Granular Synchrony

Neil Giridharan[1]
Unversity of California, Berkeley, CA, USA

Ittai Abraham[1]
Intel Labs, Petah Tikva, Israel

Natacha Crooks
Unversity of California, Berkeley, CA, USA

Kartik Nayak
Duke University, Durham, NC, USA

Ling Ren[1]
University of Illinois Urbana-Champaign, IL, USA

Abstract

Today's mainstream network timing models for distributed computing are synchrony, partial synchrony, and asynchrony. These models are coarse-grained and often make either too strong or too weak assumptions about the network. This paper introduces a new timing model called granular synchrony that models the network as a mixture of synchronous, partially synchronous, and asynchronous communication links. The new model is not only theoretically interesting but also more representative of real-world networks. It also serves as a unifying framework where current mainstream models are its special cases. We present necessary and sufficient conditions for solving crash and Byzantine fault-tolerant consensus in granular synchrony. Interestingly, consensus among n parties can be achieved against $f \geq n/2$ crash faults or $f \geq n/3$ Byzantine faults without resorting to full synchrony.

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases Timing model, synchrony, asynchrony, consensus, blockchain, fault tolerance

Digital Object Identifier 10.4230/LIPIcs.DISC.2024.30

Funding *Ling Ren*[1]: National Science Foundation award #2143058.

1 Introduction

A fundamental aspect of any distributed computation is the *timing model*. There are three mainstream timing models: synchrony, asynchrony, and partial synchrony. Under synchrony, messages arrive before a known upper bound Δ. Under asynchrony, messages arrive in any finite amount of time. With partial synchrony [16], there is an unknown but finite Global Stabilization Time (GST), and the network is asynchronous before GST and synchronous afterwards.

The synchrony model is arguably a rosy reality: even a single message that takes longer than Δ to arrive is a violation of the synchrony model (forcing us to consider either the sender or recipient to be faulty). On the other hand, the asynchrony model is extremely pessimistic, making it challenging, or even impossible, to design protocols in it. The most well-known example may be the FLP impossibility [18], which states that any consensus protocol that can tolerate even a single crash fault in asynchrony must have an infinite execution. This implies that deterministic consensus in asynchrony is impossible. The partial

[1] This work was started while authors were at VMware Research.

synchrony model tries to balance asynchrony and synchrony and has been the most widely adopted in practice so far. But it is close to asynchrony in essence and shares the same fault tolerance bounds as (randomized) asynchronous protocols.

This paper argues that the current characterization of network timings is too coarse-grained. We recognize the variability and heterogeneity of modern networks and propose that they should be modeled in a *granular manner* via a graph consisting of a mixture of synchronous, partially synchronous, and asynchronous links. We call the new model *granular synchrony*.

Our new model is more than yet another theoretical construct. It is rooted in and motivated by our understanding and characterizations of modern distributed systems and networks. Modern distributed systems increasingly span datacenters, be it for disaster recovery or fault isolation [32, 6, 28]. Within datacenters, networks are mostly synchronous [35]. Spikes in message delays do occur [3], but such spikes are rare and almost never happen to the entire datacenter [21]. Across datacenters and over the Internet, networks are mostly well-behaved but are susceptible to significant fluctuations [22] and adversarial attacks [14].

The granular synchrony timing model can serve as a unifying framework for network timing models. Synchrony, partial synchrony, and asynchrony are all extreme cases of it. Outside these extreme cases, the granular synchrony model is a natural intermediate between synchrony and partial synchrony (or asynchrony) and gives rise to new results that can be construed as an intermediate between fundamental results in distributed computing.

For concreteness, we focus on the problem of fault-tolerant consensus [27] in this paper. It is well-known that under synchrony, the agreement variant of consensus can be solved in the presence of $f < n$ crash faults or $f < n/2$ Byzantine faults (assuming digital signatures). With partial synchrony, fewer faults can be tolerated: $f < n/2$ crash faults or $f < n/3$ Byzantine faults [16]. Asynchrony has the same fault thresholds and further requires the use of randomization [18].

We derive necessary and sufficient conditions for solving crash fault-tolerant (CFT) and Byzantine fault-tolerant (BFT) consensus in granular synchrony. A key benefit and interesting implication of the granular synchrony model is that we do *not* have to assume full synchrony to tolerate $f \geq n/2$ crash faults or $f \geq n/3$ Byzantine faults. Instead, consensus can be reached if and only if the underlying communication graph satisfies certain conditions.

We remark that all our protocols are *graph-agnostic*, meaning they do not need to know the synchronicity property of any link. As a result, our protocols can work in the following alternative formulation of the granular synchrony model. The consensus algorithm is parameterized by n and f. Initially, all communication links are synchronous. The adversary has the power to corrupt f nodes and alter some links to be partially synchronous or asynchronous but must not violate the necessary condition for the given n and f. On the other hand, most of our impossibility proofs rule out algorithms that know the graph and are tailored for the graph. This strengthens both our protocols and our impossibility results.

We will consider two variants of the granular synchrony model. The first variant only has synchronous and partially synchronous links (no asynchronous links), and we refer to it as *granular partial synchrony*. CFT consensus in granular partial synchrony can be solved if and only if any quorum of $n - f$ nodes collectively can communicate synchronously with at least $f + 1$ nodes despite faulty nodes. BFT consensus in granular partial synchrony can be solved if and only if any set of $n - 2f$ correct nodes can communicate synchronously with at least $f + 1$ correct nodes despite faulty nodes.

The second variant further allows asynchronous links, and we refer to it as *granular asynchrony*. For CFT consensus to be solved deterministically in granular asynchrony, it is additionally required that after removing all asynchronous edges and all crashed nodes,

less than $n - f$ nodes are outside the largest connected component of the remaining graph. For undirected graphs, this condition is weaker than the correct $\diamond f$-source condition in [4] (see §B) and establishes the minimum synchrony condition needed to circumvent the FLP impossibility [18]. For BFT consensus to be solved deterministically in granular asynchrony by a *graph-agnostic* algorithm, it is additionally required that there is a correct node with partially synchronous paths to at least f other correct nodes. The necessary and sufficient condition for algorithms that know the graph is still open.

2 Model and Definitions

We assume communication links are bi-directional. In granular partial synchrony, each link can be either synchronous or partially synchronous. In granular asynchrony, each link can be synchronous, partially synchronous, or asynchronous. A synchronous link delivers each message sent on the link within a known upper bound Δ. A partially synchronous link respects the Δ message delivery bound after GST. An asynchronous link has no delay bound and just has to deliver each message eventually. We assume all communication links are reliable and FIFO (first-in-first-out), and deliver each transmitted message exactly once.

Beyond this, the model is the same as traditional consensus literature. There are n nodes in total. The adversary can corrupt up to f nodes and can do so at any time during the protocol execution (i.e., the adversary is adaptive). In the CFT case, faulty nodes can fail by crashing only. In the BFT case, faulty nodes can behave arbitrarily and can be coordinated by the adversary. For BFT, we further assume the existence of digital signatures and public-key infrastructure (PKI) and that faulty nodes cannot break cryptographic primitives. A message is only considered valid by correct nodes if its accompanying signature is verified (we omit writing these signature operations in the protocols).

Our protocols do not require any form of clock synchronization among nodes, and instead just require bounded clock skews. To elaborate, certain steps of our protocols require nodes to wait for some amount of time (e.g., 4Δ). For simplicity, our protocol description assumes each node will wait for precisely that amount of time. But it is not hard to see that our protocols still work if each node waits for a time that falls in a known bounded range (e.g., between 4Δ and 5Δ), which is easy to achieve with bounded clock skews.

It is convenient to describe the network as an undirected graph $G = (V, E)$. Each vertex represents a node, and each edge represents a communication link. We use vertex and node interchangeably, and edge and link interchangeably. Our protocols are graph agnostic: they do not assume knowledge of the graph.

▶ **Definition 1** (Synchronous path)**.** *Node a has a synchronous path to node b, written as $a \to b$, if there exist a sequence of synchronous edges $(a, i_1), (i_1, i_2), , \ldots, (i_k, b)$ where every intermediate node i_j is correct.*

Note that in the above definition, only intermediate nodes need to be correct. Therefore, every node, even a faulty one, has a synchronous path to itself, i.e., $a \to a, \forall a \in V$. We generalize the notion of synchronous paths from two nodes to two sets of nodes A and B.

▶ **Definition 2.** *$A \to B$ if $\forall b \in B, \exists a \in A$ such that $a \to b$.*

▶ **Definition 3** (Path length, distance and diameter)**.** *The length of a path is the number of edges in it. If $a \to b$, the synchronous distance between these two nodes is the length of the shortest synchronous path between them. The synchronous diameter of a graph G is*

$$d(G) := \max_{F,a,b \text{ s.t. } |F| \le f,\ a \to b} d(a, b).$$

Partially synchronous path, path length, distance, and diameter $d'(G)$ are similarly defined. Note that a partially synchronous path can contain synchronous edges.

