correct nodes commit the same blocks in the same order. It suffices to show that correct nodes commit backbone blocks in the same order, as the order of non backbone blocks is determined by the first backbone block that references them and some arbitrary deterministic rule.

Lemma 5. *There can be at most one unique B per view obtaining a complete certificate and/or an adopt certificate (for all correct nodes).*

Proof. The lemma follows directly from BBCA consistency, as in each view correct nodes initialize a single BBCA instance.

Lemma 6. *Suppose a node p commits a backbone block B_v for view v. Then there exists a view v' where $v' \geq v$, such that p has BBCA-completed a block $B_{v'}$ at view v' and such that there is a causal chain of complete certificates or adopt certificates of backbone blocks from $B_{v'}$ back to B_v.*

Proof. Nodes process commit blocks in strictly increasing views, iterating back to their last committed view. The lemma follows immediately from the code: procedure TRYCOMMIT is invoked only upon BBCA-complete, and iterates backward through blocks in the *finalized* array, each of which has a complete or an adopt certificate.

Lemma 7. *If two correct nodes p, q commit backbone blocks B_1, B_2 respectively for view v, then $B_1 = B_2$.*

Proof. For the Lemma statement to hold, p and q have finalized view v with B_1 and B_2 respectively. Therefore, both B_1 and B_2 received a complete and/or an adopt certificate for view v, and by Lemma 5 $B1 = B2$.

Lemma 8 (Consistency). *If a correct node p commits B_v for some view v, and another correct node q commits any C_w for view $w \geq v$, then q commits B_v for view v.*

Proof. Let v' be the lowest view where $v' \geq v$ for which p BBCA-completes some block $B_{v'}$ (v' exists by Lemma 6). Likewise, let w' be the lowest view where $w' \geq v$ for which q BBCA-completes some block $C_{w'}$ (likewise, w' exists). Without loss of generality, assume $w' \geq v'$.

If $w' = v'$, then and by Lemma 7, $C_{w'} = B_{v'}$. Otherwise, $w' > v'$.

When q iterates from view w' back to view v, it goes through view v'. By assumption, p BBCA-completed $B_{v'}$ and q did not obtain a complete certificate for view v'. Therefore, q either adopts $B_{v'}$ or skips view v'. However, by the Complete-Adopt property of BBCA, q cannot skip with $2f+1$ $\langle$BBCA-NOADOPT$\rangle$ for view v', since p has BBCA-completed a block in view v'.Therefore, q commits some backbone block in view v' via an adopt certificate, and by Lemma 5 this block must be $B_{v'}$.

In both scenarios, by Lemma 7, both p and q iterate from $B_{v'}$ back to B_v through an identical sequences of committed backbone blocks.

7.2 Liveness

We assume a view synchronization protocol which guarantees that, after GST, all correct nodes enter view v within bounded time Δ_{sync}. Section 7.3 shows BBCA-CHAIN can satisfy this property without incurring extra communication, using the DAG causal ordering of blocks by all nodes.

A view-timer Γ with duration Δ_Γ is picked such that after GST, if $2f+1$ correct nodes enter view v at time t then Γ suffices for the leader of view v to

(i) Enter view v,

(ii) Broadcast a backbone block via BBCA, and reach BBCA-COMPLETE at all $2f+1$ correct nodes.

I.e. $\Gamma \geq \Delta_{sync} + 3\Delta$, where Δ the maximum network delay after GST.

Lemma 9. *After GST, if the leader of view v is correct,* BBCA-COMPLETE *is reached in view v by all correct nodes.*

Proof. Let as assume that we are after GST and the first correct node p to ender view v does so at time t. By Lemma 14, all correct nodes, including the leader, enter view v the latest at time $t + \Delta_{sync}$. Therefore the leader broadcasts a block B_v the latest at time $t + \Delta_{sync}$, that will get accepted inside BBCA (we assume the leader is correct), the latest at $t + \Delta_{sync} + \Delta$ by all correct nodes. The latest at $t + \Delta_{sync} + 2\Delta$ all correct nodes will have received $2f+1$ ECHO messages and the latest at $t + \Delta_{sync} + 3\Delta$ all correct nodes will have BBCA-completed B_v, i.e, before Γ expires for p.