The (partially) synchronous distance is only defined for a pair of nodes that have a (partially) synchronous path between them. We also remark that for the Byzantine case, distance is only defined for a pair of correct nodes. The max in the diameter definition is taken over all pairs with the corresponding distance defined. The two diameters capture the worst-case round-trip delays among nodes connected by synchronous and partially synchronous paths, respectively. If $d(G)$ or $d'(G)$ is known, they can be directly used in our protocols; otherwise, $|V| - 1$ is a trivial upper bound. We will simply write d and d' when there is no ambiguity.

▶ **Definition 4** (Consensus). *In a consensus protocol, every node has an initial input value and must decide a value that satisfies the following properties.*

- *Agreement: No two correct nodes decide different values.*[2]
- *Termination: Every correct node eventually decides.*
- *Validity: If all nodes have the same input value, then that is the decision value.*

3 CFT Consensus in Granular Partial Synchrony

▶ **Theorem 5.** *Under granular partial synchrony, CFT consensus on a graph $G = (V, E)$ is solvable if and only if, regardless of which up to f nodes are faulty, $\forall A \subseteq V$ with $|A| \geq n - f$, $\exists B \subseteq V$ with $|B| \geq f + 1$ such that $A \to B$.*

In words, the condition is that any set A of size at least $n - f$ has a potentially larger set B of size at least $f + 1$, such that for any node $b \in B$ there exits $a \in A$ and a synchronous path from a to b. Intuitively, if a message arrives at all of A, then it will arrive at all of B after some delay.

It is worth noting that classic crash fault tolerance bounds are special cases of our theorem. For example, when all links are synchronous, any node has synchronous paths to all n nodes. Thus, synchronous CFT consensus can be solved for any $n \geq f + 1$. At the other extreme, $n = 2f + 1$ is the smallest value of n for which the condition in Theorem 5 trivially holds even when all edges are partially synchronous (see necessity proof). The more interesting part of our theorem is of course when we have a mix of synchronous and partially synchronous edges. Figure 1 gives examples of these intermediate cases where CFT consensus is solvable with $f + 1 < n \leq 2f$.

3.1 Necessity

We first prove the "only if" part of Theorem 5. The proof is similar to the DLS proof in partial synchrony [16]. To ensure agreement, we must ensure that nodes cannot be partitioned into two disjoint groups with no synchronous inter-group links. The condition in Theorem 5 ensures exactly that.

Proof. For $n \geq 2f + 1$, the "only if" part of the theorem is vacuous because the condition trivially holds: $n - f \geq f + 1$, and every node has a synchronous path to itself.

For $n \leq 2f$, we prove by contradiction. Suppose there is an algorithm that solves consensus on a graph G that does not satisfy the condition in the theorem. Then, there exists a set F of up to f nodes such that, if nodes in F crash, there exists a set A of at least

[2] For CFT consensus, we actually achieve the stronger property of uniform agreement, which states that no two nodes (even faulty ones) decide differently.

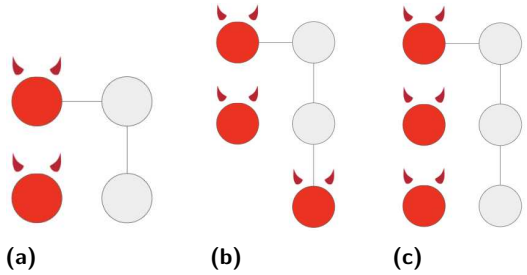

Figure 1 Only synchronous links are shown in the figure for brevity. Faulty nodes are denoted in red with horns, and the correct nodes are denoted in gray. The figure shows the necessary and sufficient condition in theorem 5 being satisfied for **(a)** $n = 4$, $f = 2$, **(b)** $n = 5$, $f = 3$, and **(c)** $n = 6$, $f = 3$.

$n - f$ nodes, which collectively have synchronous paths to at most f nodes. Let B be the set of these f nodes excluding A. Let C be the remaining nodes, i.e., $C = [n] \setminus \{A \cup B\}$. Note that $\{A, B, C\}$ is a three-way disjoint partition of the n nodes. Also note that $|A \cup B| \leq f$ and $|B \cup C| = n - |A| \leq f$. Next, we consider three executions.

In execution 1, all nodes have input v_1 and nodes in $B \cup C$ crash at the beginning. Since $|B \cup C| \leq f$, A eventually decides v_1 in time t_1 due to validity. In execution 2, all nodes have input $v_2 \neq v_1$ and nodes in $A \cup B$ crash at the beginning. Since $|A \cup B| \leq f$, C eventually decides v_2 in time t_2 due to validity.

In execution 3, nodes in A have input v_1, nodes in C have input v_2, nodes in B crash at the beginning, and $GST > \max(t_1, t_2)$. Note that crashing B (instead of F) does not change the fact that A has synchronous paths to $A \cup B$ only. This is because, with B crashed, nodes in $F \setminus B$ do not have synchronous paths to A themselves (otherwise, they would have synchronous paths to A with F crashed). Thus, synchronous paths from A to C cannot go through $F \setminus B$. Because there are no synchronous edges between A and C, the adversary can delay the delivery of all messages between A and C until after GST. Thus, A cannot distinguish execution 3 from execution 1 and C cannot distinguish execution 3 from execution 2. Then, A decides v_1 and C decides v_2, violating agreement. ◀

3.2 Protocol

Next, we present a new CFT consensus protocol assuming the condition in theorem 5 holds. This establishes the sufficiency of the condition.

Overview. A natural starting point is a standard quorum-based partially synchronous CFT consensus protocol. Such protocols require $n > 2f$ to ensure any two quorums of size $n - f$ intersect. When $n \leq 2f$, two quorums of $n - f$ may not intersect. But when the condition in theorem 5 holds, a quorum of $n - f$ nodes can hear from $f + 1$ nodes of any critical information in bounded time. This effectively promotes a quorum of size $n - f$ to $f + 1$ and ensures safety as a quorum of size $f + 1$ always intersects a quorum of size $n - f$.

Similar to other leader-based partially synchronous consensus protocols, our protocol operates in a series of views, where each view has a leader. The leader of view v is denoted as L_v. Leaders can be elected using a simple round-robin order. If a view after GST has a correct leader, nodes will commit that leader's proposal and terminate. There is a view change procedure to replace a leader who is not making progress. We focus on a single-shot consensus here, but the protocol can be easily adapted to the multi-shot setting.

Algorithm 1 CFT consensus protocol in granular partial synchrony for node i.

1: $v_i \leftarrow 0$ ▷ Initialize local view number
2: $lock \leftarrow (0, input_i)$ ▷ Initially lock on the input value
3: **enter** view 1

4: **upon** entering view v **do**
5: $v_i \leftarrow v$
6: **start** $view\_timer \leftarrow timer(4\Delta)$ ▷ Timer for changing view
7: **send** $\langle$STATUS, $v, lock\rangle$ to L_v

8: **upon** receiving $n - f$ $\langle$STATUS, $v_i, -\rangle$ and $i = L_{v_i}$ **do**
9: $val \leftarrow$ value from the highest lock (by view) received
10: **send** $\langle$PROPOSE, $v_i, val\rangle$ to all ▷ Leader proposal

11: **upon** receiving $\langle$PROPOSE, $v_i, val\rangle$ **do**
12: $lock \leftarrow (v_i, val)$
13: **send** $\langle$VOTE, $v_i, val\rangle$ to all

14: **upon** receiving $n - f$ $\langle$VOTE, $v_i, val\rangle$ or $\langle$COMMIT, $val\rangle$ **do**
15: **send** $\langle$COMMIT, $val\rangle$ to all
16: **commit** val and **terminate**

17: **upon** $view\_timer$ expiring **do**
18: **send** $\langle$NEWVIEW, $v_i + 1\rangle$ to all

19: **upon** receiving $\langle$NEWVIEW, $v\rangle$ where $v > v_i$ **do**
20: **echo** $\langle$NEWVIEW, $v\rangle$ and to all
21: **send** $\langle$LOCKED, $lock\rangle$ to all
22: **stop** accepting PROPOSE messages in views up to $v - 1$
23: **wait** $2d\Delta$ time
24: **enter** view v

25: **upon** receiving $\langle$LOCKED, $lock'\rangle$ **do**
26: $lock \leftarrow$ higher lock (by view) between $lock$ and $lock'$
27: **echo** $\langle$LOCKED, $lock'\rangle$ to all

Locks. A $lock := (view, value)$ consists of a view and value. Initially, each node locks on its input value with view number 0. When a node receives a proposal from the leader of the current view, it updates its lock to the current view and the proposed value. Locks are ranked by view numbers. Note that except for the initial view 0, there cannot be two locks with the same view number but different values, since only one value is proposed per view. Locks from view 0 can be ranked arbitrarily.

We describe the protocol next.

Status step. Each view begins with every node sending a STATUS message to the leader of the current view. A node also starts a timer for the view.

Leader proposal step. When L_v is in view v and receives $n - f$ $\langle$STATUS, $v, -\rangle$ messages, it proposes the highest locked value among those. Note that L_v only sends one PROPOSE message in a view. When a node is in view v and receives a $\langle$PROPOSE, $v, val\rangle$ message, it updates its lock to (v, val) and sends a $\langle$VOTE, $v, val\rangle$ message to all nodes.

Commit step. When a node receives a quorum of $n-f$ $\langle$Vote, $v, val\rangle$ messages or a single $\langle$Commit, $val\rangle$ message, it commits val, sends a $\langle$Commit, $val\rangle$ message to all nodes, and terminates.

View change step. When a node times out in a view v without committing a value, it sends $\langle$NewView, $v+1\rangle$ to all nodes, asking them to move to the next view. Upon receiving $\langle$NewView, $v\rangle$ for a higher view v, a node echoes $\langle$NewView, $v\rangle$ and its own lock to all nodes, waits for $2d\Delta$ time, and then enters view v. During this waiting period, the node will not send Vote for its current view but will listen for Locked messages to update its lock and also echo locks. The $2d\Delta$ time accounts for the worst-case round-trip delay to send a NewView message and receive the Locked message.

3.3 Analysis

▶ **Lemma 6.** *If some node commits val in view v, then any* $\langle$Propose, $v', val'\rangle$ *message in view* $v' \geq v$ *must have* $val' = val$.

Proof. We prove this lemma by induction on view v'. The base case of $v' = v$ is straightforward since each leader proposes only one value, so $val' = val$.

For the inductive step, suppose the lemma holds up to view $v'-1$, and we consider view v'. Suppose for the sake of contradiction that some node commits val in view v, and there is a $\langle$Propose, $v', val'\rangle$ message from $L_{v'}$ for $val' \neq val$. $L_{v'}$ must have received $\langle$Status, $v', -\rangle$ messages from a set P of $n-f$ nodes. By the condition in theorem 5, $P \to Q$, where Q is a set of $f+1$ nodes. Since a node committed val in view v, there must exist a set R of $n-f$ nodes that sent $\langle$Vote, $v, val\rangle$ messages and updated $lock := (v, val)$ in view v. Sets Q and R intersect in at least one node. Let this node be q.

Since the graph is undirected, there must exist a node $p \in P$ such that $q \to p$. By the induction hypothesis, Propose messages from view v to $v'-1$ must be for val. Since a node only updates its lock monotonically based on view numbers, node q must have a lock with view $\geq v$ for val. Let t_p be the time node p echoed $\langle$NewView, $v'\rangle$. By time $t_p + d\Delta$, node q receives $\langle$NewView, $v'\rangle$. Upon receiving $\langle$NewView, $v'\rangle$, node q sends a $\langle$Locked, $lock\rangle$ message to all nodes. This lock is received by node p by time $t_p + 2d\Delta$. Node p updates its lock to view $\geq v$ for val before entering view v'. Thus, L_v receives at least one Status message for val with view $\geq v$ and propose val, a contradiction. ◀

▶ **Theorem 7** (Agreement). *No two nodes commit different values.*

Proof. Let v be the smallest view in which a node commits some value, say val. Since only val can be proposed in view v and all subsequent views by lemma 6, no node can commit a different value. ◀

▶ **Theorem 8** (Termination). *All correct nodes eventually decide.*

Proof. With round-robin leader election, correct nodes are elected leaders infinitely often. Thus, there must be a view v, after $GST + 2d\Delta$, whose leader is correct. We next prove that all nodes will decide and terminate in view v (if they don't decide earlier).

Let t ($t \geq GST + 2d\Delta$) be the first time some correct node enters view v. This correct node sends $\langle$NewView, $v\rangle$ to all nodes at $t - 2d\Delta \geq GST$. All correct nodes receive $\langle$NewView, $v\rangle$ by time $t - 2d\Delta + \Delta$, wait $2d\Delta$ themselves, and enter view v by time $t + \Delta$. Upon entering view v, they send $\langle$Status, $v, -\rangle$ messages to L_v. L_v receives $n-f$ $\langle$Status, $v, -\rangle$ messages by time $t+2\Delta$, and sends a $\langle$Propose, $v, -\rangle$ message to all nodes. All correct

nodes receive the $\langle \text{PROPOSE}, v, - \rangle$ message and send $\langle \text{VOTE}, v, - \rangle$ messages by time $t + 3\Delta$. All correct nodes receive $n - f$ $\langle \text{VOTE}, v, - \rangle$ messages and commit by time $t + 4\Delta$. Since a node's view timer is 4Δ, all correct nodes commit and terminate in view v. ◀

▶ **Theorem 9** (Validity). *If all nodes have the same input val, then all correct nodes eventually decide val.*

Proof. If all nodes have the same input val, all nodes set $lock \leftarrow (0, val)$. Following a similar proof as in lemma 6, no other value can be proposed in all subsequent views. Validity follows from termination. ◀

4 CFT Consensus in Granular Asynchrony

▶ **Theorem 10.** *Under granular asynchrony, CFT consensus on a graph $G = (V, E)$ can be solved deterministically if and only if, (i) the condition in theorem 5 holds and (ii) for all F with $|F| \leq f$, less than $n - f$ nodes are outside the largest connected component of $G' = (V - F, \diamond E)$ where $\diamond E$ is the set of synchronous and partially synchronous edges.*

In other words, condition (ii) says that if we remove all asynchronous edges and all faulty nodes from G and further remove the largest connected component in the remaining graph, then there are fewer than $n - f$ nodes left.

4.1 Necessity

Proof. Condition (i) is already proved to be necessary in theorem 5. We focus on condition (ii). Suppose for the sake of contradiction there exists a deterministic algorithm $\mathcal{A}$ that solves CFT consensus on a graph G that violates condition (ii). This means there exists a set F with $|F| \leq f$ such that removing the largest connected component from $G' = (V - F, \diamond E)$ (G with F and all asynchronous edges removed) leaves $\geq n - f$ nodes.

Suppose the graph G' has q connected components. Clearly, $q > 1$. Let C_i be i-th connected component in G'. We have $|F \cup C_i| \leq f$ for all i because even the largest connected component plus F has at most f nodes.

We construct an external system consisting of q nodes connected only by asynchronous links. We can convert $\mathcal{A}$ into a deterministic algorithm that solves consensus in this external system while tolerating one crash fault. To do so, let the i-th node in the external system, q_i, simulate the nodes in C_i in $\mathcal{A}$. If q_i has input v_i, then all nodes in C_i have input v_i in the simulation.

An execution in this external system with q_i crashing at time t faithfully simulates an execution of $\mathcal{A}$ with F crashing in the beginning and C_i crashing at time t. In particular, observe that two connected components in G' only have asynchronous edges between them once nodes in F crash. Since $|F \cup C_i| \leq f$ for all i, $\mathcal{A}$ solves consensus in the original system. Thus, the simulated algorithm solves consensus deterministically in the external system while tolerating one crash fault in asynchrony. This contradicts the FLP impossibility [18]. ◀

4.2 Protocol

Next, we adapt our previous CFT consensus protocol in algorithm 1 from granular partial synchrony to granular asynchrony, assuming the condition in theorem 10 holds. This establishes the sufficiency of the condition.

Our prior CFT consensus protocol still maintains safety under granular asynchrony, but liveness no longer holds because there is no time when all edges behave synchronously (asynchronous links do not have a GST assumption). As a result, correct leaders in our prior

Algorithm 2 CFT consensus protocol in granular asynchrony for node i.

$v_i \leftarrow 0$ ▷ Initialize local view number
$lock \leftarrow (0, input_i)$ ▷ Initially lock on the input value
enter view 1

upon entering view v **do**
    $v_i \leftarrow v$
    send $\langle$Status, $v, lock\rangle$ to all

upon receiving $n-f$ $\langle$Status, $v_i, -\rangle$ where $i \neq L_{v_i}$ **do**
    echo these $n-f$ $\langle$Status, $v_i, -\rangle$ to all
    start *proposal_timer* $\leftarrow$ *timer*$(3d'\Delta)$

upon receiving $\langle$Propose, $v_i, val\rangle$ **do**
    $lock \leftarrow (v_i, val)$
    echo $\langle$Propose, $v_i, val\rangle$ to all
    send $\langle$Vote, $v_i, val\rangle$ to all

upon *proposal_timer* expiring and no leader proposal received **do**
    send $\langle$ViewChange, $v_i\rangle$ to all

upon receiving $n-f$ $\langle$ViewChange, $v\rangle$ **do**
    send $\langle$NewView, $v+1\rangle$ to all

Vote, Commit, Locked, NewView messages at all nodes and Status messages at view leaders are processed the same way as in Algorithm 1

protocol may continuously time out. Luckily, condition (ii) in theorem 10 can be leveraged to guarantee that when the set F of crashed nodes stops growing, and a correct node in the largest connected component of $G' = (V - F, \diamond E)$ is elected leader after GST, this leader will not be replaced and will make progress. To do so, we first require $n - f$ nodes to initiate a view change. This way, because all nodes in F are crashed and fewer than $n - f$ nodes are outside the largest connected component of $G' = (V - F, \diamond E)$, we just need to make sure that no node in this largest connected component initiates a view change. This technique is similar to those used in view synchronizes [11, 10] to make sure correct nodes eventually overlap and remain in the same view to ensure termination.

We only describe the status and view change steps since the rest of the protocol remains the same as algorithm 1.