Lemma 10. *For every view v, a correct node eventually enters the view. Furthermore, upon entering view v, the node has received one of the following from view $v-1$: (i) a commit certificate of the backbone block of view $v-1$, (ii) an adopt certificate of the backbone block of view $v-1$, (iii) $2f+1$ signed noadopt of the backbone block of view $v-1$.*

Proof. We prove this by induction. After entering view v, each node broadcasts a new-view block for view $v-1$ either upon receiving messages from others (including possibly the backbone block for view $v-1$), or its own view $v-1$ timer expires. Since eventually all correct nodes send new-view blocks, a node p enters view v when one of the following conditions holds:

- p received $2f+1$ new-view messages for view $v-1$ carrying (signed) BBCA-NOADOPT (for view $v-1$).
- p received a commit-certificate for view $v-1$.
- p received a new-view message carrying an adopt-certificate for view $v-1$.

Lemma 11. *For every view v a correct node either commits a block, or skips the view.*

Proof. By Lemma 10, eventually every correct node enters v having received a "skip" certificate ($2f+1$ noadopt's), a commit certificate, or an adopt certificate. By Lemma 9, after GST eventually there is a view $v' \geq v$ whose leader is correct and all nodes BBCA-complete the backbone block of v'. A node then invokes TRYCOMMIT which iterates backward starting at the committed block of view v'. Starting with the block that becomes committed in view v', each committed block B encountered in the loop determines a block B_{prev} that immediately precedes B in the sequenced $backbone$:

- If $B.ref$ references a commit or an adopt certificate, then B_{prev} is the certified block.
- Otherwise, B_{prev} is the block whose view is maximal among the commit and adopt certificates which are referenced by blocks in $B.ref.noadopted$.

Then, all the views between B_{prev} and B becomes finalized as skipped; B_{prev} becomes committed and the next loop continues iterating backward from it.

The loop completes when all non-finalized views $\leq v'$ become finalized. Therefore, if view v hasn't been finalized already, it will become finalized either as skipped or committed.

Lemma 12 (Chain growth). *The chain of blocks grows infinitely, i.e., eventually, all correct nodes commit a backbone block for some view v.*

Proof. By Lemma 9 and the assumption that GETPROPOSER(v) returns a correct leader for some view v after GST, all correct nodes BBCA-complete some backbone block B_v for view v. We therefore need to show that eventually B_v will also be committed. By Lemma 11, for every view $v' < v$ either some block is committed, or the view is skipped. Therefore, v will be recursively committed.

Lemma 13 (Censorship resistance). *After GST, if a correct node broadcasts a block b, then the block will be eventually committed.*

Proof. If the block is a backbone block, then by Lemma 9, b gets BBCA-completed for some view v. By Lemma 11, for all views $v' < v$ eventually, either some block gets committed or the view is skipped and, therefore, eventually b gets committed. Otherwise the block will be committed as a causal reference of a backbone block.

7.3 View Synchronization

Lemma 14 (View Synchronization). *Let p be the first correct node enters view $v + 1$ at time t after GST. Then all correct nodes view $v + 1$ the latest at time $t + \Delta$, where Δ is the maximum network delay.*

Proof. We examine exhaustively all ways in which p may enter view $v + 1$.