Status and propose step. Upon entering a new view v, a node sends a $\langle$Status, $v, lock\rangle$ message to **all** nodes. When a node receives at least $n - f$ $\langle$Status, $v, -\rangle$ messages, it forwards this set of Status messages to all nodes and starts a timer of $3d'\Delta$ duration. Upon receiving a proposal, a node forwards the proposal to all nodes, in addition to locking on and voting for the proposal. The same vote and commit steps from algorithm 1 follow.

View change. A node suspects the leader is faulty if it does not receive a $\langle$Propose, $v, -\rangle$ message before its timer expires. When this occurs, a node sends a $\langle$ViewChange, $v\rangle$ message to all nodes, indicating it wishes to quit view v. When a node receives $n - f$ $\langle$ViewChange, $v\rangle$ messages for the current view v, it sends a $\langle$NewView, $v + 1\rangle$ message to all nodes. Upon receiving a NewView message, a node carries out the same new view step from algorithm 1.

4.3 Analysis

The agreement and validity proofs are identical to the granular partial synchrony CFT case. We focus on termination.

▶ **Lemma 11.** *If no correct node ever terminates, then every correct node keeps entering higher views.*

Proof. Suppose for the sake of contradiction, there exists a correct node n_1, which never enters a higher view. Let v be the view n_1 is in. If any correct node ever enters a view higher than v, it sends a NewView message for that higher view to all nodes. n_1 will eventually receive this higher NewView message and enter a higher view, a contradiction. Thus, no node ever enters a view higher than v. Before entering view v, n_1 has sent $\langle \text{NewView}, v\rangle$ to all nodes. All correct nodes will eventually receive this $\langle \text{NewView}, v\rangle$ message, enter view v, and send $\langle \text{Status}, v, -\rangle$ messages. Eventually, correct nodes will receive $n-f$ $\langle \text{Status}, v, -\rangle$ messages and start their proposal timers. If n_1 receives $n-f$ $\langle \text{ViewChange}, v\rangle$ messages, it will enter view $v+1$, a contradiction. Thus n_1 never receives $n-f$ $\langle \text{ViewChange}, v\rangle$ messages. Then, there must be at least one correct node that never sends $\langle \text{ViewChange}, v\rangle$ and instead echoes $\langle \text{Propose}, v, -\rangle$ to all nodes. Eventually, all correct nodes will receive $\langle \text{Propose}, v, -\rangle$ message and send $\langle \text{Vote}, v, -\rangle$ messages to all nodes. Eventually n_1 will receive $n-f$ $\langle \text{Vote}, v, -\rangle$ messages and terminate, a contradiction. ◀

▶ **Theorem 12.** *All correct nodes eventually terminate.*

Proof. Suppose for the sake of contradiction that some correct node never terminates. Observe that if one correct node terminates, it sends a Commit message and makes all correct nodes eventually terminate. Thus, no correct node ever terminates. By lemma 11, every correct node keeps entering higher views.

Eventually, there will be a first time after $GST + 2d\Delta$ that some correct node enters a view v such that (i) the set F of crashed nodes no longer grows in views $\geq v$, (ii) $L_v \notin F$, and (iii) L_v is in the largest connected component $G' = (V - F, \diamond E)$. Let C denote this largest connected component. We next prove no node in C will ever send $\langle \text{ViewChange}, v\rangle$.

Let p be the first node in C that enters view v, and let p enter view v at time $t > GST + 2d\Delta$. Observe that no node in C will send $\langle \text{ViewChange}, v\rangle$ before time $t + 3d'\Delta$ (proposal timer duration is $3d'\Delta$). Nodes in F crashed before entering view v and cannot send $\langle \text{ViewChange}, v\rangle$. Due to the condition in theorem 10, $n - |C \cup F| < n - f$. Thus, there will not be $n-f$ $\langle \text{ViewChange}, v\rangle$ messages before $t + 3d'\Delta$.

p sends $\langle \text{NewView}, v\rangle$ at time $t - 2d\Delta > GST$. All nodes in C receive $\langle \text{NewView}, v\rangle$ by $t - 2d\Delta + d'\Delta$, enter view v by $t + d'\Delta$, and stay in view v at least until $t + 3d'\Delta$.

When a node $q \in C$ receives $n-f$ $\langle \text{Status}, v, -\rangle$ messages at time $t' > t$, q echoes these $n-f$ messages and starts its proposal timer. All nodes in C enter view v by time $t + d'\Delta$ and are ready to echo these $\langle \text{Status}, v, -\rangle$ messages. (Recall that d' is the partially synchronous diameter of the graph.) L_v, which is in C, receives these $n-f$ $\langle \text{Status}, v, -\rangle$ messages by time $\max(t + 2d'\Delta, t' + d'\Delta) < t' + 2d'\Delta$. L_v sends a $\langle \text{Propose}, v, -\rangle$ message by time $t' + 2d'\Delta$ and it reaches q by time $t' + 3d'\Delta$, which is before q's proposal timer expires. Thus, q does not send $\langle \text{ViewChange}, v\rangle$. This establishes that no node in C will ever send $\langle \text{ViewChange}, v\rangle$. Again, nodes in F never send $\langle \text{ViewChange}, v\rangle$. Since $n - |C \cup F| < n - f$, there will never be $n-f$ $\langle \text{ViewChange}, v\rangle$ messages. Thus, no correct node ever enters a view higher than v. This contradicts lemma 11. ◀

5 BFT Consensus in Granular Partial Synchrony

▶ **Theorem 13.** *Under granular partial synchrony, BFT consensus with $n \geq 2f+1$ on a graph G is solvable if and only if, for any set F of at most f faulty nodes, $\forall A \subseteq V - F$ with $|A| \geq n - 2f$, $\exists B \subseteq V - F$ with $|B| \geq f + 1$ such that $A \to B$.*

In words, the condition is that any honest set A of size at least $n - 2f$ has a potentially larger honest set B of size at least $f + 1$, such that for any node $b \in B$ there exits $a \in A$ and a synchronous path from a to b. Intuitively, if a message arrives at all of A, then it will also arrive at all of B after some delay.

Note that in BFT consensus, it never hurts the adversary to corrupt the maximum number of nodes allowed since Byzantine nodes can actively participate. This is why we can focus on the case of $|F| = f$ (as opposed to $|F| \leq f$).

Observe that the classic Byzantine fault tolerance bounds are special cases of our theorem. For example, when $n = 2f + 1$ and all links are synchronous, any $n - 2f = 1$ correct node has synchronous paths to all $n - f = f + 1$ correct nodes, so consensus is solvable. At the other extreme, $n = 3f + 1$ is the smallest value of n for which the condition in theorem 13 trivially holds even when all edges are partially synchronous (see necessity proof). And again, we will focus on the more interesting region of $2f + 1 < n \leq 3f$.

5.1 Necessary

The proof is again very similar to DLS [16]. The essence of the condition (and the proof) is to prevent a "split-brain" attack in which two groups of $n - 2f$ correct nodes cannot communicate in time and separately make progress with f Byzantine nodes.

Proof of Theorem 13 necessity part. For $n \geq 3f + 1$, the theorem is vacuous because the condition trivially holds: any set of $n - 2f \geq f + 1$ correct nodes have synchronous paths to at least $f + 1$ correct nodes (i.e., themselves).

For $n \leq 3f$, we prove by contradiction. Suppose there is an algorithm that solves consensus on a graph G that does not satisfy the condition in the theorem. Then, there exists a set F of f nodes such that, if nodes in F are faulty, a set A of $n - 2f$ correct nodes collectively have synchronous paths to at most f correct nodes. Let B be the set of these f nodes excluding A. Let C be the remaining nodes, i.e., $C = [n] \setminus \{F \cup A \cup B\}$. Note that $\{A, B, F, C\}$ is a four-way disjoint partition of the n nodes. Also note that $n - 2f = |A| \leq |A \cup B| \leq f$, $|F| = f$, and $|C| = n - |F \cup A \cup B| \leq f$.

Next, we consider three executions. In execution 1, all nodes have input v_1, and nodes in C are Byzantine. Since $|C| \leq f$, $A \cup B$ eventually decide v_1 in time t_1 due to validity. In execution 2, all nodes have input v_2, and nodes in $A \cup B$ are Byzantine. Since $|A \cup B| \leq f$, C eventually decide v_2 in time t_2 due to validity.

In execution 3, nodes in $A \cup B$ have input v_1, nodes in C have input v_2, nodes in F are Byzantine, and $GST > \max(t_1, t_2)$. F will behave towards $A \cup B$ like in execution 1 and towards C like in execution 2. Because there is no synchronous link between $A \cup B$ and C, $A \cup B$ cannot distinguish execution 3 from execution 1 and C cannot distinguish execution 3 from execution 2. Thus, $A \cup B$ decides v_1 and C decides v_2, violating agreement. ◀

5.2 Protocol

Next, we give a new BFT consensus protocol assuming the condition in theorem 13 holds. The protocol we present here achieves external validity [12]. In appendix C, we show how to extend it to achieve the strong unanimity validity in definition 4. This establishes the sufficiency of the condition.

Like in the CFT case, we will start from a standard leader-based partially synchronous BFT protocol and then take advantage of our graph condition to upgrade a quorum of $n-2f$ correct nodes to $f+1$ correct nodes.

A *lock* is a set L of $n-f$ signed matching $\langle$Vote-1, $view, val\rangle$ messages from distinct nodes. Locks are ranked by their view numbers. We describe the protocol next.

Status step. Each view begins with every node sending a Status message to the leader of the current view. A node also starts a timer for the view.

Leader proposal step. When the leader of view v, L_v, receives a set S of $n-f$ $\langle$Status, $v, -\rangle$ messages from distinct nodes, it picks the highest-ranked lock among those. If no lock is reported, then the leader can safely propose its own input value, val_i. Otherwise, the leader must propose the value in the highest-ranked lock. The leader sends $\langle$Propose, $v, val, S\rangle$ to all nodes. Note that a correct leader only sends one Propose message in a view.

Equivocation check step. When a node receives $\langle$Propose, $v, val, S\rangle$, it checks whether val is the highest-ranked locked value from the set S. If so, it forwards the Propose message to all nodes and starts a timer for $d\Delta$ to listen for conflicting Propose messages in the same view. If it receives a conflicting Propose message, it detects the leader is faulty, forwards the equivocation to all nodes, and sends a ViewChange message for the current view. If the timer expires and no conflicting Propose message is received, the node will send a $\langle$Vote-1, $v, val\rangle$ message to all nodes indicating its support for the leader's proposal.

Locking step. When a node receives $n-f$ $\langle$Vote-1, $v, val\rangle$ messages, it forms a lock certificate L for val in view v. The node updates its $lock \coloneqq L$ and sends a $\langle$Vote-2, $v, val\rangle$ message to all nodes. The equivocation check guarantees the uniqueness of the locked value in each view.

Commit step. Upon receiving $C \leftarrow n-f$ $\langle$Vote-2, $v, val\rangle$ messages, a node sends a $\langle$Commit, $C\rangle$ message. Upon receiving a $\langle$Commit, $C\rangle$ message, it commits and terminates.

View Change. A node sends $\langle$ViewChange, $v\rangle$ if it detects equivocation or times out in view v. Upon receiving $f+1$ ViewChange messages, a node stops sending Vote-1/Vote-2 messages in view v and sends its lock to all nodes. A node cannot immediately enter the next view but instead must wait $2d\Delta$ time before doing so. This is to give enough time for locks to propagate in the network.

5.3 Analysis

External validity is easily ensured if all correct nodes validate the proposed value before voting for it. In appendix C, we show how to achieve the strong unanimity validity in definition 4. We now focus on agreement and termination.

▶ **Lemma 14.** *If there exist $n-f$ $\langle$Vote-1, $v, val\rangle$ messages and $n-f$ $\langle$Vote-1, $v, val'\rangle$ messages in the same view v, then $val = val'$.*

Proof. Suppose for the sake of contradiction there exist a set S of $n-f$ $\langle$Vote-1, $v, val\rangle$ messages and a set S' of $n-f$ $\langle$Vote-1, $v, val'\rangle$ messages where $val \neq val'$.

Algorithm 3 BFT consensus protocol in granular partial synchrony for node i.

$v_i \leftarrow 0$, $lock \leftarrow \perp$ ▷ Initialize local view number and lock
enter view 1

upon entering view v **do**
  $v_i \leftarrow v$
  start $view\_timer \leftarrow timer((5+d)\Delta)$ ▷ Timer for changing view
  send $\langle \text{STATUS}, v, lock \rangle$ to L_v

upon receiving $S \leftarrow n - f$ $\langle \text{STATUS}, v_i, - \rangle$ **do**
  $val \leftarrow$ value in the highest lock in S, or $input_i$ if all locks in S are $\perp$
  send $\langle \text{PROPOSE}, v_i, val, S \rangle$ to all

upon receiving $\langle \text{PROPOSE}, v_i, val, S \rangle$ from L_{v_i} **do**
  if val matches the highest locked value in S or all locks in S are $\perp$ **then**
    echo $\langle \text{PROPOSE}, v_i, val, S \rangle$ to all
    start $vote\_timer \leftarrow timer(d\Delta)$ ▷ To detect equivocation

upon $vote\_timer$ expiring and no equivocation detected **do**
  send $\langle \text{VOTE-1}, v_i, val \rangle$ to all

upon receiving $L \leftarrow n - f$ $\langle \text{VOTE-1}, v_i, val \rangle$ **do**
  $lock \leftarrow L$
  send $\langle \text{VOTE-2}, v_i, val \rangle$ to all

upon receiving $C \leftarrow n - f$ $\langle \text{VOTE-2}, v_i, val \rangle$ or one $\langle \text{COMMIT}, C \rangle$ **do**
  send $\langle \text{COMMIT}, C \rangle$ to all
  commit val and **terminate**

upon receiving $\langle \text{PROPOSE}, v_i, val, - \rangle$ and $\langle \text{PROPOSE}, v_i, val', - \rangle$ where $val' \neq val$ **do**
  echo $\langle \text{PROPOSE}, v_i, val, - \rangle$ and $\langle \text{PROPOSE}, v_i, val', - \rangle$ to all
  send $\langle \text{VIEWCHANGE}, v_i \rangle$ to all

upon $view\_timer$ expiring **do**
  send $\langle \text{VIEWCHANGE}, v_i \rangle$ to all

upon receiving $VC \leftarrow f + 1$ $\langle \text{VIEWCHANGE}, v \rangle$ where $v > v_i$ **do**
  stop sending VOTE-1/VOTE-2 messages for views up to v
  echo VC to all
  echo $\langle \text{LOCKED}, lock \rangle$ to all
  wait $2d\Delta$
  enter view $v + 1$

upon receiving $\langle \text{LOCKED}, lock' \rangle$ **do**
  $lock \leftarrow$ higher lock between $lock$ and $lock'$
  echo $\langle \text{LOCKED}, lock' \rangle$ to all

Of the $n - f$ nodes whose VOTE-1 messages are in S, at least a set P of $n - 2f$ must be correct. By the condition in theorem 13, $P \to H$ where H is a set of $f + 1$ correct nodes. Due to quorum intersection, $S' \cap H$ must contain at least one node, which is correct. Let c' be this node. Since the graph is undirected, there exists $c \in S$ such that $c' \to c$.

Let t be the time c' starts its vote timer. At time t, c' also forwards the $\langle$PROPOSE, $v, val', -\rangle$ message to all nodes. By time $t + d\Delta$, c receives this message. Thus, c must have sent $\langle$VOTE-1, $v, val\rangle$ before time $t + d\Delta$. Otherwise, c would have detected leader equivocation and would not have voted. Then, c must have forwarded $\langle$PROPOSE, $v, val, -\rangle$ to all nodes before time t. c' receives this $\langle$PROPOSE, $v, val, -\rangle$ message before time $t + d\Delta$, which is before its vote timer expires. Thus, c' detects leader equivocation and would not have voted. This contradicts $c' \in S'$. ◀

▶ **Lemma 15.** *If some node commits val in view v, then any set of $n - f$ $\langle$VOTE-1, $v', val'\rangle$ messages (lock certificate) in view $v' \geq v$ must have $val' = val$.*

Proof. We prove this lemma by induction on view v'. The base case of $v' = v$ is straightforward by lemma 14. For the inductive step, suppose the lemma holds up to view $v' - 1$, and now we consider view v'. Suppose for the sake of contradiction that some node commits val in view v, and there exist $n - f > f$ nodes that send $\langle$VOTE-1, $v', val'\rangle$ messages for $val' \neq val$. A correct node will only send $\langle$VOTE-1, $v', val'\rangle$ if a proposal carries in view v' a set S of $\langle$STATUS, $v', -\rangle$ messages. Thus, there exists a subset $H \subseteq S$ of $n - 2f$ correct nodes which sent $\langle$STATUS, $v', -\rangle$. By the condition in theorem 13, $H \to Q$, where Q is a set of $f + 1$ correct nodes.

Since a node committed val in view v, there must exist some set $n - f$ nodes that sent $\langle$VOTE-2, $v, val\rangle$, of which a set R of at least $n - 2f$ are correct. Before sending $\langle$VOTE-2, $v, val\rangle$ messages, these correct nodes updated $lock \coloneqq (v, val)$ in view v. Sets Q and R intersect in at least one correct node. Let this node be q. Since the graph is undirected and $H \to Q$, there must exist a node $h \in H$ such that $q \to h$. By the induction hypothesis, any lock certificate from view v to $v' - 1$ must be for val. Since a node only updates its lock monotonically based on view numbers, node q must have a lock with view $\geq v$ for val. Let t_h be the time node h echoed $f + 1$ $\langle$VIEWCHANGE, $v'\rangle$ messages. By time $t_h + d\Delta$, node q must have received $f + 1$ $\langle$VIEWCHANGE, $v'\rangle$ messages. Node q will then echo a $\langle$LOCKED, $lock\rangle$ message to all nodes. This will be received by node h by time $t_h + 2d\Delta$. Node h will update its lock to be at least view v for val. Thus, from nodes in H, $L_{v'}$ receives at least one STATUS message for val with view $\geq v$. By the induction assumption, any lock certificate not for val must have view $< v$. Thus, no correct node sends $\langle$VOTE-1, $v', val'\rangle$, a contradiction. ◀

▶ **Theorem 16** (Agreement). *No two correct nodes commit different values.*

Proof. Let v be the smallest view in which a correct node commits some value, say val. By lemma 15, only val can receive $n - f$ $\langle$VOTE-1, $v\rangle$ messages in any view $v' \geq v$, so no other value can be committed by a correct node. ◀

▶ **Theorem 17** (Termination). *All correct nodes eventually decide.*

Proof. With round-robin leader election, correct nodes are elected leaders infinitely often. Thus, there must be a view v, after $GST + 2d\Delta$, whose leader is correct. We next prove that all nodes will decide and terminate in view v (if they don't decide earlier).