1. p BBCA-completes the backbone block of view v. In this case p broadcasts a new-view block for $v + 1$, which all correct nodes receive the latest after Δ. This allows them to enter view $v + 1$ themselves.
2. p receives a new-view block for view $v + 1$ which references a completed backbone block for view v. As in the previous case, p broadcasts a new-view block for $v + 1$, and all correct nodes enter view $v + 1$ the latest after Δ.
3. p receives a new-view block for view $v + 1$ which references an adopted backbone block for view v. Again, p broadcasts a new-view block for $v + 1$, and all correct nodes enter view $v + 1$ the latest after Δ.
4. p probes BBCA (either due to a timeout, or after receiving $f + 1$ new-view messages with NOADOPT) which returns BBCA-ADOPT. In this case p broadcasts a new-view block for $v + 1$ with an adopt certificate, which all correct nodes receive the latest after Δ and enter view $v + 1$ themselves.
5. p probes BBCA (either due to a timeout, or after receiving $f + 1$ new-view messages with NOADOPT) which returns BBCA-NOADOPT. By the assumption that p is the first correct node who enters view $v + 1$, p has received $2f + 1$ new-view messages with a $\langle \text{NOADOPT}, v + 1\rangle$ signature. At least $f + 1$ of them are from correct nodes. Let's assume that the $f + 1^{th}$ correct node who sent a new-view message with $\langle \text{NOADOPT}, v + 1\rangle$ did so at time τ. Therefore, all corrects nodes receive $f + 1$ new-view messages by time at most $\tau + \Delta$ and send a new-view message themselves. Therefore, all correct nodes receive in total $2f + 1$ messages by time $\tau + 2\Delta$ and enter view $v + 1$, that is at most Δ after p.

8 Related Work

The focus of our work is solving consensus and state-machine-replication (SMR) while allowing underlying parallel communication for high throughput.

We therefore review SMR solutions along a spectrum: on one extreme are traditional leader-based SMR protocols, in the middle are solutions that leverage causality to parallelize certain aspects, and at the far extreme are leaderless/multi-leader solutions that commit proposals at bulk.

PBFT [8], a landmark in BFT solutions introduced two decades ago, emphasizes optimistically low latency. It established the view-by-view "recipe" where the leader of a new-view submits a proposal with a justification proof. This approach for justifying a new leader proposal after a view-change is the foundation of all protocols in the PBFT family, including FaB [31], Zyzzyva [26], Aardvark [10], and SBFT [22]. Notably, the last builds on the pioneering works of Cachin et. al [6] and Reiter [34], to linearize the common case message complexity by employing signature aggregation. Still, the view-change justification proof is complex to code and incurs quadratic communication complexity. Tendermint [4] introduced a simpler view-change sub-protocol than PBFT, later adopted in Casper [5]. HotStuff [43] harnesses and enhances the simple Tendermint view-change via an extra phase; several advances to HotStuff [24, 41, 20, 1, 21] eventually led to HotStuff-2 [29] which successfully removes the requirement of an extra phase.

Follow-up research works [11, 42, 23, 38, 39, 27] focused on improving throughput by enabling multiple nodes to act as leaders in parallel, and therefore, alleviate the computational and communication single-leader bottleneck of previously mentioned works. Still, those protocols require similar view-change mechanisms with their single-leader counterparts, resulting in periods with underutilized network resources. This issue was addressed by DAG-based solutions which employ causal ordering and pipelining to decouple consistent block dissemination from ordering.

One class of DAG-based algorithms are asynchronous and leaderless and include DAG-based solutions that date back to the 1990's as well as recent protocols for blockchains [15, 32, 2, 25, 14]. Recently, a class of DAG-based algorithms emerged that borrow the leader-based approach of algorithms like PBFT/HotStuff and optimize performance for partial synchrony settings. These DAG-based consensus protocols [19, 13, 25, 14, 37, 30] form a backbone sequence of backbone blocks on the DAG. However, as discussed previously, the current generation of DAG-riding solutions incurs increased latency and a complicated logic. Very recently there has been an effort to reduce latency by interleaving two instances of Bullshark on the same DAG [33, 36], resulting in one BCB latency reduction. Cordial Miners is an alternative approach to reducing latency in DAG protocols by using only causal Best-Effort Broadcast (BEB).

BBCA-CHAIN borrows the key ideas that deal with partial synchrony from the above works, modifying it to substantially reduce latency and simplify the logic. Importantly, BBCA-CHAIN gets rid of specialized layers altogether, leading to an arguably simpler implementation. It further reduces latency by integrating voting in the BBCA broadcast of a backbone block and by allowing non-backbone blocks to use BEB.

A key ingredient in DAG-based solutions is the notion of the secure, causal, and reliable broadcast, introduced by Reiter and Birman [35], and later refined by Cachin [6] and Duan et al. [16]. This primitive was utilized in a variety of BFT replicated systems, but not necessarily in the form of "zero message overhead" protocols riding on a DAG. Several of these pre-blockchain-era BFT protocols are DAG based, notably Total [32] and ToTo [15], both of which are BFT solutions for the asynchronous model.

GradedDAG [12] is a randomized consensus protocol that inspired us to abstract a core primitive whose strong guarantees simplify the DAG-level consensus logic. GradedDAG uses graded Byzantine reliable broadcast (GRBC), an BRB variant that yields graded output, to facilitate consensus, whereas BBCA-CHAIN uses BBCA with Complete-Adopt exposed from a BCB primitive whose weaker properties suffice for consensus. BBCA-CHAIN operates under the partial synchrony model whereas GradedDAG is asynchronous.

The intuition behind Complete-Adopt is a notion at the heart of consensus solutions, familiar to "Commit-Adopt" by Gafni et al. in [18]. In the context of BBCA, it stipulates that a broadcast can complete only after a quorum has locked it. Although many forms of broadcasts have been formulated, including gradecast with multiple grades of output, to our knowledge no previous broadcast primitive has been formulated with a Complete-Adopt guarantee.

Finally, we remark that data availability is an orthogonal problem which already in PBFT [8] was addressed by batching transactions hashes into blocks and form consensus on blocks. Recently, Narwhal [14] decoupled data availability from total ordering for DAG-based protocols. In parallel, DispersedLeder [42] and, later, HotShot [40] proposed a data-availability component which pre-disseminates transactions and obtains a certificate of uniqueness and availability.