Let t ($t \geq GST + 2d\Delta$) be the first time some correct node enters view v. This correct node echoes $f + 1$ $\langle$VIEWCHANGE, $v - 1\rangle$ messages to all nodes at $t - 2d\Delta \geq GST$. All correct nodes will receive the new view certificate by time $t - 2d\Delta + \Delta$, wait $2d\Delta$ themselves,

and enter view v by time $t + \Delta$. Upon entering view v, they send $\langle$Status, $v, -\rangle$ messages to L_v. L_v receives $n - f$ $\langle$Status, $v, -\rangle$ messages by time $t + 2\Delta$, and send a $\langle$Propose, $v, -\rangle$ message to all nodes. All correct nodes will receive the $\langle$Propose, $v, -\rangle$ message by time $t + 3\Delta$ and start their vote timers. Since L_v is correct and does not equivocate, all correct nodes will send a $\langle$Vote-1, $v, -\rangle$ message by time $t + (3 + d)\Delta$. All correct nodes will receive $n - f$ $\langle$Vote-1, $v, -\rangle$ messages by time $t + (4 + d)\Delta$, and send a $\langle$Vote-2, $v, -\rangle$ message. All correct nodes will receive $n - f$ $\langle$Vote-2, $v, -\rangle$ messages and commit by time $t + (5 + d)\Delta$). Since a node's view timer is $(5 + d)\Delta$, and changing views requires $f + 1$ $\langle$ViewChange, $v\rangle$ messages, all correct nodes will remain in view v, commit and terminate in view v. ◀

6 Related Work

Necessary and sufficient conditions to solve consensus in all three classic timing models have been long established [27, 17, 15, 18, 9, 16]. There is also a large body of work on CFT and BFT consensus protocols in all three timing models. Our protocols adopt standard techniques from previous protocols such as quorum intersection [26, 29, 13], synchronous equivocation detection [23, 1, 2], and view synchronizers [11, 10].

Weaker models than synchrony have been suggested in the literature. Some of these are orthogonal to the timing model. A line of work studies consensus on *incomplete* communication graphs [34, 24, 25]. The mobile link failure model [33] allows a bounded number of lossy links. These models are orthogonal because they still need to adopt one of the classic timing models for the links that exist in the graph and are not lossy. The mobile sluggish model [19] allows temporary unbounded message delays for a set of honest nodes (the set can change over time). The sleepy model [30] allows a large fraction of nodes to be inactive. Both are models of node failures. Correct nodes that are not sluggish/sleepy are still assumed to have pair-wise synchronous links with each other.

The Visigoth fault tolerance (VFT) paper [31] proposes a timing model that consists of synchronous and asynchronous links. Their model assumes every node has asynchronous links to at most s correct nodes and synchronous links to the remaining nodes. For CFT, VFT requires $n - s \geq f + 1$, so every node must have at least $f + 1$ synchronous links. For BFT, VFT requires every node to have $n - s \geq 2f + 1$ synchronous links. In contrast, our graph conditions are weaker (less restrictive) in that they only require a set of $n - f$ nodes for CFT ($n - 2f$ correct nodes for BFT) to have synchronous paths to at least $f + 1$ nodes ($f + 1$ correct nodes for BFT). We additionally consider partially synchronous edges.

Another line of work that considers a mixture of links studies the minimal condition to circumvent the FLP [18] impossibility and solve consensus deterministically [20, 5, 8, 7, 4]. Many of these works [20, 5, 8] consider the harder setting of directed graphs, while we only consider undirected graphs. Since they focus on circumventing FLP, they only consider a mixture of asynchronous and partially synchronous links, but no synchronous links. Our main focus is to use synchronous links to achieve better fault tolerance than those under partial synchrony. But as mentioned, when $n > 2f$ for crash and $n > 3f$ for Byzantine, our "safety-critical" condition becomes vacuous, and our model degenerates to a mixture of partially synchronous and asynchronous links. In this context, our work establishes the minimum condition for circumventing FLP for CFT consensus in undirected graphs.

7 Conclusion

This paper introduces the granular synchrony model that considers a mixture of synchronous, partially synchronous, and asynchronous links to better capture the heterogeneity of modern networks. We present necessary and sufficient conditions for solving crash and Byzantine consensus in granular synchrony. Our results show that consensus is solvable in the presence of $f \geq n/2$ crash faults and $f \geq n/3$ Byzantine faults in granular synchrony, even though not all links are synchronous.

References

1 Ittai Abraham, Dahlia 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 (SP)*, pages 106–118, 2020. doi:10.1109/SP40000.2020.00044.

2 Ittai Abraham, Kartik Nayak, Ling Ren, and Zhuolun Xiang. Good-case latency of byzantine broadcast: A complete categorization. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*, PODC'21, pages 331–341, New York, NY, USA, 2021. Association for Computing Machinery. doi:10.1145/3465084.3467899.

3 Saksham Agarwal, Qizhe Cai, Rachit Agarwal, David Shmoys, and Amin Vahdat. Harmony: A congestion-free datacenter architecture. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, pages 329–343, Santa Clara, CA, April 2024. USENIX Association. URL: https://www.usenix.org/conference/nsdi24/presentation/agarwal-saksham.

4 Marcos K. Aguilera, Carole Delporte-Gallet, Hugues Fauconnier, and Sam Toueg. Communication-efficient leader election and consensus with limited link synchrony. In *Proceedings of the Twenty-Third Annual ACM Symposium on Principles of Distributed Computing*, PODC '04, pages 328–337, New York, NY, USA, 2004. Association for Computing Machinery. doi:10.1145/1011767.1011816.

5 M.K. Aguilera, C. Delporte-Gallet, H. Fauconnier, and S. Toueg. Consensus with byzantine failures and little system synchrony. In *International Conference on Dependable Systems and Networks (DSN'06)*, pages 147–155, 2006. doi:10.1109/DSN.2006.22.

6 Ailidani Ailijiang, Aleksey Charapko, Murat Demirbas, and Tevfik Kosar. Wpaxos: Wide area network flexible consensus. *IEEE Trans. Parallel Distrib. Syst.*, 31(1):211–223, January 2020. doi:10.1109/TPDS.2019.2929793.

7 Olivier Baldellon, Achour Mostéfaoui, and Michel Raynal. A necessary and sufficient synchrony condition for solving byzantine consensus in symmetric networks. In *International Conference on Distributed Computing and Networking*, pages 215–226. Springer, 2011. doi:10.1007/978-3-642-17679-1_19.

8 Zohir Bouzid, Achour Mostfaoui, and Michel Raynal. Minimal synchrony for byzantine consensus. In *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing*, pages 461–470, 2015. doi:10.1145/2767386.2767418.

9 Gabriel Bracha. Asynchronous byzantine agreement protocols. *Information and Computation*, 75(2):130–143, 1987. doi:10.1016/0890-5401(87)90054-X.

10 Manuel Bravo, Gregory Chockler, and Alexey Gotsman. Liveness and Latency of Byzantine State-Machine Replication. In Christian Scheideler, editor, *36th International Symposium on Distributed Computing (DISC 2022)*, volume 246 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 12:1–12:19, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.DISC.2022.12.

11 Manuel Bravo, Gregory Chockler, and Alexey Gotsman. Making byzantine consensus live. *Distributed Computing*, 35(6):503–532, 2022. doi:10.1007/S00446-022-00432-Y.

12 Christian Cachin, Klaus Kursawe, and Victor Shoup. Random oracles in constantipole: practical asynchronous byzantine agreement using cryptography (extended abstract). In

Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing, PODC '00, pages 123–132, New York, NY, USA, 2000. Association for Computing Machinery. doi:10.1145/343477.343531.

13 Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance. In *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, OSDI '99, pages 173–186, USA, 1999. USENIX Association. URL: https://dl.acm.org/citation.cfm?id=296824.

14 Shinyoung Cho, Romain Fontugne, Kenjiro Cho, Alberto Dainotti, and Phillipa Gill. Bgp hijacking classification. In *2019 Network Traffic Measurement and Analysis Conference (TMA)*, pages 25–32, 2019. doi:10.23919/TMA.2019.8784511.