References

1. M. Abspoel, T. Attema, and M. Rambaud. Malicious security comes for free in consensus with leaders. Cryptology ePrint Archive, 2020.
2. L. Baird. The swirlds hashgraph consensus algorithm: Fair, fast, Byzantine fault tolerance. Swirlds Tech Reports SWIRLDS-TR-2016-01, Tech. Rep, 34:9–11, 2016.
3. G. Bracha and S. Toueg. Asynchronous consensus and broadcast protocols. J. ACM, 32(4):824–840, 1985.
4. E. Buchman. Tendermint: Byzantine fault tolerance in the age of blockchains. PhD thesis, University of Guelph, 2016.
5. V. Buterin and V. Griffith. Casper the friendly finality gadget. arXiv preprint arXiv:1710.09437, 2017.
6. C. Cachin, K. Kursawe, F. Petzold, and V. Shoup. Secure and efficient asynchronous broadcast protocols. In Annual International Cryptology Conference, pages 524–541. Springer, 2001.
7. C. Cachin, K. Kursawe, and V. Shoup. Random oracles in constantipole: practical asynchronous byzantine agreement using cryptography. In Proceedings of the nineteenth annual ACM symposium on Principles of distributed computing, pages 123–132, 2000.
8. M. Castro and B. Liskov. Practical Byzantine fault tolerance. In M. I. Seltzer and P. J. Leach, editors, Proceedings of the Third USENIX Symposium on Operating Systems Design and Implementation (OSDI), New Orleans, Louisiana, USA, February 22-25, 1999, pages 173–186. USENIX Association, 1999.
9. P. Civit, M. A. Dzulfikar, S. Gilbert, V. Gramoli, R. Guerraoui, J. Komatovic, and M. Vidigueira. Byzantine consensus is $\theta(n^2)$: The Dolev-Reischuk bound is tight even in partial synchrony! In International Symposium on Distributed Computing, 2022.
10. A. Clement, E. Wong, L. Alvisi, M. Dahlin, and M. Marchetti. Making Byzantine fault tolerant systems tolerate Byzantine faults. In Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation, NSDI'09, page 153–168. USENIX Association, 2009.
11. T. Crain, C. Natoli, and V. Gramoli. Red belly: A secure, fair and scalable open blockchain. In 2021 IEEE Symposium on Security and Privacy (SP), pages 466–483. IEEE, 2021.
12. X. Dai, Z. Zhang, J. Xiao, J. Yue, X. Xie, and H. Jin. GradedDAG: An asynchronous DAG-based BFT consensus with lower latency. In 42nd International Symposium on Reliable Distributed Systems, SRDS 2023, Marrakech, Morocco, September 27-29, 2023. IEEE, 2023.
13. G. Danezis and D. Hrycyszyn. Blockmania: from block dags to consensus. arXiv preprint arXiv:1809.01620, 2018.
14. G. Danezis, L. Kokoris-Kogias, A. Sonnino, and A. Spiegelman. Narwhal and tusk: a dag-based mempool and efficient BFT consensus. In Y. Bromberg, A. Kermarrec, and C. Kozyrakis, editors, EuroSys '22: Seventeenth European Conference on Computer Systems, Rennes, France, April 5 - 8, 2022, pages 34–50. ACM, 2022.
15. D. Dolev, S. Kramer, and D. Malki. Early delivery totally ordered multicast in asynchronous environments. In FTCS-23 The Twenty-Third International Symposium on Fault-Tolerant Computing, pages 544–553. IEEE, 1993.
16. S. Duan, M. K. Reiter, and H. Zhang. Secure causal atomic broadcast, revisited. In 2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), pages 61–72. IEEE, 2017.
17. C. Dwork, N. A. Lynch, and L. J. Stockmeyer. Consensus in the presence of partial synchrony. J. ACM, 35(2):288–323, 1988.
18. E. Gafni. Round-by-round fault detectors (extended abstract) unifying synchrony and asynchrony. In Proceedings of the seventeenth annual ACM symposium on Principles of distributed computing, pages 143–152, 1998.
19. A. Gagol, D. Lesniak, D. Straszak, and M. Swietek. Aleph: Efficient atomic broadcast in asynchronous networks with byzantine nodes. In Proceedings of the 1st ACM Conference on Advances in Financial Technologies, AFT 2019, Zurich, Switzerland, October 21-23, 2019, pages 214–228. ACM, 2019.
20. R. Gelashvili, L. Kokoris-Kogias, A. Sonnino, A. Spiegelman, and Z. Xiang. Jolteon and Ditto: Network-adaptive efficient consensus with asynchronous fallback. In Financial Cryptography and Data Security: 26th International Conference, FC 2022, pages 296–315. Springer, 2022.
21. N. Giridharan, H. Howard, I. Abraham, N. Crooks, and A. Tomescu. No-commit proofs: Defeating livelock in BFT. Cryptology ePrint Archive, 2021.
22. G. Golan Gueta, I. Abraham, S. Grossman, D. Malkhi, B. Pinkas, M. Reiter, D. Seredinschi, O. Tamir, and A. Tomescu. SBFT: A scalable and decentralized trust infrastructure. In Proceedings - 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2019, Proceedings - 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2019, pages 568–580, 2019.
23. S. Gupta, J. Hellings, and M. Sadoghi. RCC: resilient concurrent consensus for high-throughput secure transaction processing. In 37th IEEE International Conference on Data Engineering, ICDE 2021, Chania, Greece, April 19-22, 2021, pages 1392–1403. IEEE, 2021.
24. M. M. Jalalzai, J. Niu, C. Feng, and F. Gai. Fast-HotStuff: A fast and resilient HotStuff protocol. arXiv preprint arXiv:2010.11454, 2020.