15 D. Dolev and H. R. Strong. Authenticated algorithms for byzantine agreement. *SIAM J. Comput.*, 12(4):656–666, November 1983. doi:10.1137/0212045.

16 Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, 35(2):288–323, April 1988. doi:10.1145/42282.42283.

17 Michael J. Fischer, Nancy A. Lynch, and Michael Merritt. Easy impossibility proofs for distributed consensus problems. In *Proceedings of the Fourth Annual ACM Symposium on Principles of Distributed Computing*, PODC '85, pages 59–70, New York, NY, USA, 1985. Association for Computing Machinery. doi:10.1145/323596.323602.

18 Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, April 1985. doi:10.1145/3149.214121.

19 Yue Guo, Rafael Pass, and Elaine Shi. Synchronous, with a chance of partition tolerance. In *Advances in Cryptology – CRYPTO 2019: 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18–22, 2019, Proceedings, Part I*, pages 499–529, Berlin, Heidelberg, 2019. Springer-Verlag. doi:10.1007/978-3-030-26948-7_18.

20 Moumen Hamouma, Achour Mostéfaoui, and Gilles Trédan. Byzantine consensus with few synchronous links. In *Principles of Distributed Systems: 11th International Conference, OPODIS 2007, Guadeloupe, French West Indies, December 17-20, 2007. Proceedings 11*, pages 76–89. Springer, 2007. doi:10.1007/978-3-540-77096-1_6.

21 Owen Hilyard, Bocheng Cui, Marielle Webster, Abishek Bangalore Muralikrishna, and Aleksey Charapko. Cloudy forecast: How predictable is communication latency in the cloud?, 2023. arXiv:2309.13169, doi:10.48550/arXiv.2309.13169.

22 Toke Høiland-Jørgensen, Bengt Ahlgren, Per Hurtig, and Anna Brunstrom. Measuring latency variation in the internet. In *Proceedings of the 12th International on Conference on Emerging Networking EXperiments and Technologies*, CoNEXT '16, pages 473–480, New York, NY, USA, 2016. Association for Computing Machinery. doi:10.1145/2999572.2999603.

23 Jonathan Katz and Chiu-Yuen Koo. On expected constant-round protocols for byzantine agreement. In Cynthia Dwork, editor, *Advances in Cryptology - CRYPTO 2006*, pages 445–462, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg. doi:10.1007/11818175_27.

24 Muhammad Samir Khan, Syed Shalan Naqvi, and Nitin H. Vaidya. Exact byzantine consensus on undirected graphs under local broadcast model, 2019. arXiv:1903.11677.

25 Muhammad Samir Khan and Nitin Vaidya. Asynchronous byzantine consensus on undirected graphs under local broadcast model, 2019. arXiv:1909.02865.

26 Leslie Lamport. The part-time parliament. *ACM Trans. Comput. Syst.*, 16(2):133–169, May 1998. doi:10.1145/279227.279229.

27 Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, July 1982. doi:10.1145/357172.357176.

28 Iulian Moraru, David G. Andersen, and Michael Kaminsky. There is more consensus in egalitarian parliaments. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, SOSP '13, pages 358–372, New York, NY, USA, 2013. Association for Computing Machinery. doi:10.1145/2517349.2517350.

29 Brian M. Oki and Barbara H. Liskov. Viewstamped replication: A new primary copy method to support highly-available distributed systems. In *Proceedings of the Seventh Annual ACM*

Symposium on Principles of Distributed Computing, PODC '88, pages 8–17, New York, NY, USA, 1988. Association for Computing Machinery. doi:10.1145/62546.62549.

30 Rafael Pass and Elaine Shi. The sleepy model of consensus. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology – ASIACRYPT 2017*, pages 380–409, Cham, 2017. Springer International Publishing. doi:10.1007/978-3-319-70697-9_14.

31 Daniel Porto, João Leitão, Cheng Li, Allen Clement, Aniket Kate, Flavio Junqueira, and Rodrigo Rodrigues. Visigoth fault tolerance. In *Proceedings of the Tenth European Conference on Computer Systems*, EuroSys '15, New York, NY, USA, 2015. Association for Computing Machinery. doi:10.1145/2741948.2741979.

32 Fedor Ryabinin, Alexey Gotsman, and Pierre Sutra. SwiftPaxos: Fast Geo-Replicated state machines. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, pages 345–369, Santa Clara, CA, April 2024. USENIX Association. URL: https://www.usenix.org/conference/nsdi24/presentation/ryabinin.

33 U. Schmid, B. Weiss, and J. Rushby. Formally verified byzantine agreement in presence of link faults. In *Proceedings 22nd International Conference on Distributed Computing Systems*, pages 608–616, 2002. doi:10.1109/ICDCS.2002.1022311.

34 Lewis Tseng and Nitin Vaidya. Exact byzantine consensus in directed graphs, 2014. arXiv:1208.5075.

35 Tian Yang, Robert Gifford, Andreas Haeberlen, and Linh Thi Xuan Phan. The synchronous data center. In *Proceedings of the Workshop on Hot Topics in Operating Systems*, HotOS '19, pages 142–148, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3317550.3321442.

A BFT Consensus in Granular Asynchrony

We present a sufficient condition for solving BFT consensus in granular asynchrony.

▶ **Theorem 18.** *If (i) the condition in theorem 13 holds and (ii) for all F with $|F| = f$, there exists a node in graph $G' = (V - F, \diamond E)$, which has partially synchronous paths to f other nodes in G', then BFT consensus on graph $G = (V, E)$ can be solved deterministically under granular asynchrony.*

We have proved the necessity of condition (i) (for all algorithms) in Section 5.1. Condition (ii) was proven necessary in [7] for algorithms that work for the family of all graphs that satisfy the condition (i.e., graph-agnostic algorithms). If algorithms can be tailored to the graph, the tight condition for Byzantine consensus remains open.

A.1 Protocol

Next, we adapt our previous BFT consensus protocol in algorithm 3 from granular partial synchrony to granular asynchrony, assuming the condition in theorem 18 holds. This establishes the sufficiency of the condition.

As with our granular asynchrony CFT consensus protocol, we utilize condition (ii) in theorem 10 to guarantee that, when the correct node with partially synchronous paths to f other nodes in $G' = (V - F, \diamond E)$ is elected leader after GST, this leader will not be replaced and will make progress. To do so, we first require $n - f$ nodes to initiate a view change, of which at least $n - 2f$ must be correct. This way, because fewer than $n - 2f$ correct nodes are asynchronously connected to the leader, we just need to make sure that none of the f nodes the leader is connected to via partially synchronous paths initiates a view change.

We only describe the status and view change steps, since the rest of the protocol remains the same as algorithm 3.

Algorithm 4 BFT consensus protocol in granular asynchrony for node i.

1: $v_i \leftarrow 0,\ lock \leftarrow \perp$ ▷ Initialize local view number and lock
2: **enter** view 1

3: **upon** entering view v **do**
4: $v_i \leftarrow v$
5: **send** $\langle$Status, $v, lock\rangle$ to all

6: **upon** receiving $n-f$ $\langle$Status, $v_i, -\rangle$ messages where $i \neq L_v$ **do**
7: **echo** these $n-f$ $\langle$Status, $v_i, -\rangle$ to all
8: **start** *proposal_timer* $\leftarrow$ *timer*$(3d'\Delta)$ ▷ Timer before changing view

9: **upon** *proposal_timer* expiring and no leader proposal received **do**
10: **send** $\langle$ViewChange, $v_i\rangle$ to all

11: Proposal, Vote-1, Vote-2, Commit messages, $n-f$ ViewChange messages (instead of $f+1$), equivocation detection at all nodes, and Status messages at view leaders are processed the same way as in Algorithm 3

Status step. Upon entering a new view v, a node sends a $\langle$Status, $v, lock\rangle$ message to **all** nodes. When a node receives at least $n-f$ $\langle$Status, $v, -\rangle$ messages, it forwards this set of Status messages to all nodes and starts a timer with $3d'\Delta$ duration. The same propose, vote, and commit steps from algorithm 3 follow.

View change. A node suspects the leader is faulty if it does not receive a $\langle$Propose, $v, -, -\rangle$ message before its proposal timer (instead of view timer) expires. A view change certificate consists of $n-f$ $\langle$ViewChange, $v\rangle$ messages (instead of $f+1$ in algorithm 3). Upon receiving $n-f$ $\langle$ViewChange, $v\rangle$, a node carries out the same waiting period step from algorithm 1.

A.2 Analysis

The agreement and validity proofs are identical to the granular partial synchrony BFT case. We focus on termination.

▶ **Lemma 19.** *If no correct node ever terminates, then every correct node keeps entering higher views.*

Proof. Suppose for the sake of contradiction, there exists a correct node n_1, which never enters a higher view. Let v be the view n_1 is in. If any correct node ever enters a view $v' > v$, it must have echoed $n-f$ $\langle$ViewChange, $v'-1\rangle$ messages to all nodes. n_1 will eventually receive this set $n-f$ $\langle$ViewChange, $v'-1\rangle$ messages and enter a higher view, a contradiction. Thus, no correct node ever enters a view higher than v. Before entering view v, n_1 must have sent $n-f$ $\langle$ViewChange, $v-1\rangle$ to all nodes. All correct nodes will eventually receive this set of $\langle$ViewChange, $v-1\rangle$ messages, enter view v, and send a $\langle$Status, $v, -\rangle$ message. Eventually, correct nodes will receive $n-f$ $\langle$Status, $v, -\rangle$ messages and start their proposal timers. If n_1 receives $n-f$ $\langle$ViewChange, $v\rangle$ messages, it will enter view $v+1$, a contradiction. Thus n_1 never receives $n-f$ $\langle$ViewChange, $v\rangle$ messages. Then, there must be at least one correct node, n_2, which never sends $\langle$ViewChange, $v\rangle$, and instead echoes $\langle$Propose, $v, -, -\rangle$ to all nodes. Eventually, all correct nodes will receive

a $\langle$Propose, $v, -, -\rangle$ message and echo it. If a correct node detects leader equivocation, it will forward it to all correct nodes. n_2 will eventually receive the conflicting Propose messages and send a $\langle$ViewChange, $v\rangle$ message, a contradiction. Thus, no correct node will detect leader equivocation. Then, all correct nodes will send $\langle$Vote-1, $v, -\rangle$ messages to all nodes. Eventually all correct nodes will receive $n - f$ $\langle$Vote-1, $v, -\rangle$ messages, and send a $\langle$Vote-2, $v, -\rangle$ message. Eventually, n_1 will receive $n - f$ $\langle$Vote-2, $v, -\rangle$ messages, commit and terminate, a contradiction. ◀

▶ **Theorem 20.** *All correct nodes eventually terminate.*

Proof. Suppose for the sake of contradiction that some correct node never terminates. Observe that if one correct node terminates, it sends a Commit message and makes all correct nodes eventually terminate. Thus, no correct node ever terminates. By lemma 19, every correct node keeps entering higher views.

Eventually, there will be a first time after $GST + 2d\Delta$ that some correct node enters a view v such that (i) $L_v \notin F$, and (ii) L_v has paths to at least f other nodes in graph $G' = (V - F, \diamond E)$. Let C denote this set of nodes including L_v. We next prove no node in C will ever send $\langle$ViewChange, $v\rangle$.

Let p be the first node in C that enters view v, and let p enter view v at time $t > GST + 2d\Delta$. Observe that no node in C will send $\langle$ViewChange, $v\rangle$ before time $t + 3d'\Delta$ (proposal timer duration is $3d'\Delta$). Due to the condition in theorem 18, $n - |C| < n - f$. Thus, there will not be $n - f$ $\langle$ViewChange, $v\rangle$ messages before $t + 3d'\Delta$.

p sends $n - f$ $\langle$ViewChange, $v - 1\rangle$ messages at time $t - 2d\Delta > GST$. All nodes in C receive $n - f$ $\langle$ViewChange, $v - 1\rangle$ messages by time $t - 2d\Delta + d'\Delta$, enter view v by time $t + d'\Delta$, and stay in view v at least until time $t + 3d'\Delta$.

When a node $q \in C$ receives $n - f$ $\langle$Status, $v, -\rangle$ messages at time $t' > t$, q echoes these $n - f$ messages and starts its proposal timer. All nodes in C enter view v by time $t + d'\Delta$ and are ready to echo these $\langle$Status, $v, -\rangle$ messages by $t + d'\Delta$. L_v, which is in C, receives these $n - f$ $\langle$Status, $v, -\rangle$ messages by time $\max(t + 2d'\Delta, t' + d'\Delta) < t' + 2d'\Delta$. L_v sends a $\langle$Propose, $v, -\rangle$ message by time $t' + 2d'\Delta$ and it reaches q by time $t' + 3d'\Delta$, which is before q's proposal timer expires. Thus, q does not send $\langle$ViewChange, $v\rangle$. This establishes that no node in C will ever send $\langle$ViewChange, $v\rangle$.

Since $n - |C| < n - f$, there will never be $n - f$ $\langle$ViewChange, $v\rangle$ messages. Thus, no correct node ever enters a view higher than v. This contradicts lemma 19. ◀

B Comparison with [4]

[4] showed that a correct $\diamond f$-source is a sufficient condition for solving CFT consensus in a directed graph. A correct $\diamond f$-source is a correct node that has f outgoing fault-free paths that are eventually synchronous. [4] argued the potential optimality of their result by showing that every node being a $\diamond(f - 1)$-source is not sufficient for solving CFT consensus. Our results show that, at least in the case of undirected graphs, a correct $\diamond f$-source is not necessary. Our condition (ii) in theorem 10 is weaker and is sufficient.

To show our condition is weaker, we first prove that a correct $\diamond f$-source implies the condition (ii) in theorem 10. Let C be the connected component in $G' = (V, \diamond E)$ that the correct $\diamond f$-source belongs to. We have $|C| \geq f + 1$. Removing $F \cup C$ must leave at most $n - f - 1$ nodes in the remaining graph.

Next, Figure 2 shows an example of a graph that satisfies our condition but does not have a correct $\diamond f$-source. For this graph, if the adversary corrupts B and C, then there is no correct $\diamond f$-source since A only has a link to B and D only has a link to C. This graph,

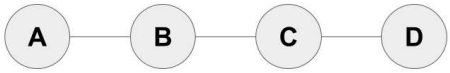

Figure 2 In this graph $n = 4$ and $f = 2$. Each edge represents a synchronous link and a missing edge represents an asynchronous link.

Algorithm 5 BFT Unanimity Validity.

```
1: v_i ← 0, lock ← ⊥                                  ▷ Initialize local view number and lock
2: inputs ← {}
3: echo ⟨Input, input_i⟩ to all
4: start input_timer ← timer(2dΔ)
5: upon receiving m ← ⟨Input, input_j⟩ do
6:     echo m
7:     inputs ← inputs ∪ {m}
8: upon input_timer expiring do
9:     send ⟨Forward-Inputs, inputs⟩ to all
10: upon receiving FI ← n − f ⟨Forward-Inputs, inputs⟩ do
11:     if having received I ← f + 1 ⟨Input, val⟩ messages in FI then
12:         lock ← I
13: enter view 1
```

however, satisfies the condition (ii) in theorem 10. If $|F| = 0$, removing the largest connected component (the entire graph) leaves 0 nodes, satisfying the condition. For any choice of F with $|F| = 1$, the largest connected component after removing F must be of size at least 2. Thus, there will be at most 1 remaining node, satisfying the condition. For any choice of F such that $|F| = 2$, the largest remaining connected component must be of size at least 1. Thus, there will be at most 1 remaining node, satisfying the condition.

C BFT Unanimity Validity

In this section, we give a way to convert our BFT algorithms from external validity to strong unanimity validity. The idea is to try to have nodes lock before starting the first view, and if all correct nods have the same input, then that input is the only lock.

▶ **Lemma 21.** *If all correct nodes have the same input, then all correct nodes will lock on this value before entering view* 1*, and any lock in view* 0 *must be for val.*

Proof. In view 0, all correct nodes send their inputs and echo other nodes' inputs they receive (using Input and Forward-Inputs messages) before their input timer expires in $2d\Delta$ time. For any two correct nodes p and q such that $p \to q$, p will receive q's input before p's input timer expires. Similarly, q will receive p's input before q's input timer expires. Consider any correct node c. Node c will eventually receive a set A of $n - f$ ⟨Forward-Inputs, *inputs*⟩ messages. Among them, a subset B of $n - 2f$ are from correct nodes. By the condition in theorem 13, $B \to C$ where C is a set of $f + 1$ correct nodes. Since every node in B waits $2d\Delta$ before sending a Forward-Inputs message, this is sufficient time for each node in C

to receive an input from some node in B and also sends its input to that node in B. Thus, B will contain the input values from C, a set of $f + 1$ correct nodes. If all correct nodes have the input val, node c must receive at least $f + 1$ $\langle \text{INPUT}, val \rangle$ messages, and there are at most f INPUT messages for a different value (from f Byzantine nodes). Therefore, every correct node will set its lock to $I \leftarrow f + 1$ $\langle \text{INPUT}, val \rangle$ in view 0, and any lock in view 0 must be for val. ◀

▶ **Lemma 22.** *If all correct nodes have the same input, then any lock in view $v \geq 0$ must be for val.*

Proof. The base case is established by lemma 21. Now assume the lemma holds for all $v - 1$, and consider view v. Suppose for the sake of contradiction a lock forms for $val' \neq val$. L_v must have proposed $val' \neq val$. By the induction assumption, any lock must be for val. Thus, L_v must have received $S \leftarrow n - f$ STATUS messages where all locks are $\perp$. By lemma 21, all correct nodes will lock on val before entering view 1. The set S must contain a STATUS message from at least one correct node. This correct node will at least have a lock in view 0 or higher, and thus its STATUS message will not have $lock = \perp$, a contradiction. ◀

▶ **Theorem 23.** *If all correct nodes have the same input, then only that value can be decided.*

Proof. By lemma 22, any lock must be for val, the input of the correct nodes. Only locked values can be decided. Validity then follows from termination. ◀