25. I. Keidar, E. Kokoris-Kogias, O. Naor, and A. Spiegelman. All you need is DAG. In A. Miller, K. Censor-Hillel, and J. H. Korhonen, editors, PODC '21: ACM Symposium on Principles of Distributed Computing, Virtual Event, Italy, July 26-30, 2021, pages 165–175. ACM, 2021.
26. R. Kotla, L. Alvisi, M. Dahlin, A. Clement, and E. Wong. Zyzzyva: Speculative Byzantine fault tolerance. In Proceedings of Twenty-First ACM SIGOPS Symposium on Operating Systems Principles, SOSP '07, page 45–58. Association for Computing Machinery, 2007.
27. K. Lev-Ari, A. Spiegelman, I. Keidar, and D. Malkhi. Fairledger: A fair blockchain protocol for financial institutions. In International Conference on Principles of Distributed Systems, 2019.
28. A. Lewis-Pye. Quadratic worst-case message complexity for state machine replication in the partial synchrony model. ArXiv, abs/2201.01107, 2022.
29. D. Malkhi and K. Nayak. Hotstuff-2: Optimal two-phase responsive bft. Cryptology ePrint Archive, 2023.
30. D. Malkhi and P. Szalachowski. Maximal extractable value (MEV) protection on a DAG. CoRR, abs/2208.00940, 2022.
31. J.-P. Martin and L. Alvisi. Fast Byzantine consensus. Dependable and Secure Computing, IEEE Transactions on, 3:202– 215, 08 2006.
32. L. E. Moser and P. M. Melliar-Smith. Byzantine-resistant total ordering algorithms. Information and Computation, 150(1):75–111, 1999.
33. Mysten Labs. Private communication, 2023.
34. M. K. Reiter. Secure agreement protocols: Reliable and atomic group multicast in Rampart. In Proceedings of the 2nd ACM Conference on Computer and Communications Security, CCS '94, page 68–80. Association for Computing Machinery, 1994.
35. M. K. Reiter and K. P. Birman. How to securely replicate services. ACM Transactions on Programming Languages and Systems (TOPLAS), 16(3):986–1009, 1994.
36. A. Spiegelman, B. Aurn, R. Gelashvili, and Z. Li. Shoal: Improving dag-bft latency and robustness. arXiv preprint arXiv:2306.03058, 2023.
37. A. Spiegelman, N. Giridharan, A. Sonnino, and L. Kokoris-Kogias. Bullshark: DAG BFT protocols made practical. In Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, pages 2705–2718, 2022.
38. C. Stathakopoulou, T. David, M. Pavlovic, and M. Vukolic. [solution] Mir-BFT: Scalable and robust BFT for decentralized networks. J. Syst. Res., 2(1), 2022.
39. C. Stathakopoulou, M. Pavlovic, and M. Vukolic. State machine replication scalability made simple. In Y. Bromberg, A. Kermarrec, and C. Kozyrakis, editors, EuroSys '22: Seventeenth European Conference on Computer Systems, Rennes, France, April 5 - 8, 2022, pages 17–33. ACM, 2022.
40. E. Systems. The espresso sequencer: HotShot consensus and tiramisu data availability. arXiv preprint arXiv:2306.03058, 2023.
41. T. D. Team. DiemBFT v4: State machine replication in the Diem blockchain. `https://developers.diem.com/docs/technical-papers/state-machine-replication-paper`, 2021.
42. L. Yang, S. J. Park, M. Alizadeh, S. Kannan, and D. Tse. DispersedLedger: High-Throughput Byzantine consensus on variable bandwidth networks. In 19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22), pages 493–512, 2022.
43. M. Yin, D. Malkhi, M. K. Reiter, G. G. Gueta, and I. Abraham. HotStuff: BFT consensus with linearity and responsiveness. In Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC '19, page 347–356. Association for Computing Machinery, 2019.